MAGIC: An Interface for Generating Mapping Information between Object-based and Relational Systems

Mena E., Illarramendi A. and Blanco J.M.

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

Abstract

Nowadays, it is increasing the interest of working with systems designed under the object-based paradigm. However, most organizations possess their data stored in relational or pre-relational databases, data that they need for the every day work. In order to permit them to continue dealing with the existing databases while taking advantage of the features provided by the new systems, different tools are being designed. In this paper we present an interface that allows generating mapping information between relational databases and schemata defined using systems designed under the object-based paradigm. This interface provides a user friendly environment and syntactic and semantic checking to help the user in the connexion process.

Keywords: Relational Database Systems, Object-oriented Systems, Object-based Knowledge Representation Systems

1 Introduction

It is widely recognized that object-based technology plays and is going to play a very important role in the near future. However, most organizations possess their data stored in relational or pre-relational databases, data that they need for their every day work. In order to allow a smooth migration from the old database technology to the object-based one, different tools are being designed, tools that do not alter the daily work but nevertheless permit organizations to incorporate the new technology.

Among systems designed, under the object-based technology we consider two types. In the first type, we include the object-oriented database systems such as O2 [CLV88] or GemStone [BMO+89]. In the second type, we include object-based knowledge representation systems, such as those designed under the paradigm of KL-ONE [BS85], e.g. Back [PSKQ89]. Although the functionalities provided by them are different, like the possibility of defining behaviour by the systems that belong to the first group, in contrast with the classification mechanism[1] incorporated in the systems that belong to the second type, they mainly agree in the kind of basic elements they offer to define the structural part of a data or knowledge base.

Moreover, it is clear that the classical database technology and the object-based technology have different objectives, but their coalition provides many practical benefits, which includes the possibility of dealing efficiently with persistent data stored in relational and pre-relational databases under the representational power of the object-oriented database systems or object-based knowledge representation systems and the coexistence of different autonomous databases under a global integrated view (Federated Systems) defined using an object-oriented database system or an object-based knowledge representation system. For both situations, a mapping information must be generated that relates basic elements provided by the different systems.

In the literature the problem of connecting different types of systems has been studied from different points of view. The goal of some of them consists of defining an interface on top of a particular system that provides some functionalities inherent to the other system. In this group we can include systems that provide a relational interface for hierarchical [Lie81] or CODASYL [Ill87] database systems. However, a different approach has been followed for those that wish to provide an interoperable capability between both systems. In this last case two different alternatives are distinguished aiming at different levels of integration. One type is the multidatabase approach [LMR90], in which users can query different databases with a single request, but have to specify where the data is located. A different type is federation, where a federated schema is provided to the users to formulate queries over. In the last case, location transparency and site autonomy are supported by the system. Nevertheless, not all federated approaches use the same methodology; some do Structural Mapping –defining correspondences between the data elements of the systems which have to be integrated, others do Operational Mapping, where correspondences are made between operations at different levels [BNPS89]. Even within the Structural Mapping approach, a variety of solutions have been presented. On the one side, we find those that propose a manual integration [Mot87], where the user is responsible to build the federated schema; on the other side, those that claim automatic integration (i.e.[SK92],

[1]Classification understood as a mechanism that discovers the subsumption relationships among classes when a new class is defined.

[SPD92],[LNE89], [CHS91], [NGG89]), where the responsibility for generating the federated schema lies with the system.

In general, system integration involves many different problems such as integrating schemata of local systems into a federated schema, mapping actual data from the local schemata to the federated one and mapping queries formulated over the federated schema into queries against local schemata. Although our general goal is to build a federated database system that permits one to integrate heterogeneous relational databases by using a system designed under the object-based paradigm [BIG95] , and therefore we have studied all the distinct problems , in this paper we concentrate in the second problem, that is, mapping actual data from the local schemata to the federated one. Our search of the literature shows that in general, a precise definition of the generated mapping information, when connecting different types of systems, is not introduced and so systems that facilitate that work are missing. Nevertheless, a clear semantic of a mapping information is absolutely necessary in any connexion context. The goal of the paper is twofold, on the one hand, to explain the features of the mapping information needed for connecting object-based systems and relational databases. On the other hand, to show an interface, MAGIC, that facilitates the task of generating that mapping information.

