Connecting Knowledge bases with databases: a complete mapping relation

Alfredo Goñi José Miguel Blanco Arantza Illarramendi

Facultad de Informática, Universidad del País Vasco.
Apdo. 649, 20.080 San Sebastián. SPAIN
e-mail: jipileca@si.ehu.es

Abstract

In this paper we present a mapping relation between relational databases and terminological knowledge based systems. We define precisely its syntax and semantics. Moreover we demonstrate its completeness property in the sense that the knowledge base queries can be answered, so every query that could be formulated over the knowledge base can be translated into a query over the relational database where in fact are stored the data.

1 Introduction

It is widely recognized that knowledge based systems play and are going to play a very important role in the near future. However, most organizations possess their data stored in databases, data that they need for their every day work. In order to permit them to continue dealing with the existing databases while taking advantage of the features provided by the knowledge based systems, different tools are being designed. Those tools do not alter the daily work but nevertheless permit organizations to incorporate the new technology.

It is clear that database and knowledge based systems have different objectives, but their coalition provides many practical benefits which includes: the possibility of dealing efficiently with persistent data stored in databases under the representational power of the knowledge based systems [Int87] and the coexistence of different autonomous databases, under a global integrated view (Federated Database Systems) defined using a knowledge based system [NGG89]. In our case, we have designed a Federated Database System that supports on the one hand, interoperability between different and heterogeneous relational databases using for that a particular kind of knowledge based system -a terminological system- , and on the other, allows knowledge based applications to access data stored in relational databases [BIG94].

In this paper we concentrate in one aspect of the Federated Database System, the mapping information that correlates data elements of the relational schemata and of the terminologies. Our goal is twofold. First, to define precisely the way in which that mapping information can be defined: its syntax and semantic. That task is not trivial taking into account that two models, with different expressive power, are being related. In fact our search of the literature shows that the relation for mapping between these different type of systems has not been studied in detail, only few works such as [BB93], [Dev93] have considered it partially. Second, to show that for every query formulated correctly on the terminology, defined over one or more relational databases, there exists a corresponding relational algebra expression. In this case we try to define the completeness property in this context. The intent of many definitions for completeness, in the area of databases, has been to quantify the expressiveness of query languages [Ull88]. Knowing that the query languages offered by the systems that we want to connect have different expressive power, our definition of completeness tries to guarantee that any query formulated over the terminology will have a translation to queries over the databases because the data are only stored in the databases. Completeness in this sense is nearer to the operational equivalence concept defined in [ME92]: *"Let S1 and S2 be two database systems with operation sets F1 and F2 respectively, F1=*$\{f_{11},\ldots f_{1n}\}$*, F2=*$\{f_{21},\ldots f_{2m}\}$*. If for every* f_{1i} *in set F1, there exists a set of one or more operations in F2 which, when executed in some fixed order, have the same effect as* f_{1i} *on any database state, then system S2 is said to be operationally equivalent to S1"*. In our case, the operations for F1 and

F2 will be those used to formulate queries over the terminology and over the relational schemata respectively.

In this point it is interesting to mention that we are not going to include in the mapping information definition the distinction between the case that the terminology is supported by only one database and the case that it is supported by several ones. That aspect, that includes the management of the distribution and the access way to the databases is relevant when the mapping is exploited but not in the moment of its definition.

Lastly, we bring the reader's attention to the wide spectrum of possible transformations that the mapping information covers, permitting one to exploit all the features provided by the terminological system when creating a terminology from a relational one.

A. Terminological Systems

Terminological systems, also known as systems based on Description Logics, are descendants of KL-ONE[BS85]. There exist many systems of this type [WS92] such as BACK, LOOM and CLASSIC some of them are commercial products.

Terminological systems provide languages that allow for describing concepts (set of instances) and roles (relationships between instances or between instances and simple types as number, string, etc.). The concept and role descriptions are variable-free and composed by different constructors. Concept descriptions are conjunctions of some primitive concepts and some constructors (atleast, atmost, all, **:**). Concept descriptions may describe *defined* concepts if they specify necessary and sufficient conditions that any instance must verify in order to belong to the described concept (for example *assistant_professor* := *teacher and title:'Assistant'*) or concept descriptions may describe *primitive* concepts if they specify only necessary conditions but not sufficient (for example *teacher* :< *person*). Terminological systems may reason by looking up the descriptions and decide if a description *subsumes* another one. They can also make a *classification* of the concepts into a hierarchy by checking the subsumption relationships among concept descriptions. In terminological systems, descriptions can be seen as views or as queries, and therefore it is also possible to reason with the queries. In [Bor] it appears an interesting survey of applications of Terminological systems for Data Management. It is also shown how enhanced access to data and knowledge can be achieved by using descriptions for schema design and integration, queries, answers, updates, rules and constraints.

BACK [PSKQ89] is a concrete terminological system that we have used. However, the definitions of this paper are valid for any terminological system that support the same constructors.

B. Federated Database System Four main modules appear in our Federated Database System architecture: *Translator*, *Integrator*, *Query Processor* and *Monitor*.

- Translator Module. It produces, with the help of the Person Responsible for the Integration, a knowledge base terminology from a conceptual schema of a component database.
- Integrator Module. It produces a federated schema by integrating a set of knowledge base terminologies previously obtained by the Translator module. This federated schema is also represented as a knowledge base terminology[1].
- Query Processor Module. It obtains the answer to the user queries over the federated schema by accessing to the databases. This module has two kinds of components, the Global Query Processor and the Local Query Processor. The first one optimizes knowledge base queries (using the classification mechanism of terminological systems, syntactic, semantic and caching optimization techniques), finds out which databases contain the requested information, decomposes a query into subqueries that will run over different databases, and reconstructs the answer from the different results obtained for the subqueries. The goals of the second process are to make local optimizations and to find the answer for the subqueries.
- Monitor Module. It responds automatically, i.e., without user intervention, to design changes made in the schema of a component database that affect the federated schema. Two components, the Modification Manager and the Mapping Modification Manager constitute this module. The former one detects and identifies relevant database design changes, while the latter one reestablishes the consistency of the federated system. Therefore, this module deals with the important problem of meta data consistency among the federated schema and the autonomous database schemata.

[1]Hereafter we call it terminology

The main contributions of this Federated Database System are the incorporation of an active behavior that allows for maintaining automatically the consistency between database schema definitions and terminological definitions, and the possibility of taking advantage of the classification mechanism provided by the terminological systems in order to obtain a greater automatization of the integration process.

In this system the mapping information is first created by the Translator module, next modified by the Integrator and it is exploited by the Query Processor in order to find the answer of queries formulated over the terminology on the underlying databases.

In the rest of this paper we present first the types of links that can be defined between the main data elements that appear in the terminology and those of the relational model. Next we explain the syntax and the semantics of the mapping information. And finally, we show that for every query formulated over the terminology there is a corresponding query in terms of the relational schemata support of the terminology.

2 Link types for concepts and roles

In this section we present the problem of defining the mapping between a terminology and a relational schema, or what is the same, between the main data elements that appear in the terminology (concepts and roles) and those of the relational model (relations and attributes).

A relational database is composed by a relational schema and its extension. The schema is in general static, while the extension changes more often producing the different *states* of the database. Our goal is to link the terminological and relational data elements in such a way that, given a mapping definition and one state of a relational database, the instances of the terminological concepts as well as the role values for those instances can be determined. That means that the mapping information must permit that the terminological system provides the same services as when there does not exist an integration.

In the following we show the ideas in which we are going to base the support for instances, concepts and roles respectively.

2.1 The instances

Dealing with a terminological system when a concept instance is created, the system is in charge of assigning an identifier that permits its unique identification. However, when a terminological-relational association is established, there exists a relation among the content of the database and the instances, in such a way that to one state of the database corresponds the existence of a set of instances.

The mapping that we are going to define is based on the idea that each instance that belongs to the extension of the terminology is supported in a value stored in the database. That value verifies the following properties:

- Its elimination of the database is equivalent to the deletion of the corresponding instance.
- Its permits the unique identification of the instance.
- Its allows obtaining the values that take the roles of the instance that it identifies (values that are also stored in the underlying databases).

2.2 The concepts

A concept groups individual elements of the real world. For example, the concept *teacher* represents the set of teachers of a faculty. Therefore, it can be established a mapping between a concept and a relation directly, because, in general a relation also groups a set of objects (each one represented by a tuple) that share some features.