In the remainder of this paper we present first the definition of the mapping information. Next we explain in detail the features of the provided interface.

2 Definition of the mapping relation

As we have mentioned before, the goal of the mapping information is to establish the relationship between basic elements of two systems. In object-oriented database systems and object-based knowledge representation systems the two main categories of basic elements are classes[2] and class attributes[3] while in relational databases are relations and attributes. Classes group individual objects that shares common structure and behavior, and the class attribute values represent the object's status. The set of attributes of an object represent the object structure.

A very simple mapping process between relational databases and object-oriented database or object-based knowledge representation systems would consist of translating each relation of a relational schema into a class and each attribute into a class attribute. However, in such a case, one would not take profit of the richer representational power offered by the object-based system. For example, from the relational schema in figure 1.a which represents information about teachers and students in one faculty, we could be interested in obtaining a semantically richer schema, represented in figure 1.b.

In figure 1.b we observe that *TEACHER* and *STUDENT* have been generalized into a class *PERSON* and therefore we can have an abstract and uniform view of them. On the other side, *STUDENT* has been specialized in two classes: *UNDERGRADUATE* and *POSTGRADUATE*. This permits to classify the students into two categories and

[2]There not exist an standardized notation. For example, in BACK, systems classes are called concepts and attributes are called roles.

[3]In order to avoid the confusion between attributes of classes and attributes of relations, the former ones will be called class attributes.

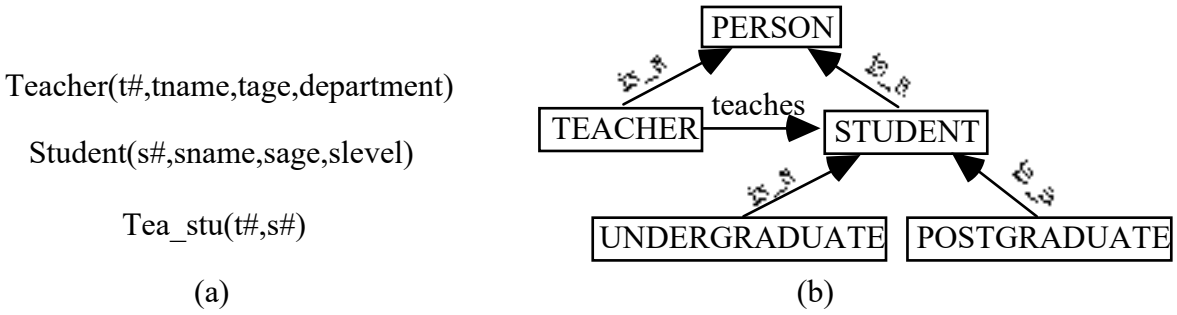

Figure 1: Schemata that should be related by a mapping information

so facilitates formulation of queries related with each category. Last, the *tea_stu* relation has been transformed into a class attribute *teaches* of *TEACHER*.

Nevertheless, we cannot forget that the resultant schema is just a virtual image of the underlying database, so, a mapping information is needed to correlate objects of both schemata. The elements of the resultant schema will be classes and class attributes, and therefore a mapping will be needed that relates these elements with the elements of the relational schemata, namely, relations and attributes. The mapping information defines links, in such a way, that given a link definition and one state of a relational database, the objects of the corresponding classes as well as the class attribute values for those objects can be determined.

2.1 Mapping for classes

The simplest type of link that can be defined is one which relates a class with a basic relation. The semantic of this link will be that there exists a class object for each tuple of the basic relation. For example, consider the relation *teacher(t#,tname,tage,department)*. It is possible to link this relation with the class *TEACHER*, which means that for each tuple in the relation *teacher* there is one object of the class *TEACHER*.

However, in many situations, it can happen that only a subset of tuples of a basic relation, which satisfy some conditions, determine class objects. For example, let suppose that we are interested in defining the class *POSTGRADUATE* in such a way that there is an object of this class for each tuple of the relation *student* that satisfies the condition $slevel = 3$. Therefore, we generalize the previous link definition by allowing a link to be established between a class and a relation[4]. The semantic of the link is redefined to mean that there exists a class object for each tuple of the relation.