The simplest type of link that can be defined is one that relates a concept to a basic relation. The semantic of this link will be that **there exists a concept instance for each tuple of the basic relation**. For example, consider the relation *TEACHER* in figure 1, the link of this relation with the concept *teacher* in figure 2, means that for each tuple in the relation *TEACHER* there is one instance of the concept *teacher*.

However, in many situations it can happen that only a subset of tuples of a basic relation, which satisfy some conditions, determine concept instances. For example, suppose that we are

TEACHER(ID#,Reg#,SS#,Name,Addr,Phone#,Title,Hours#,Dept)
ADMINISTRATIVE_STAFF(ID#,Reg#,Contract_type,Level)
RESEARCH_ASSISTANT(ID#,Name,Addr,Phone#,Start_Date,Final_Date,Type,Salary)
STUDENT(ID#,Exp#,Name,Addr,Year)
PROJECT(Code,Name,Leader,Budget,Start_Date,Final_Date)
SUBJECT(Sub#,Name,Type,Credits,Responsible,Dept)
PRERREQUISITE(Sub#_Pre,Sub#_Post)
DEPARTMENT(Dept#,Name,Head,Secretary)

Figure 1: Schema of a relational database

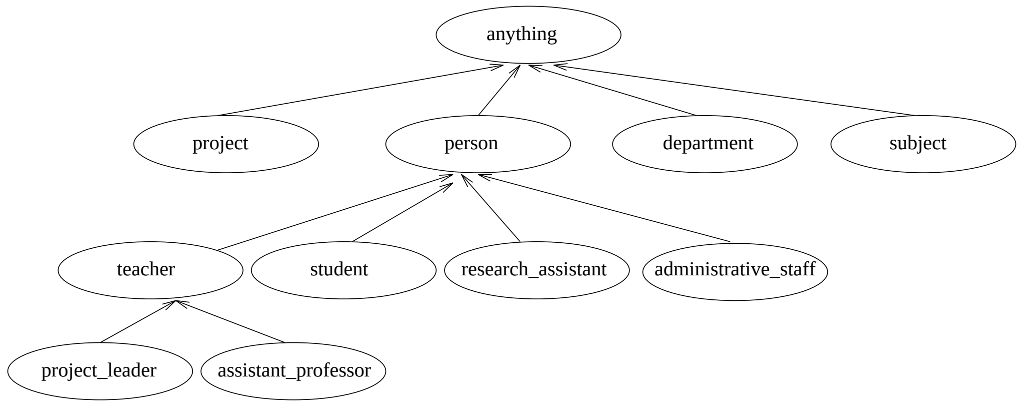

Figure 2: Concepts of the terminology corresponding to schema in figure 1

interested in defining the concept *assistant_professor* that has been defined in such a way that there is an instance of this concept for each tuple of the relation *TEACHER* that satisfies the condition *Title='Assistant'*. Therefore, we generalize the previous definition by allowing a link to be established between a concept and a relation that can be either basic or derived. The semantic of the link is redefined to mean that **there exists a concept instance for each tuple of the relation**[2].

In other contexts, the need arises for several tuples of a relation to represent one concept instance. This means that there exists a concept instance for each different value that takes one (or more) attributes of a relation. For example, there exists an instance of the concept *project-leader* for each different value that the attribute *Leader* takes in table *PROJECT*.

As a result of situations such as the previous one, we could also find cases where a tuple participates in the support of more than one instance. For example, a tuple of the relation *PROJECT* implies the existence of an instance of the concept *project* and another one of the concept *project-leader*.

Finally, several relations may determine the instances of a concept. For example, the concept *person* is linked with the relations named *TEACHER, STUDENT, RESEARCH_ASSISTANT and ADMINISTRATIVE_STAFF* from the considered schema, and there will be an instance of *person* for each tuple of each relation. In this case, a link will be established between a concept and a set of relations.

In summary a link can be established between a concept and one (or several) relation attribute, in such a way that there will exist a concept instance for each different value that the attribute takes for the tuples of that relation. In the case that the attribute is a key attribute, there will be an instance for each tuple. Moreover, a concept can be based on more than one relation. In that case, there will be an instance for each different value taken by the corresponding attributes for each one of the relations.

[2]Relation refers to a basic relation as well as to a derived one.

2.3 The roles

So far we have presented the link types for a concept; we now concentrate on the role values. Roles represent binary relationships between concept instances and other instances or values. A role can be seen as a predicate with two arguments: the *domain* (the concept with which is associated) and the *range* (the type of values that it can take).

Example: The roles *Responsible* and *Responsible_of* can be described as
Responsible :< *domain(subject) and range(teacher)*,
Responsible_of :< *domain(teacher) and range(subject)*.
The role *Responsible* is associated with the concept *subject* and takes as values instances of the concept *teacher*. The role *Responsible_of* is the inverse role of *Responsible*.

Therefore, the extension of a role can be seen as a set of pairs *(i,v)* where i corresponds to an instance identifier and v is an atomic value or an instance identifier. When the role is multivalued there can exist more than one pair *(i,v)* for one instance. On the contrary, if an instance does not take any value for one role then no pair will be associated to it.

In the example of figure 3 the concepts *teacher* and *subject* related through the roles *Responsible* (it corresponds to a teacher responsible of a subject) and *Responsible_of* (it corresponds to the subjects for which a teacher is responsible of) are included. The extension of the role *Responsible_of* corresponding to the state represented in the previous figure is: {(T2,S3),(T3,S1),(T3,S2)}. For the same state, the extension of the role *Responsible* is {(S1,T3), (S2,T3),(S3,T2)}.

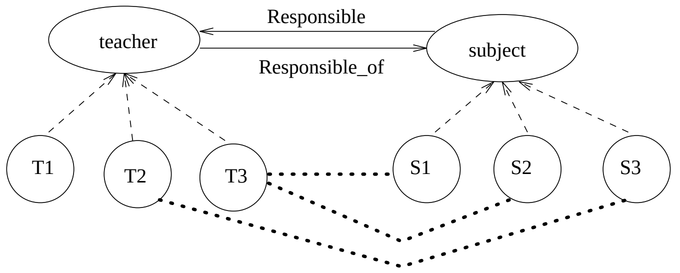

Figure 3: Roles Responsible and Responsible_of in the example terminology

To see the extension of a role as a set of pairs it will allow the definition of the role support as a relation with two attributes (or groups of attributes). The first one will correspond to the values that identify the instance and the second one to the values that the role take. So, the role *Responsible* of the previous example can be associated with the attributes *Sub#* and *Responsible* of the relation *SUBJECT* defined in the schema of figure1.

In general, the mapping information of a role is based on the mapping information of the concept to which it belongs. Therefore, the simplest type of link that can be defined for a role is one that relates the role to an attribute of a relation that supports the concept to which it belongs. Two different situations can arise:

- There exists an instance for each relational tuple. In this case, each role value will be the corresponding attribute value, excluding the null values for which there is no image. For example, the role *Title* of *teacher* can be linked to the attribute *Title* of *TEACHER*.
- There exists an instance for a set of tuples. In this case, the role will take as values all the distinct values that correspond to the attributes for the set of tuples, excluding the null values. For example, the role *Dept_name* associated with *department* can be linked to the attribute *Dept* of *TEACHER*.

However, a role may also be based on an attribute that does not appear in the relation that supports the concept to which the role belongs. For this case, it is necessary to establish a link between the role values obtained from the attributes and the instances in which the role takes part. For example, the role *Leaders* associated with *teacher* will be linked with the attribute *PROJECT.Leader* of the derived relation $TEACHER \bowtie_{ID\#=Leader} PROJECT$. In this way the link between instances of *teacher* and the value of the role *Leader* is established through the join of two relations.

ID#	Name	Address	Phone#	StartingDate	...
18123456	Pedro Alkiza	Arrasate, 10, 4	210989	1/10/92	
11234568	Marta Vadillo	Gernika, 18, 1	202020	1/10/93	
14235681	Ander Urizar	Getaria, 8, 2	null	1/10/93	
22235681	Pedro Pérez	Isabel II, 8, 2	451010	1/10/92	

Figure 4: Tuples in the relation *RESEARCH_ASSISTANT*

ID#	Reg#	ContractType	Level
81234561	Tomás Solé	Permanent	11
62345681	Isabel Uria	Temporal	11
43235681	Alfredo Alonso	Permanent	14
16589022	Pedro Pérez	Permanent	16

Figure 5: Tuples in the relation *ADMINISTRATIVE_STAFF*

3 Formalization of the mapping information

The goal of this section is to define in a rigorous way the terms in which we are going to establish the mapping terminological-relational. For this reason first, we introduce a set of basic principles with respect to the association between values and the objects of the real world that they represent. Those principles are used then to define the support of concepts and roles. Finally, backing in the definition of concept and role support we characterize the *extension function*, defined by a support function that links a terminology with a relational schema.