In other cases, the need arises for one class object to represent several tuples of a relation. This means that there exists a class object for each different value that takes one (or more) attributes of a relation. For example, it is possible to link the relation *teacher* and its attribute *dept* with the class *DEPARTMENT*, as a consequence there will be an object of *DEPARTMENT* for each different value that takes the attribute *dept* in the tuples of the relation *teacher*.

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 object. For example, a tuple of

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

the relation *teacher* implies the existence of an object of the class *TEACHER* and, sometimes, another of the class *DEPARTMENT*.

Finally, several relations may determine the objects of a class. For example, the class *PERSON* is linked with two relations, *teacher* and *student*, in this way, there is an object of *PERSON* for each tuple of the relations *teacher* and *student*. In this case a link will be established between a class and a set of relations.

The mapping information associated with a class, also called *class support*, can be represented as a list of tuples of the form:

$$< R, (atr_1, \ldots, atr_n), T >$$

where R is either a basic or a derived relation; atr_1,...,atr_n are attributes of R that permit the identification of objects supported by the relation; and T is the type of the attributes. Each tuple will be called *basic class support*. In the appendix is presented a complete syntactic definition of the mapping information.

Moreover, in object-based technology objects of classes are univocally identified by the object identifier (*oi*), that makes an object distinguishable from the others. In our context it is possible to obtain a unique identifier *oi* for each object by using the values of the attributes atr_1,...,atr_n of the tuple(s) in which the object is based. Let i be an object of a class C, and let (t_1,...,t_m) the set of tuples of R in which i is based, the object identifier of i, oi_i, is the result of applying the function *get_oi* to the list of values that the attributes *atr1,...,atrn* have for the tuples $t_1 \ldots t_m$.

For example, *TEACHER* → <*teacher,t#,integer*>, means that for each tuple of *teacher* there will be an object of *TEACHER* (because *t#* identifies univocally the tuples of *teacher*). Let be $< 11769, Smith, 40, CS >$ a tuple of *teacher*, the *oi* associated with the object supported by this tuple is the result of $get\_oi(TEACHER, 11769)$.

In this way, it is possible to identify the set of objects of the class C by applying the function *get_oi* to the tuples obtained as result of projecting the attributes *atr1,...,atrn* over the relation R.

2.2 Mapping for class attributes

So far we have presented the link types for a class; we now concentrate on the class attribute values. In general, the mapping information of a class attribute is based on the mapping information of the class to which it belongs to. Therefore, the simplest type of link that can be defined for a class attribute is one which relates the class attribute with an attribute of a relation that supports[5] the class to which it belongs to. Two different situations can arise:

- There exists an object for each relation tuple. In this case, each class attribute value will be the corresponding attribute value, excluding the null values for which there is no image. For example, the class attribute *tage* of *TEACHER* can be linked to the attribute *tage* of *teacher*.

- There exists an object for a set of tuples. In this case, the class attribute 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 class attribute *d_name* associated with *DEPARTMENT* can be linked to the attribute *dept* of *teacher*.

[5]A relation supports a class when this relation appears in the mapping information of that class.

However, a class attribute may also be based on an attribute that does not appear in the relation that supports the class to which the class attribute belongs to. For this case, it is necessary to establish a link between the class attribute values obtained from the attributes and the objects in which the class attribute takes part. For example, the class attribute *teaches* defined for the class *TEACHER* will be linked with the attribute *sname* of the relation *student*. In this way the link between objects of *TEACHER* and the value of the class attribute *teaches* is established through the join of three relations, *teacher*, *tea-stu* and *student*.

As in the case of classes, the mapping information of a class attribute will be a list; tuples in this list will have the form:

$$< R, (atrd_1, \ldots, atrd_n), T, (atrn_1, \ldots, atrn_m), T_{rl}, f_{rl} >$$

where R is either a basic or a derived relation; $atrd_1$,...,$atrd_n$ are attributes of R that permit the identification of objects supported by the relation; T is the type of those attributes, $atrn_1$,...,$atrn_n$ are attributes of R that define the class attribute values (or contain the class attribute values) corresponding to the class objects; f_{rl} is a function defined as $f_{rl} : D_1 \times \ldots \times D_m \rightarrow T_{rl}$, where D_i is the domain of attribute $atrn_i$ for all i between 1 and n and finally, T_{rl}, is the range of the role (f_{rl} allows the transformation of attribute values, e.g. from $ to ECUs). Each tuple will be called *basic class attribute support*.