3.1 Basic Definitions

When a mapping between a terminology and a relational schema is established, an association among the data elements of the terminology (concepts and roles) an a set of values stored in the relational database is defined. However, the extension of a concept or a role is something more than a set of values: it represents a set of objects and relations taken out from the reality. That is, finally a value has an interpretation within the context in which it is used. Taking into account that in our context (database integration) can coexist different conceptions or representations of the same object, we are obliged to reason about the objects themselves (in addition of reasoning about the values that represent the objects) in order to establish the meaning of the mapping. For this reason we introduce the notion of *Real World State* (RWS). This notion has been used with different shades in distinct integration works [LNE89], [SPD91]. In our case we use it with the following goal: to define the constraints that a mapping between a terminology and a relational schema must verify.

In the following we describe the notion of RWS of a value and of a set of values, as well as the notions of *ambiguity* and *redundancy*.

Definition 1 *Let o be a real world object that can be identified in different ways in one or more contexts. If v is a value that identifies o in a particular context, then we will say that $o \in RWS(v)$.*

Definition 2 *Let v be a value, we will say that v is* non-ambiguous *in a particular context if it represents a unique real world object in that context.*

Example 1 *By examining the extensions of relations* RESEARCH_ASSISTANT *and* ADMINISTRATIVE_STAFF *(figures 4 and 5), it can be seen that* 'Pedro Pérez' *is a value that represents two different persons. Therefore, it is ambiguous in this context.*

Definition 3 *Let V be a set of values $v_1, \ldots, v_n$; we define the RWS of V as*

$$RWS(V)=RWS(v_1)\cup\ldots\cup RWS(v_n)$$

Definition 4 *A set V of values is* redundant *if it contains two values that represent the same real world object.*

$$V \text{ is redundant} \Leftrightarrow \exists v_i, v_j \in V\ (v_i \neq v_j \wedge RWS(v_i) = RWS(v_j))$$

Example 2 *The set of values* {Pedro Alkiza,18123456,...} *, formed by values extracted from the extension of* RESEARCH_ASSISTANT*, is redundant, because there exist at least two different values (*Pedro Alkiza *and* 18123456*) that represent the same person.*

3.2 Concept support

In this subsection we introduce the formal specification of the mapping between a concept and a relational schema. This mapping, to which we call *concept support*, will have to verify several requirements in order to be able to be valid: lack of redundancy and ambiguity.

Definition 5 *Let E be a relational schema composed by a set of relations, a* basic concept support *S_b defined upon E is a triple $<R,(atr_1,\ldots,atr_n),T>$, where R is, either a basic relation of E, or a derived relation upon E; $atr_1,\ldots,atr_n$ are attributes of R; and $T = D_1 \times \ldots \times D_n$, where D_i is the domain of the attribute atr_i for all i between 1 and n.*

Example 3 *The basic supports corresponding to the examples used in section 2 are:*

- *<TEACHER,(ID#),integer> for the concept* teacher*.*
- *<$\sigma_{Title='Assistant'}$(TEACHER),(ID#),integer> for the concept* assistant_professor*.*
- *<PROJECT,(Leader),integer> for the concept* project_manager*.*

A basic concept support S_b defines a set of values for a state e of E[3],

$$S_b(e)=\{v \in T \mid \exists t \in e(R) \text{ s.t. } (t[atr_1],\ldots,t[atr_n])=v\}$$

$$S_b(e) \subseteq T$$

Example 4 *Let S_b = <ADMINISTRATIVE_STAFF,(ID#),integer> be the basic support and e the extension represented in figure 5, then*

$$S_b(e) = \{81234561, 62345681, 43235681, 16589022\}$$

The RWS defined by a basic concept support S_b for a state e of E is the RWS of the set of values that S_b defines for this state.

$$RWS(S_b,e)=RWS(S_b(e))$$

Definition 6 *Let E be a set of relations, a* concept support *S defined upon E[4] is a list of basic concept supports $S_{b1},\ldots,S_{bn}$ defined upon E.*

Example 5 *Let E be the schema of figure 1, the concept support for* person *can be defined as the list formed by the following basic concept supports:*

<TEACHER,ID#,integer>
<ADMINISTRATIVE_STAFF,ID#,integer>
<STUDENT,ID#,integer>
<RESEARCH_ASSISTANT,ID#,integer>

A concept support S defines a set of values for a state e of E. We will refer to this set of values as $S(e)$. $S(e)$ is the union of the values defined by the basic supports that compose S for e.

$$S(e)=S_{b1}(e)\cup\ldots\cup S_{bn}(e)$$

$$S(e) \subseteq T_1\cup\ldots\cup T_n$$

The RWS defined by a concept support S for a state e of E is the RWS of the set of values that define S for e.

[3]We will use the notation $e(R)$ to refere to the extension of R for the state e of E, t will be used to denote a tuple and $t[atr]$ will refer to the value that the attribute atr takes for the tuple t.

[4]Hereafter, in order to abbreviate, it will be supposed that a support S is defined upon a set of relations E.

$$\text{RWS(S,e)=RWS(S(e))}$$

Definition 7 *A concept support S is* well formed *if for all state e of E, $S(e)$ is non-redundant and for all $v \in S(e)$ it is verified that v is not ambiguous.*

Our goal when introducing the notion of well formed support is to restrict the types of valid mappings between a terminology and relational schemata to those that define a biyective function between the concept instances and the values that represent them.

3.3 Role support

In the same way as it has been done for concepts, we will define in the following the *role support.* What it is new with respect to the concept support is the possibility of including a function in the role support. Its goal is to permit changes of representation of scale, of type, etc., that extend the capabilities of the integration.

Definition 8 *Let E a relational schema composed by a set of relations, we define a* basic role support *S_{b_r} as a tuple*

$$<R_{rl},(atrd_1,\dots,atrd_n),(atrn_1,\dots,atrn_m),T_C,f_{rl},T_r>$$

where R_{rl} is, either a basic relation of E, or a derived relation upon E; $atrd_1,\dots,atrd_n$ and $atrn_1,\dots,atrn_m$ are attributes of R_{rl}; $T_C=T_1\times\dots\times T_n$, where T_i is the domain of the attribute $atrd_i$ for all i between 1 and n; f_{rl} is a function with definition f_{rl}:$D_1 \times \dots \times D_m \rightarrow T$, where D_j is the domain of the attribute $atrn_j$ for all j between 1 and m and T is the range of the role. Finally, $T_r=D_1\times\dots\times D_m$. When a basic role support does not include a function then T is equal to T_r.

As we will see in the following section, in order to establish the correspondence between BACK descriptions and relational algebra expressions, the function f_{rl} will have to verify two restrictions. The first one is that its inverse must be known. The second one is that f_{rl} has to be in *strictly ascending* order with respect to the order defined over T_C.

Example 6 *The tuple <RESEARCH_ASSISTANT,(ID#),(Name),integer,-,string> is a simple example of basic role support (corresponding to the role Name).*

For a state e of E, a basic role suport S_{b_r} defines a set of pairs of values of the form:
$S_{b_r}(e)=\{<v, v_r> \mid \exists t_r \in e(R_{rl}) t.q.((t_r[atrd_1],\dots,t_r[atrd_n])=v \wedge$
$f_{rl}(t_r[atrn_1],\dots,t_r[atrn_m])=v_r)\}$

Example 7 *For the state introduced in figure 4 the set of pairs defined by the basic role support of Name would be:*

(<18123456,Pedro Alkiza>,<11234568,Marta Vadillo>,
<14235681,Ander Urizar>,<22235681,Pedro Pérez>)

The RWS defined by a basic role support S_{b_r} for a state e of E is the set of pairs of real world objects represented by the set of pairs of values corresponding to $S_{b_r}(e)$.

$$\text{RWS}(S_{b_r},e)=\{<o, o_r> \mid \exists < v, v_r > \in S_{b_r}(e) \wedge RWS(v) = \{o\} \wedge RWS(v_r) = \{o_r\}\}$$

Definition 9 *Let E be a set of relations, we define a* role support *S_r upon E as a list of basic role supports $S_{b_{r1}},\dots,S_{b_{rn}}$.*

Example 8 *The role support* Name *for the concept* person *that has been previously introduced is formed by the following elements:*

<TEACHER,(ID#),(Name),integer,-,string>
<ADMINISTRATIVE_STAFF,(ID#),(Name),integer,-,string>
<STUDENT,(ID#),(Name),integer,-,string>
<RESEARCH_ASSISTANT,(ID#),(Name),integer,-,string>

Definition 10 *The RWS defined by a role support S_r for a state e of E is the union of the RWS for e of the basic role supports that compose S_r.*

$$RWS(S_r,e)=RWS(S_{b_{r1}}(e)) \cup \dots \cup RWS(S_{b_{rn}}(e))$$

3.4 Extension defined by the mapping information

By the moment, we have described the way in which the concepts and roles can be mapped with a relational schema. *A mapping between a terminology and a relational schema will be defined as the set of concept and role supports.* A mapping between a terminology and a relational schema is valid if verifies some restrictions that we will explain in the following. Finally, we will describe the extension of a terminology that defines the mapping information.

Definition 11 *Let T be a terminology and let E be a set of relations, we define a* support function *F upon T as an association among concepts and roles of T and concept and role supports defined upon E.*

F = {<ter, S_{ter}> | ter is a concept or a role of T and S_{ter} is, respectively, a well formed concept support or a role support defined upon E }

Definition 12 *Let T be a terminology, E is a set of relations and F a support function that defines a mapping between T and E. We will say that F is* well formed *if for each C_1, C_2 concepts of T with supports S_{C_1}, S_{C_2} in F, for each role r of T with support S_r in F, it is verified that:*

- $\forall e(E) \forall v_1 \in S_{C_1}(e(E)) \forall v_2 \in S_{C_2}(e(E))(v_1 = v_2 \leftrightarrow EMR(v_1) = EMR(v_2))$

  In other words, there does not exist a state where the same value identifies two instances that represent two different real world objects.

- *If the domain of the role r is the concept C_1:*

  $$\forall e(E) \forall < i, v > \in S_r(e(E))(i \in S_{C_1}(e(E)))$$

  It enforces that for each pair belonging to the extension defined by a role support, the left value (corresponding to an instance identifier) belongs to the extension of the concept that has that role as domain.

- *If the range of the role r is the concept C_2:*

  $$\forall e(E) \forall < i, v > \in S_r(e(E))(v \in S_{C_2}(e(E)))$$

  When a role has a concept as range, it enforces that for each pair belonging to the extension defined by a role support, the right value (corresponding to an instance identifier) belongs to the extension of the concept that has that role as range.

In order to clarify the meaning of the definition, we will give three examples that do not verify these restrictions (corresponding to a support function that **is not** well formed), one for each requirement of the previous definition:

- *'Pedro Pérez'* belongs to the set of defined values, for a particular extension e, for $S_{research\_assistant}$ and for $S_{administrative\_staff}$, but it represents two different persons.
- The role domain *Name* is *person* and the pair $< 12121212, Ana\ Luesma >$ belongs to $S_{Name}(e)$ but, however, 12121212 does not belong to $S_{person}(e)$.
- The role *Manager* has *project* as domain and *teacher* as range. The pair $< DIPU10, 23232323 >$ belongs to $S_{Manager}(e)$ but however 23232323 does not belong to $S_{teacher}(e)$.

Definition 13 *A support function F that relates a terminology T with a set of relations E defines an extensional function ε for each state $e(E)$.*

Let C and r be a concept and a role of a terminology T:

- *$\varepsilon[C]=S_C(e(E))$ if and only if $<C,S_C>\in F$*
- *$\varepsilon[r]=S_r(e(E))$ if and only if $<r,S_r>\in F$*

Definition 14 *Let F be a support function that relates a terminology T with a set of relations E. We say that F* satisfies *T if for each state $e(E)$ the function ε, defined by F, satisfies[5] T.*

The notion of satisfaction *is important, because it defines the validity of a support function. In order to illustrate it, let us see the following case:*

Let us suppose that a terminology includes the concept intern_project, *described as a concept that is a specialization of the concept* project. *This terminology is bound with the set of relations by a support function F that, for a particular state e_0 of E, defines an extensional function*

[5] In BACK it is said that an extension *satisfies* a terminology when both are consistent between them [PSKQ89].

```
<concept> ::= <concept_name>
           |  atleast '(' number ',' <role> ')'
           |  <role> ':' <expression>
           |  <concept> and <concept>
           |  <concept> and atmost '(' number ',' <role> ')'
           |  <concept> and not '(' <concept> ')'
           |  <concept> and all '(' <role> ',' <conceptual_type> ')'
```

Figure 6: Brief syntax of concept descriptions

- ε[project] = {DIPU12,DIPU16,ESPR06,GOVA8,INT10}
- ε[intern_project] = {INT10,INT12}

This extensional function does not satisfy the terminology, because as said by the definition of extensional function the instances of intern_project *must be a subset of the instances of* project *but this is not verified atleast in the state* e_0 *(INT12 is not a* project*).*

With the definition of *satisfaction* we conclude this section, where starting with the notion of RWS of a set of values we have defined the concept and role support. Then we have defined a support function between a terminology and a relational schema and described the requirements that a support function must verify in order to be able to associate the state of a database with the extension of a terminology.

4 Enlargement of the mapping to deal with descriptions

In the previous section, the notion of *support function* has been presented and its semantics has been formally specified in terms of the relation established between a *state* of a set of relations and the *extension* of a terminology. In the following, we will show that for each description built over a terminology (mapped with a set of relations by a well formed support function) there is a corresponding query in terms of the relational schema(ta) support of the terminology. We will use the relational algebra extended with aggregate functions as the query language.

This section is organized in two parts, the first one is related to concept descriptions and the second one to role descriptions. In both of them, the different constructors that can appear in a description are analyzed, with the goal of associating the corresponding mapping information to each one of them.

4.1 Concept descriptions

In a terminological system, a concept description is formed by the conjunction of basic concept descriptions (see figure 6). Therefore, it seems reasonable to define first an operation that describes the *conjunction of concept supports* in order to define the support of a concept description. Then, we will see that the operation *conjunction of concept supports* is not enough and why another operation called *difference of concept supports* is needed. Finally, the support associated with basic concept descriptions will be presented.

CONJUNCTION OF CONCEPT SUPPORTS. Let S_1 y S_2 be two *concept supports* defined upon E, the *conjunction* of S_1 and S_2 is a concept support S such that verifies that for each state e of E, S(e) = S_1(e) $\cap$ S_2(e).

$$S=\{<\pi_{\overline{atr_1}}(R_1) \cap \pi_{\overline{atr_2}}(R_2)\ ,\overline{atr_1},T_1> : <R_1,\overline{atr_1},T_1> \text{ is a basic support of } S_1 \wedge <R_2,\overline{atr_2},T_2> \text{ is a basic support of } S_2 \wedge T_1=T_2\}$$

Example 9 *Let* S_e*={<STUDENT,(ID#),integer>} be the concept support of* student *and let* S_p*={<TEACHER,(ID#),integer>} be the concept support of* teacher. *The intersection of both supports will be:*

$$intersección(S_e,S_p)=\ \{<\pi_{ID\#}(STUDENT)\cap\pi_{ID\#}(TEACHER),(ID\#),integer>\}$$

Lemma 1 *Let C_1 and C_2 be two concepts belonging to a teminology T, associated with a database by a well formed support function F that satisfies T. If the terminology is augmented with a new defined concept C whose description is* C_1 and C_2 *and the support function F with the pair $<C$,intersection(S_{C_1},S_{C_2})$>$, the obtained support function, F', is well formed and satisfies the new terminology T'.*

Demonstration. First of all, we will prove that F' is well formed. As the only modification in the initial terminology has been to add a new concept, then it will have to be proved that the support of this new concept is well formed (see definition 4.7) and the first of the three points in definition 4.12:

- As for each state e of the database, $S_{C_1}(e)$ and $S_{C_2}(e)$ are well formed and F is also well formed, then it is not possible that there exist two different values of $S_C(e)$ that represent the same real world object. We will demonstrate it by *reductio ad absurdum*. Let us suppose that for the state e, there exist two different values, v_1 and v_2, that belong to $S_C(e)$ and that represent the same object. This means that v_1 and v_2 belong to $S_{C_1}(e)$ and to $S_{C_2}(e)$. As S_{C_1} and S_{C_2} are well formed, then the RWS(v_1) $\neq$ RWS(v_2), that is a contradiction with the previous affirmation.