Figure 2 shows the mapping information, by using the representation described before, for the example presented in figure 1.

TEACHER	(<teacher,t#,integer>)
POSTGRADUATE	(<$\sigma_{slevel=3}$(student),s#,integer>)
DEPARTMENT	(<teacher,dept,string>)
PERSON	(<teacher,t#,integer>,<student,s#,integer>)
TEACHES	(<teacher$\bowtie$tea-stu$\bowtie$student,t#,integer, sname,string, —>)

Figure 2: Example of Mapping Information

3 MAGIC: A Mapping Generator Interface

As we can deduce from the previous section, the process of generating the mapping information it is quite complicate and requires to manage, at the same time, many different kinds of informations. For this reason we have designed and implemented MAGIC, a MApping Generator InterfaCe that facilitates the generation process. The main features that MAGIC provides are:

- A syntactic and semantic checking during the generation process. So, it is guaranteed that the mapping information generated is correct.
- A guided mapping generation process, providing at each step only the choices that are permitted.

- A user friendly interface based on a window framework.

Basically, the process of generating a mapping information consists of defining the list of basic class supports, or basic class attribute supports, one by one.

Two main windows will appear (see figure 5 in section 3.1.1) during the process of creating the mapping information of a class or a class attribute, namely:

- *Graphical Representation of the Relation.* This window will show a tree representation of the relation (basic or derived) being in the generation process. The nodes labeled with the symbol *?* will represent the operands that have not been yet introduced by the user. The next operand that must be introduced is distinguished by a double box. If the representation is bigger than the window space, then the scroll operation is allowed.

- *Description of Class Support.* This window will contain the history of the basic supports that have already been defined for the class within the new one that is in the generation process.

Above the previous two windows, other different windows will appear depending on the operation active at each moment.

In the following, we show the details of the interface for classes and class attributes.

3.1 Interface for a class

The two main steps for generating the mapping information associated to a class are:

1. To obtain the relation (basic or derived) that permits supporting the class objects.

2. To select the attribute(s) that allows identifying class objects.

These steps will be repeated for each basic support, as we can see in the figure 3.

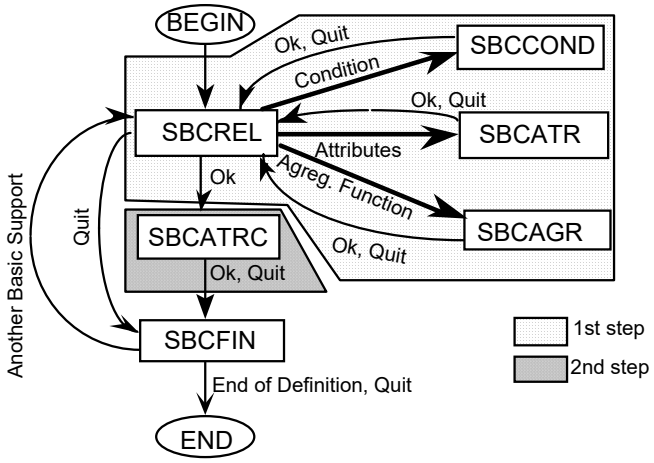

Figure 3: Definition of a class support

3.1.1 Obtaining a Relation

The goal of this step is to obtain the relational expression that allows selecting the tuples that will support objects of a class. In this case, above the two permanent windows, the following ones will appear:

- *Making the Relation.* It is a menu with the options showed in figure 4.
- *Tables.* List of basic tables that can be used to define the relational expression.
- *Operations.* Set of relational algebra operators allowed in any relational expression.

Undo	To undo the last operation
OK	To finish the step
Quit	To eliminate the current basic support
Attributes	To select a set of attributes
Condition	To define a condition
Aggr. Function	To define an aggregate function

Figure 4: Options of the *Making the Relation* window

For example, in the left side of figure 5 we show a graphical framework when the selection operation has been chosen to define a relational expression for the class *POSTGRADUATE*. Notice that in the *Tables* window three basic relations, *teacher*, *student* and *tea-stu* appear.