  It must also be verified that for all state e and for all $v \in S_C(e)$, v is not ambiguous. We will use the same technique as in the previous case. Let us suppose that there exists a state e for which there is an ambiguous value $v \in S_C(e)$. As v also belongs to $S_{C_1}(e)$ and to $S_{C_2}(e)$, then or S_{C_1} or S_{C_2} are not well formed, or that F is not well formed. But this cannot happen because it contradicts the hypothesis.
- If $\exists e \exists v \in S_C(e) \exists v_i \in S_{C_i}(e)(v = v_i \wedge EMR(v) \neq EMR(v_i))$ (in other words, F' is not well formed) then this would imply that F is also not well formed, due to $\exists e \exists v \in S_{C_1}(e) \exists v_i \in S_{C_i}(e)(v = v_i \wedge RWS(v) \neq RWS(v_i))$.

Secondly, in order to prove that F' satisfies T', we will use the *reductio ad absurdum* again. F' does not satisfy T' if a) there exists an instance of C that is not an instance of C_1 or of C_2, what it is not possible due to the way of constructing S_C; or b) if there exists an instance of C_1 and of C_2 that is not an instance of C. This last situation would imply that there exists a state e such that there exists a value v that belongs to $S_{C_1}(e)$ and to $S_{C_2}(e)$ that does not belong to S_C, but this is not possible due to the fact that the support of S_C is based on the intersection of the relations that form the support of C_1 and C_2.

DIFFERENCE OF CONCEPT SUPPORTS. Unfortunately, the conjunction of concept supports is not enough to obtain the support of all the concept descriptions. When the constructors **all**, **atmost** and **not** are used, it is necessary to find another solution based on the *difference* of concept supports. We will introduce the problem with an example based on the concepts and roles of figure 3. In the description *teacher and atmost(3,Responsible_of)*, the second part of it, *atmost(3,Responsible_of)*, defines a concept that is a subconcept of the concept that is the domain of the role *Responsible_of* (in this case *teacher*). The instances of this subconcept cannot have more than three values for the role *Responsible_of*. Let us suppose that the role support of *Responsible_of* (defined over the schema of the figure 1) is:

<SUBJECT,Responsible,Sub#,integer,-,integer>

This support defines a set of pairs (teacher,subject), but among these, there is not a pair that corresponds to the teachers that are not responsible of any subject. Therefore, it is not possible to obtain the support for *atmost(3,Responsible_of)* only in base to the support of *Responsible_of*. However, it is possible to define the support for the concept whose instances are the teachers responsible of more than three subjects (the complementary of the concept defined by *atmost(3,Responsible_of)*).

If we work with *safe*[6] descriptions, then if the query contains the constructors **atmost**, **all** and **not**, then it will have to include another component based in other different constructors that do not have this problem.

In the previous example, the first component of the description is the concept *teacher*. With the mapping information associated with *teacher* is possible to obtain all the instances of this, that include all the instances of the concept *teacher and atmost(3,Responsible_of)*. In order to

[6]P.T. Devanbu defines the notion of *safe query* in the context of systems based on Description Logics [Dev93], and relates it with the standard notion [Ull88].

obtain these last ones, we can use the operation **difference** between the support of *teacher* and the support of the corresponding concept to teachers responsible of more than three subjects.

Definition 15 *Let S_1 and S_2 be two* concept supports *defined upon E, the* difference *of S_1 and S_2 is a concept support S such that for each state e of E it is verified that $S(e) = S_1(e) - S_2(e)$.*

$$S=\{< \pi_{\overline{atr_1}}(R_1) - \pi_{\overline{atr_2}}(R_2)\ ,\overline{atr_1},T_1> \,:\, <R_1,\overline{atr_1},T_1> \textit{ is a basic support of } S_1 \wedge$$
$$<R_2,\overline{atr_2},T_2> \textit{ is a basic support of } S_2 \wedge T_1{=}T_2\}$$

Example 10 *Let S_e=\{<STUDENT,(ID#),integer>\} be the concept support of* student *and let S_p=\{<TEACHER,(ID#),integer>\} be the concept support of* teacher, *used in a previous example. The* difference *of both supports will be:*

$$difference(S_{student},S_{teacher}){=}\ \{<\pi_{ID\#}(STUDENT){-}\pi_{ID\#}(TEACHER),(ID\#),integer>\}$$

Lemma 2 *Let C_1 and C_2 be two concepts that belong to a terminology T, associated with a database by a well formed support function F that satisfies T. If the terminology T is augmented with a new defined concept C whose description is* C$_1$ and not(C$_2$) *and the support function F with the pair $<C$,difference$(S_{C_1},S_{C_2})>$ then the obtained support function, F', is well formed and satisfies the new terminology T'.*

Demonstration. The demonstration is similar to that developed for the conjunction of concept supports. First of all, we will prove that F' is well formed. As the only modification in the initial terminology has been to add a new concept, it will have to be proved that the support of this is well formed (see definition 4.7) and that the first of the three points in definition 4.12:

- As for each state e of the database, $S_{C_1}(e)$ and $S_{C_2}(e)$ are well formed and F is also well formed, then it is not possible that there exist two different values of $S_C(e)$ that represent to the same real world object. We will demonstrate it by *reductio ad absurdum.* Let us suppose that for a state e, there exists a pair of different values, v_1 and v_2, that belong to $S_C(e)$ and that represents the same object. That means that v_1 and v_2 belong to $S_{C_1}(e)$ and as S_{C_1} is well formed, then the RWS(v_1) $\neq$ RWS(v_2), that is a contradiction with the previous affirmation.

  It must also be verified that for all state e and for all $v \in S_C(e)$ v is not ambiguous. We will use the same technique as in the previous case. Let us suppose that there exists an state e such that there exists an ambiguous value $v \in S_C(e)$. As v also belongs to $S_{C_1}(e)$, this would imply that S_{C_1} or F are not well formed, but this cannot happen because it contradicts the hypothesis.

- If $\exists e \exists v \in S_C(e) \exists v_i \in S_{C_i}(e)(v = v_i \wedge RWS(v) \neq RWS(v_i))$ (that means that F' is not well formed) then F would not be well formed, because $\exists e \exists v \in S_{C_1}(e) \exists v_i \in S_{C_i}(e)(v = v_i \wedge RWS(v) \neq RWS(v_i))$.

Secondly, in order to prove that F' satisfies T', we will use the *reductio ad absurdum* again. F' does not satisfy T' if a) there exists an instance of C that is not an instance of C_1 and it is an instance of C_2, what it is not possible due to the way of constructing S_C; or b) if there exists an instance of C_1 and that is not an instance of C_2 nor C. This last situation would imply that there exists a state e such that there exists a value v that belongs to $S_{C_1}(e)$ and that does not belong to $S_{C_2}(e)$ but that does belong to S_C, but this is not possible due to the fact that the support of S_C is based on the difference of the relations that form the support of C_1 and C_2.

In figure 7 the syntax-directed definition [ASU85] the support information for the concept descriptions in BACK is shown. As it can be proved, this definition is based on the use of the operations conjunction and difference, that combine the supports of the basic components of a description. By the moment, we have introduced the notion of concept and role support and defined the support of the conjunction and difference of concepts. We still have to define the support of the descriptions formed by only one constructor (**atleast, :, atmost and all**), in order to complete the specification of the mapping information associated with concept descriptions.

```
<concept> ::= <concept_name>
        {<concept>.sop := <concept_name>.sop)}
    |   atleast '(' number ',' <role> ')'
        {<concept>.sop := S_atleast(number.val,<role>.sop))}
    |   <role> ':' <value>.val
        {<concept>.sop := S_fills(<role>.sop,<value>.val)}
    |   <role> ':' close (' <value> )'
        {<concept>.sop := S_closefills(<role>.sop,<value>)}
    |   <concept> and <concept>
        {<concept>.sop := conjunction(<concept>(1).sop,<concept>(2).sop)}
    |   <concept> and atmost '(' number ',' <role> ')'
        {<concept>.sop := difference(<concept>(1).sop,S_atleast(number.val+1,<role>.sop))}}
    |   <concept> and not '(' <concept> ')'
        {<concept>.sop := difference(<concept>(1).sop,<concept>(2).sop)}
    |   <concept> and all '(' <role> ',' <concept> ')'
        {<concept>.sop := difference(<concept>(1).sop,S_notall(<role>.sop,<concept>(2).sop)}
    |   <concept> and all '(' <role> ',' <set> ')'
        {<concept>.sop :=
        difference(<concept>(1).sop,S_notallind(<role>.sop,<set>.set))}}
    |   <concept> and all '(' <role> ',' <number_rest> ')'
        {<concept>.sop :=
        difference(<concept>(1).sop,S_notallinrnum(<role>.sop,<number_rest>.rango))}
```

Figure 7: Support for the basic components of concept descriptions

A. atleast The constructor **atleast(n,role)** defines a concept, whose instances will be the instances of the concept domain of the *role* and that have at least n values for the *role*. Therefore, it is a subconcept of the concept formed by all the instances that have a value for *role*. Notice that the role support defines the pairs (instance,value) corresponding to a role extension.

Let r be a role, S_r the support of r and n an integer greater than 0, the defined concept for $atleast(n,r)$ will have $S_{atleast}$(n,r) as support, where

$$S_{atleast}(n,r) = \{<\sigma_{COUNT>n-1}({}_{\overline{atrd}}\mathcal{F}_{COUNT\ \overline{atrn}}(R_{rl})),\overline{atrd},T_C> : <R_{rl},\overline{atrd},\overline{atrn},T_C,f_{rl},T_r> \text{ is a basic role support of } S_r\}$$

Example 11 *Let it be the role support of* Responsible_of *that one that was introduced previously: <SUBJECT,Responsible,Sub#,integer,-,integer>.*

$$S_{atleast}(4,Responsible\_of) =$$
$$\{<\sigma_{COUNT>3}({}_{Responsible}\mathcal{F}_{COUNT\ Sub\#}(\text{SUBJECT})),\text{Responsible},\text{integer}>\}$$

As it can be seen, it has been necessary to use an aggregate function, [EN89], in order to group the role values that correspond to the same instance. This function, ${}_{\overline{grouping\_attr}}\mathcal{F}_{COUNT\overline{atr}}(R)$, defines a relation that will have as attributes $\overline{grouping\_attr}$ and $COUNT$. Its extension will include a tuple for each different value that $\overline{grouping\_attr}$ has for the tuples of R, the attribute $COUNT$ will have as value the number of different values taken by $\overline{atr}$ for the set of tuples with the same value for the attribute $\overline{grouping\_attr}$.

Lemma 3 *Let r be a role that belongs to a terminology T, associated with a database through a well formed support function F that satisfies T. If the terminology is augmented with a new defined concept C whose description is* atleast(n,r) *and the support function F with the pair $<C,S_{atleast}(n,r)>$ the obtained support function, F', is well formed and satisfies the new terminology T'.*

Demonstration. First of all, we will prove that F' is well formed. As the only modification in the initial terminology has been to add a new concept, then it will have to be proved that the support of this new concept is well formed:

- As F is well formed it is not possible that there exist two different values of $S_C(e)$ that represent the same real world object. We will demonstrate it by *reductio ad absurdum.* Let us suppose that for the state e, there exist two different values, v_1 and v_2 that represent the same real world object. This means that there exist at least two values v_1 and v_2 that belong to $S_r(e)$. Due to the second condition of a well formed function, v_1 y v_2 must belong to $S_{C_1}(e)$, where C_1 is the concept that has as domain r. At this point we get a a contradiction, because the first property of a well formed support function says that there cannot exist two different values belonging to the set of values defined for a state by a concept support and that represent the same real world object.

It must also be verified that for all state e and for all $v \in S_C(e)$, v is not ambiguous. Let us suppose that there exists a state e for which there is an ambiguous value $v \in S_C(e)$. There must exist a pair of the form (v, v_1) in $S_r(e)$. Due to the second condition of well formed function, v must belong to $S_{C_1}(e)$ where C_1 is the concept that has as domain r. This implies that S_{C_1} is not well formed, that is contradiction with the hypothesis.

Secondly, in order to prove that F' satisfies T', we will use the *reductio ad absurdum* again. F' does not satisfy T' if a) there exists an instance of C that does not take at least n values, what it is not possible due to the way of constructing S_C; or b) if there exists an instance that takes at least n values and that is not an instance of C. This last situation would imply that there exists a state e such that there exist at least n pairs of the form (v, v_i) belonging to $S_r(e)$ and v belongs to $S_{C_1}(e)$ and v does not belong to $S_C(e)$, what it is not possible due to the semantics of the aggregate function in which the support of S_C is based.

The lemmas and demonstrations for the rest of the constructors are similar, and we will not introduce them, with the goal of not enlarging this presentation unnecessarily.

B. role : expression The constructor **role : expression** defines a subconcept of the concept that has *role* as domain. The instances of this subconcept will verify the restrictions imposed to the *role* values by *expression*. The expression defines a set of values that the role must take; this definition can be *close* or not. If the expression is closed (when the keyword **close** is used) then it indicates that the role must take the values that appear in the expression and not other values. In other case, the role must take at least these values but it may take others too.

B.1. First of all, we introduce the mapping information for the description **role: value**. The generalization for a list of values is trivial, but however we do not introduce it to make this presentation easier.

$$\mathrm{S}_{fills}(\mathrm{S}_r,\mathrm{v}) = \{<\sigma_{\overline{atrn}=f_{rl}^{-1}(v)}(\mathrm{R}_{rl}),\overline{atrd},\mathrm{T}_C> \; : \; <\mathrm{R}_{rl},\overline{atrd},\overline{atrn},\mathrm{T}_C,\mathrm{f}_{rl},\mathrm{T}_r> \text{ is a basic role support of } \mathrm{S}_r \}$$

In this case, it is necessary to use the inverse function of the function associated with a role support. This happens because the description is expressed over the range of values taken by the role and not over the values stored in the database.

Example 12 *The support corresponding to the concept whose instances take the value* 880 *for the role Responsible_of, whose mapping information was previously introduced and that is not associated with a transforming function is the following:*

$$S_{fills}(S_{Responsible\_of},880) = \{<\sigma_{\overline{Sub\#}=880}(SUBJECT),Responsible,integer> \}$$

B.2. As in the previous case, we introduce the mapping information for the case **role:close(value)**:

$$\mathrm{S}_{closefills}(\mathrm{S}_r,\mathrm{v}) = \{<\pi_{\overline{atrd}}(\sigma_{\overline{atrn}=f_{rl}^{-1}(v)}(\mathrm{R}_{rl})) - \pi_{\overline{atrd}}(\sigma_{\overline{atrn} \neq f_{rl}^{-1}(v)}(\mathrm{R}_{rl})) \;,\overline{atrd},\mathrm{T}_C> : \\ <\mathrm{R}_{rl},\overline{atrd},\overline{atrn},\mathrm{T}_C,\mathrm{f}_{rl},\mathrm{T}_r> \text{ is a basic role support of } \mathrm{S}_r \}$$

Example 13 *Working with the same role Responsible_of, the support for the description of the concept Responsible_of* : *close*(880) *is the following:*

$$\mathrm{S}_{closefills}(\mathrm{S}_{Responsible\_of},880) = \{<\pi_{Responsible}(\sigma_{Sub\#=880}(\mathrm{SUBJECT})) - \\ \pi_{Responsible}(\sigma_{Sub\# \neq 880}(\mathrm{SUBJECT})) \;,\mathrm{Responsible},\mathrm{integer}> \}$$

C. atmost As we have explained when introducing the operator *difference between concept supports*, to obtain the support of a description that contains the constructor **atmost** it is necessary to obtain the support of the complementary concept of **atmost(n,role)**. This complementary concept is **atleast(n+1,role)**, and therefore it is not needed a specific description for it.

Example 14 *For the previously used description* teacher and atmost(3,Responsible_of)*, that corresponds to the concept whose instances are the instances of* teacher *that verify that at most are responsible of three subjects, the mapping information would be the following:*

$$difference(S_p, S_{atleast}(4, Responsible\_of))$$

That is, the difference between the mapping information of teacher *and the concept whose instances take at least four values for the role* Responsible_of.