However, when the specific goal is to express a condition, the windows that will appear are:

- *Graphical representation of a condition.* It will contain a tree representation of a condition.
- *Attributes.* Set of attributes that can participate in the elaboration of a condition.
- *Making the Condition.* It is a menu with the options *Undo*, to undo the last operation; *OK*, to indicate the end of the task; *Quit*, to quit the definition of a condition; and *Const. Value*, to introduce constants that will take part of a condition.

In the right side of figure 5 we show the situation when it is expressed the condition *slevel = 3*.

Finally, when the goal is to define an aggregate function, the windows that will appear are:

- *Aggreg. Functions.* It will contain a set of predefined aggregate functions.
- *Selected Aggreg. Function.* It will show the selected aggregate function.
- *Choose Attributes.* It will present a set of attributes for which the aggregate function will be applied.

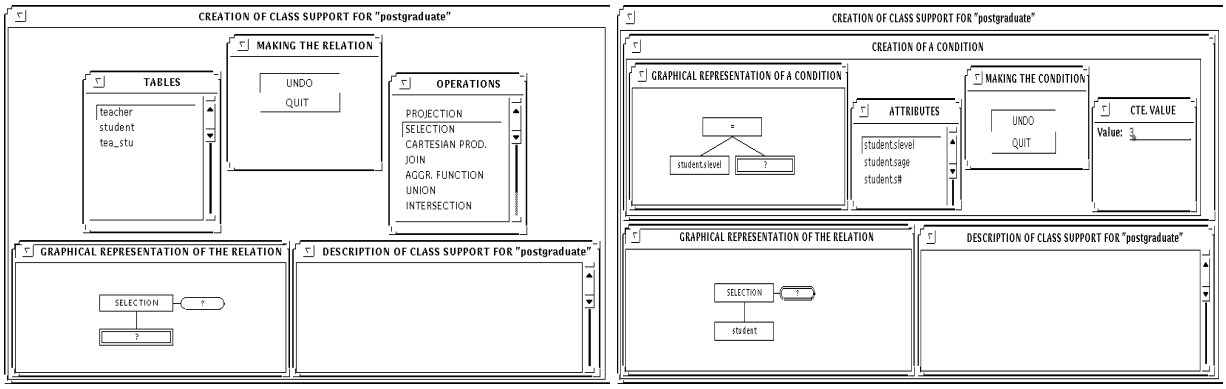

Figure 5: Creating a relation

3.1.2 Selection of attributes

The goal of this step is to select an attribute or a set of attributes that will permit the identification of objects that belong to a class. The window *Choose Attributes* will appear, this time, over the main windows. This window will contain a set of attributes that can be selected (see figure 6, left side).

Last in figure 6, right side, we present the situation when the basic class support has been constructed.

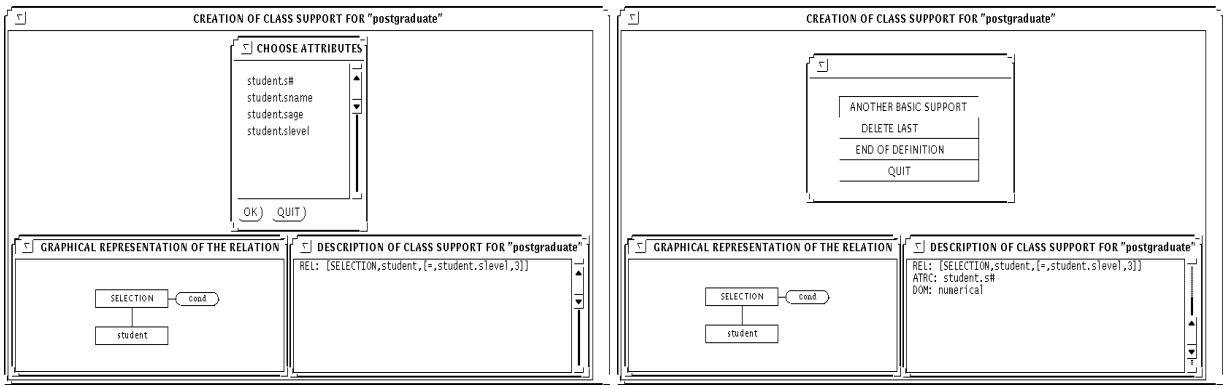

Figure 6: Two snapshots of a basic class support definition

3.2 Interface for Class Attributes

The two main steps for creating the mapping information associated to a class attribute are:

1. To select the attribute(s) of the underlying databases that will permit to obtain the class attribute values.

2. To choose a function that allows the transformation of values. For example from $ to ECU.

In general, it will happen that the mapping information associated to a class attribute will be based on the mapping information of the class for which the class attribute is defined. For this reason, the starting point offered by the system will be the mapping information of the class. However, the user is free to start from the scratch and so he could use for that the interface for generating class mapping information explained previously. In figure 7 the process of creating each basic class attribute support is described.

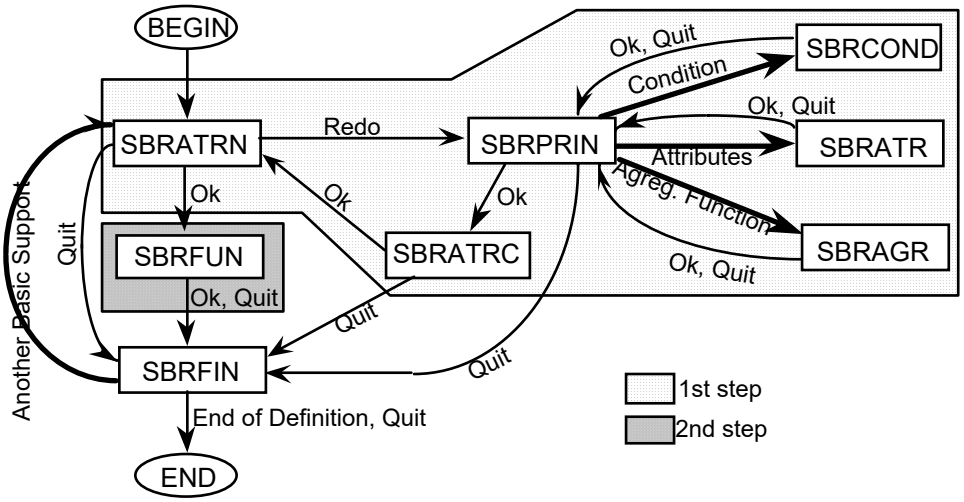

Figure 7: Definition of a class attribute support

Above the two permanent windows described in section 3, depending on the step, different windows will appear. For the first step, the window will be *Choose Attributes* explained in section 3.1.2. For the second step, the *Trans. Functions* window will appear (see figure 8).

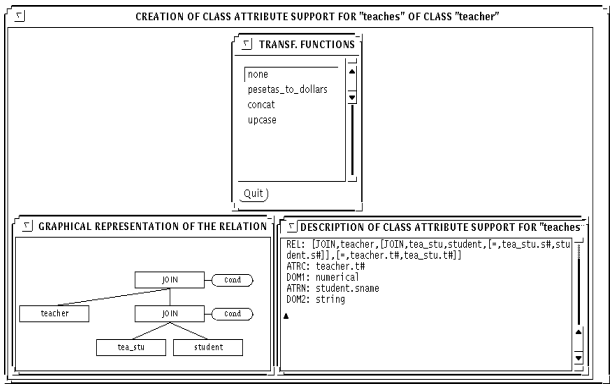

Figure 8: Choosing a transformer function

4 Conclusions

It is widely recognized the practical interest of connecting different kind of systems. An important aspect of the connexion is the definition of the mapping information that

permits to relate elements of the different systems. In this paper we have presented a system that allows generating the mapping information when connecting relational databases with object-based systems. The features of the mapping information within the user friendly interface that it provides have been explained in detail. Moreover, we have shown that the mapping information of the basic elements, classes and class attributes, can have complex structure; for this reason a user friendly interface like MAGIC which provides syntactic and semantic checking is actually very interesting.