D. all The constructor **all(role,concept_type)** defines a subconcept of the concept domain of the *role*. The instances of this subconcept will verify that the values that *role* takes will belong to the *concept_type*. Using the same reasoning as for the constructor **atmost** we find that it is not possible to obtain the concept support defined for **all(role,concept_type)**. The reason is that those instances of the concept that have *role* as domain but that do not have any value for this role, cannot be found among those defined by the role support. However, they verify the description because all the values that they take (no value in this case) verify the restriction imposed by the constructor **all**. On the contrary, it is possible to obtain from the role support, the support for the concept that takes at least one value for the role that does not verify the restriction. This concept is a subconcept of the concept formed by all the instances that take at least one value for the role.

The constructor **all** can restrict the values to take for a role to be instances of a concept or to belong to a number range or to belong to a set of non-number values. Depending on each one of the cases the way of obtaining the mapping information will be different:

D.1. all(role,concept) Let C be a concept and let S_C be the support of C; and r a role and S_r the support of r.

$$\begin{gathered}\mathrm{S}_{notall}(\mathrm{S}_r,\mathrm{S}_C) = \\ \{<\pi_{\overline{atrd}}(\mathrm{R}_{rl}) - \pi_{\overline{atrd}}(\pi_{\overline{atrd}\ \overline{atrn}}(R_{rl}) - \pi_{\overline{atrd}\ \overline{atrn}}(\mathrm{R}_{rl}\bowtie_{\overline{atrn}=\overline{atrc}}\mathrm{R}_C)\ )\ ,\overline{atrd},\mathrm{T}_C> \\ :\ <\mathrm{R}_{rl},\overline{atrd},\overline{atrn},\mathrm{T}_C,\mathrm{f}_{rl},\mathrm{T}_r> \text{ is a basic role support of } \mathrm{S}_r \\ \wedge <\mathrm{R}_C,\overline{atrc},\mathrm{T}_C> \text{ is a basic support of } \mathrm{S}_C \wedge \mathrm{T}_r=\mathrm{T}_C\}\end{gathered}$$

Example 15 *If the support of the concept* assistant_professor *in example 4.7 is* $<\sigma_{Title='Assistant'}(TEACHER),(ID\#),integer>$ *and the role support for Responsible_of of the concept* subject *is <SUBJECT,Sub#,Responsible,integer,-,integer>, then the complementary concept support equivalent to* all(Responsible_of,assistant_professor) *is:*

$$\begin{gathered}S_{notall}(S_{Responsible}, S_{assistant\_professor}) = \\ \{<\pi_{Sub\#}(\mathrm{SUBJECT}) - \pi_{Sub\#}(\pi_{Sub\#\ Responsible}(\mathrm{SUBJECT}) - \\ \pi_{Sub\#\ Responsible}(\mathrm{SUBJECT}\bowtie_{Responsible=ID\#}(\sigma_{Title='Asoc'}(\mathrm{TEACHER}))\ )\ )\ \ ,Sub\#,integer>\ \}\end{gathered}$$

D.2. all(role,value_set) Let D be a domain, defined over a set of values d; let r be a role and S_r the support of r.

$$\begin{gathered}\mathrm{S}_{notallind}(\mathrm{S}_r,\mathrm{d}) = \{<\overline{atrd},\sigma_{\overline{atrd}\neq f_{rl}^{-1}(v_1)\ and\ \ldots\ and\ \overline{atrd}\neq f_{rl}^{-1}(v_n)}(\mathrm{R}_{rl}),\mathrm{T}_C> : \\ <\mathrm{R}_{rl},\overline{atrd},\overline{atrn},\mathrm{T}_C,\mathrm{f}_{rl},\mathrm{T}_r> \text{ is a basic role support of } \mathrm{S}_r \wedge \mathrm{d}=(\mathrm{v}_1,\ldots,\mathrm{v}_n)\ \}\end{gathered}$$

D.3. all(role,number_range) Let $rnum$ be a number range, defined by a pair (inf, sup); let r be a role and S_r the support of r.

$$\begin{gathered}\mathrm{S}_{notallinrnum}(\mathrm{S}_r,(\mathrm{inf},\mathrm{sup})) = \{<\sigma_{\overline{atrd}<f_{rl}^{-1}(inf)\ or\ \overline{atrd}>f_{rl}^{-1}(sup)}(\mathrm{R}_{rl}),\overline{atrd},\mathrm{T}_C> : \\ <\mathrm{R}_{rl},\overline{atrd},\overline{atrn},\mathrm{T}_C,\mathrm{f}_{rl},\mathrm{T}_r> \text{ is a basic role support of } \mathrm{S}_r\ \}\end{gathered}$$

This basic support has been able to be defined due to the restriction that has been added to the transforming function when the notion of role support was introduced. The restriction was that only are acceptable functions those that have inverse and that are strictly crescent.

4.2 Role descriptions

Role descriptions are also formed by conjunction of basic role descriptions (see figure 8), although it is intuitively less clear that the conjunction of roles defines a new role.

We will describe first the support that corresponds to a role whose definition is formed by the conjunction of two given roles. Then we will introduce the mapping information for the roles defined as inverse of other roles. And finally, the support for the roles defined by restricting the domain or the range will be shown.

```
<role> ::= <role_name>
        |  inv ( <role> )
        |  <role> and <role>
        |  <role> and domain '(' <concept> ')'
        |  <role> and range '(' concept ')'
        |  <role> and range '(' <set> ')'
        |  <role> and range '(' <number_rest> ')'
```

Figure 8: Brief syntax of role descriptions

```
<role> ::= <role_name>
           {<role>.sop := <role_name>.sop}
         | inv ( <role> )
           {<role>.sop := S_inv(<role>(1).sop)}
         | <role> and <role>
           {<role>.sop := conjunction(<role>(1).sop,<role>(2).sop}
         | <role> and domain '(' <concept> ')'
           {<role>.sop := S_dom(<role>(1).sop,<concept>.sop)}
         | <role> and range '(' <concept> ')'
           {<role>.sop := S_rg(<role>(1).sop,<concept>.sop)}
         | <role> and range '(' <set> ')'
           {<role>.sop := S_set(<role>.sop,<set>.set)}
         | <role> and range '(' <number_rest> ')'
           {<role>.sop := S_rangen(<concept>.sop,<number_rest>.range)}
```

Figure 9: Support for the role descriptions

CONJUNCTION OF ROLE SUPPORTS Let S_{r_1} and S_{r_2} be two *role supports* defined upon E, the *conjunction* of S_{r_1} and S_{r_2} is a role support S_r such that it verifies $S_r(\mathrm{e}) = S_{r_1}(\mathrm{e}) \cap S_{r_2}(\mathrm{e})$, for each state e of E.

$$\mathrm{S}_r=\{< \pi_{\overline{atrd_1}\ \overline{atrn_1}}(\mathrm{R}_1) \cap \pi_{\overline{atrd_2}\ \overline{atrn_2}}(\mathrm{R}_2)\ ,\overline{atrd_1},\overline{atrn_1},\mathrm{T}_{C_1},\mathrm{f}_{rl_1},\mathrm{T}_{r_1}> :$$
$$<\mathrm{R}_1,\overline{atrd_1},\overline{atrn_1},\mathrm{T}_{C_1},\mathrm{f}_{rl_1},\mathrm{T}_{r_1}> \text{ is a basic role support of } S_{r_1} \wedge <\mathrm{R}_2,\overline{atrd_2},\overline{atrn_2},\mathrm{T}_{C_2},\mathrm{f}_{rl_2},\mathrm{T}_{r_2}> \text{ is}$$
$$\text{a basic role support of } S_{r_2} \wedge \mathrm{T}_{r_1}=\mathrm{T}_{r_2} \wedge \mathrm{T}_{C_1}=\mathrm{T}_{C_2} \wedge \mathrm{f}_{rl_1}=\mathrm{f}_{rl_2}\}$$

INVERSE ROLE. Let r be a role that has as domain the concept C and as range the concept C_1, the role $inv(r)$ represents the inverse of the relation represented by r, $inv(r)$ takes as domain the concept C_1 and as range the concept C.

Let r be a role and let S_r be the support of r. The role defined by $inv(r)$ will have as support $\mathrm{S}_{inv}(\mathrm{S}_r)$.

$$\mathrm{S}_{inv}(\mathrm{S}_r) = \{<\mathrm{R}_{rl},\overline{atrn},\overline{atrd},\mathrm{T}_r,\text{-},\mathrm{T}_C> :$$
$$<\mathrm{R}_{rl},\overline{atrd},\overline{atrn},\mathrm{T}_C,\text{-},\mathrm{T}_r> \text{ is a basic role support of } \mathrm{S}_r\}$$

DOMAIN RESTRICTION. Let r be a role that has as domain the concept C and let C_1 be a concept that is a subconcept of C, the role *r and domain*(C_1) defines a subrelation of the relation represented by r. This subrelation defines for the instances of C_1 (and only for them) the same relation as r.