Acknowledgements

This work was partially supported by the Spanish Government and the C.E.C. as a part of the Esprit project 5210 AIMS which involves the following participants: Datamont(I), Non Standard Logic(F), Technische Universität Berlin(D), Deuteschen Herzzentrum Berlin(D), ONERA-CERT(F), Quinary(I) and Universidad del País Vasco(E).

References

[BIG95] J.M. Blanco, A. Illarramendi, and A. Goñi. Building a federated database system: an approach using a knowledge based system. To appear in the Int. Journal on Intelligent and Cooperative Information Systems., 1995.

[BMO+89] R. Bretl, D. Maier, A. Otis, J. Penney, B. Schuchardt, J. Stein, E. H. Williams, and M. Williams. The gemstone data management system. In *Object-oriented Concepts, Databases and Applications*, 1989.

[BNPS89] E. Bertino, M. Negri, G. Pelagatti, and L. Sbatella. Integration of heterogeneous database applications through an object-oriented interface. *Information Systems*, 14(5), 1989.

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

[CHS91] C. Collet, M. N. Huhns, and W. Shen. Resource integration using a large knowledge base in CARNOT. *IEEE Computer*, December 1991.

[CLV88] P. Richard C. Lecluse and F. Velez. O2, an object-oriented data model. In *Proc. of the 1989 ACM SIGMOD Int. Conf. on the Management of Data*, 1988.

[Ill87] A. Illarramendi. *Formalización, diseño e implementación de una interfaz relacional eficaz para sistemas CODASYL.* PhD thesis, Universidad del País Vasco (EHU), July 1987.

[Lie81] Y. Lien. Hierarchical schemata for relational databases. *ACM Trans. Database Syst.*, 6, 1981.

[LMR90] W. Litwin, L. Mark, and N. Roussopoulos. Interoperability of multiple autonomous databases. *ACM Computing Surveys*, 22(3), September 1990.

[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.

[Mot87] A. Motro. Superviews: Virtual integration of multiple databases. *IEEE TOSE*, 13(7), July 1987.

[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 revised. Technical University Berlin, September 1989.

[SK92] W. Sull and R. L. Kashyap. A self-organizing knowledge representation scheme for extensible heterogeneous information environment. *IEEE Transactions on Knowledge and Data Engineering*, 4(2), April 1992.

[SPD92] S. Spaccapietra, C. Parent, and Y. Dupont. Model independent assertions for integration of heterogeneous schemas. *VLDB*, 1:81–126, 1992.

< *class support* > ::= < *basic class support* >
| < *basic class support* > < *class support* >

< *basic class support* > ::= < *relation* > < *atr list* > < *type list* >

< *class attribute support* > ::= < *basic class attribute support* >
| < *basic class attribute support* >
< *class attribute support* >

< *basic class attribute support* > ::= < *relation* >
< *atr list* > < *type list* >
< *atr list* > < *type list* >
function_name

< *relation* > ::= table_name
| projection < *relation* > < *atr list* >
| selection < *relation* > < *condition* >
| aggr. function < *relation* >
< *atr list* > < *a_function* > < *atr list* >
| join < *relation* > < *relation* > < *join_condition* >
| union < *relation* > < *relation* >
| intersection < *relation* > < *relation* >
| difference < *relation* > < *relation* >
| division < *relation* > < *relation* >

< *atr list* > ::= atr_name
| atr_name < *atr list* >

< *condition* > ::= not < *condition* >
| and < *condition* > < *condition* >
| or < *condition* > < *condition* >
| is null atr_name
| > < *operando* > < *operando* >
| < < *operando* > < *operando* >
| => < *operando* > < *operando* >
| <= < *operando* > < *operando* >
| = < *operando* > < *operando* >
| <> < *operando* > < *operando* >

< *operando* > ::= atr_name
| cte_value

< *join condition* > ::= and < *condition* > < *condition* >
| < atr_name atr_name
| > atr_name atr_name
| => atr_name atr_name
| <= atr_name atr_name
| = atr_name atr_name
| <> atr_name atr_name

< *a_function* > ::= count
| avg
| sum
| max
| min

< *type list* > ::= type_name
| type_name < *type list* >