Let C be a concept and let S_C be the support for C; let r be a role and let S_r be the support for r. The role defined by *r and domain*(C) will have $\mathrm{S}_{dom}(\mathrm{S}_r,\mathrm{S}_C)$ as support.

$$\mathrm{S}_{dom}(\mathrm{S}_r,\mathrm{S}_C) = \{<\mathrm{R}_{rl}\bowtie_{\overline{atrd}=\overline{atrc}}\mathrm{R}_C,\overline{atrd},\overline{atrn},\mathrm{T}_C,\mathrm{f}_{rl},\mathrm{T}_r> :$$
$$<\mathrm{R}_{rl},\overline{atrd},\overline{atrn},\mathrm{T}_{C_1},\mathrm{f}_{rl},\mathrm{T}_r> \text{ is a basic role support of } \mathrm{S}_r \wedge <\mathrm{R}_C,\overline{atrc},\mathrm{T}_C> \text{ is a basic support of } \mathrm{S}_C \wedge$$
$$\mathrm{T}_{C_1}=\mathrm{T}_C\}$$

RANGE RESTRICTION. Let r be a role that has as domain the concept C, the role *r and range(concept_type)* defines a subrelation of the relation represented by r. This subrelation is only established among the instances of C and those objects that belong to the *concept_type*. The mapping information associated with role depends on if the *concept_type* is a concept, a set of values or a number range. We will describe each one of the possibilities in the following:

range(concept) Let C be a concept and let S_C be the support of C; let r be a role and let S_r be the support of r. The role defined by *r and range*(C) will have $\mathrm{S}_{rg}(\mathrm{S}_r,\mathrm{S}_C)$ as support, where

$$\mathrm{S}_{rg}(\mathrm{S}_r,\mathrm{S}_C) = \{<\mathrm{R}_{rl} \bowtie_{\overline{atrn}=\overline{atrc}} \mathrm{R}_C,\overline{atrd},\overline{atrn},\mathrm{T}_d,\mathrm{f}_{rl},\mathrm{T}_r> : \\ <\mathrm{R}_{rl},\overline{atrd},\overline{atrn},\mathrm{T}_d,\mathrm{f}_{rl},\mathrm{T}_r> \text{ is a basic role support of } \mathrm{S}_r \wedge <\mathrm{R}_C,\overline{atrc},\mathrm{T}_C> \text{ is a basic support of } \mathrm{S}_C \wedge \\ \mathrm{T}_r=\mathrm{T}_C\}$$

range(value_set) Let r be a role and let S_r be the support of r, let D be a domain that defines a set of values d. The role defined by *r and range*(D) will have $\mathrm{S}_{ranged}(\mathrm{S}_r,d)$ as support, where

$$\mathrm{S}_{ranged}(\mathrm{S}_r,d) = \{<\sigma_{\overline{atrn}=f_{rl}^{-1}(v_1) \text{ or } \ldots \text{ or } \overline{atrn}=f_{rl}^{-1}(v_n)}(\mathrm{R}_{rl}),\overline{atrd},\overline{atrn},\mathrm{T}_C,\mathrm{f}_{rl},\mathrm{T}_r> : \\ <\mathrm{R}_{rl},\overline{atrd},\overline{atrn},\mathrm{T}_C,\mathrm{f}_{rl},\mathrm{T}_r> \text{ is a basic role support of } \mathrm{S}_r \wedge \mathrm{d}=(\mathrm{v}_1,\ldots,\mathrm{v}_n\}$$

range(number_range) Let r be a role and let S_r be the support of r. The role defined by *r and range*$(number\_range)$ will have as support $\mathrm{S}_{rangen}(\mathrm{S}_r,(inf, sup))$, where

$$\mathrm{S}_{rangen}(\mathrm{S}_r,(inf, sup)) = \\ \{<\sigma_{\overline{atrn} \geq f_{rl}^{-1}(inf) \text{ and } \overline{atrn} \leq f_{rl}^{-1}(sup)}(\mathrm{R}_{rl}),\overline{atrd},\overline{atrn},\mathrm{T}_C,\mathrm{f}_{rl},\mathrm{T}_r> : <\mathrm{R}_{rl},\overline{atrd},\overline{atrn},\mathrm{T}_C,\mathrm{f}_{rl},\mathrm{T}_r> \text{ is a basic} \\ \text{role support of } \mathrm{S}_r\}$$

Once the mapping information has been defined for each one of the basic components of the role descriptions we can use a syntax-directed definition (see figure 9) to specify the mapping information for any role.

4.3 Summary

Throught this section it has been specified the mapping information that corresponds to a BACK description built over a terminology mapped with a relational database. The same procedure has been followed for the concept descriptions and for the role descriptions. First, we have defined the mapping information for each constructor that can appear in a description. Next, we have associated the grammar that describes the BACK descriptions with the actions that specify the mapping information by using a syntax-directed definition.

From a practical point of view, the specification that has been developed will be used during the translation and integration processes in order to obtain, in an automatic way, the mapping information associated with new concept and roles of the terminology introduced as specializations of other concepts and roles that are already in the terminology and mapped with the databases. This will permit the Person Responsible for the Integration to focus in conceptual design aspects leaving the responsibility of managing the mapping information to the system.

5 Conclusions

Although database systems and knowledge based systems have different objectives it is widely recognized the practical interest of connecting them. In this paper we have centered in the problem of connecting relational databases with a particular type of knowledge based system, a terminological system. An important aspect of the conexion is the definition of the mapping relation that permits us to relate data elements of the relational schemata and of the terminologies. We have defined the syntax and the semantic of the mapping. Moreover, we have demonstrated its completeness property in the sense that it allows for obtaining answers to all queries formulated over the terminology using the data stored in the relational database. Last, we want to mention the wide spectrum of possible transformations that it covers, allowing for exploiting all the features provided by the terminological system when creating a knowledge representation schema from a relational one.

References

[ASU85] A. Aho, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques and Tools.* Addison Wesley, 1985.

[BB93] A. Borgida and R. J. Brachman. Loading data into description reasoners. In *Proceedings of the ACM SIGMOD Conference*, 1993.

[BIG94] J.M. Blanco, A. Illarramendi, and A. Goñi. Building a federated database system: an approach using a knowledge based system. *International Journal on Intelligent and Cooperative Information Systems*, 3(4):415–455, December 1994.

[Bor] A. Borgida. Description logics in data management. *IEEE Transactions on Knowledge and Data Engineering.* To appear.

[BS85] R. Brachman and J. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9:171–216, 1985.

[Dev93] P.T. Devanbu. Translating description logics to information server queries. In *Proceedings of the ISMM International Conference on Information and Knowledge Management CIKM*, 1993.

[EN89] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems.* Addison-Wesley, 1989.

[Int87] Intellicorp. Keeconnection^{TM}: A bridge between databases and knowledge bases. Technical report, IntelliCorp Inc., 1987.

[LNE89] J. A. Larson, S. B. Navathe, and R. Elmasri. A theory of attribute equivalence in databases with application to schema integration. *IEEE TOSE*, SE-15(4), April 1989.

[ME92] P. Mishra and M. Eich. Functional completeness in object-oriented databases. *SIGMOD RECORD*, 21(1), March 1992.

[NGG89] S. Navathe, S. K. Gala, and S. Geum. Federated information bases: A loose-coupled integration of databases systems and application subsystems. In *Proc. of the 4th. Database Symposium*, 1989.

[PSKQ89] C. Peltason, A. Schmiedel, C. Kindermann, and J. Quantz. The BACK system revisited. Technical University Berlin, September 1989.

[SPD91] S. Spaccapietra, C. Parent, and Y. Dupont. Automating heterogeneous schema integration. Technical report, Departement D'Informatique. Ecole Polytechnique Federale de Lausanne, February 1991.

[Ull88] J.D. Ullman. *Principles of Database and Knowledge-Base systems.* Computer Science Press, Inc, 1988.

[WS92] W.A. Woods and J.G. Schmolze. The KL-ONE family. *Computers Math. Applic.*, 23(2):133–177, 1992.