BUILDING A FEDERATED RELATIONAL DATABASE SYSTEM: AN APPROACH USING A KNOWLEDGE-BASED SYSTEM

JOSÉ MIGUEL BLANCO

and

ARANTZA ILLARRAMENDI

and

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

(To appear in **"International Journal of Intelligent and Cooperative Information Systems"***)*

ABSTRACT

Due to the emerging interest in integrating different application environments, there have been many recent proposals for federated systems. In this paper, a federated system that permits the integration of heterogeneous relational databases using a terminological knowledge representation system is presented. In particular, two of the system's components: the translator and the integrator are explained in depth. The translator permits one to obtain a terminology from a relational schema, either semiautomatically, by expressing database properties, or manually, by using a set of predefined operations. In turn, the integrator generates a federated terminology by integrating several terminologies using the semantics expressed as correspondences between the data elements of different terminologies. Unlike many other approaches, the use of a terminological system permits us to obtain a semantically richer federated terminology and, at the same time, define a wider and more consistent integration process.

1. Introduction

In the area of information systems, the problem of the integration of different application environments has attracted substantial attention. Of special interest is the integration of database systems with knowledge-based systems, in order to take advantage of the additional features provided by the latter, without giving up the services provided by the former. The practical benefits of this type of integration apply to many different contexts, such as accessing preexisting databases when

working with knowledge-based applications, allowing databases to provide persistent object support for knowledge bases, and using a knowledge based system to integrate different autonomous databases (Interoperable Systems). Our work is mainly related to the last context, and its goal consists of building a federated system that supports the integration of heterogeneous relational databases by using a terminological knowledge representation system[1]. Nevertheless, the proposed system is also suitable for the first context, since this can be thought of as a particular type of the last. Although our particular implementation uses a concrete terminological Knowledge Representation System (KRS), BACK [2], the proposal is extendable to other terminological KRSs.

In the literature, several attempts to solve the interoperability problem and in particular database interoperability can be found. In [3] an overview of different works is presented. In general, distinct approaches to database interoperability have been proposed, aiming at different levels of integration. One type is the multidatabase approach [4], in which users can query different databases with a single request, but have to specify where the data are located. A different type of integration is federation, where a federated schema is provided to the users to formulate queries over. In this last case, location transparency and site autonomy are supported by the system. Nevertheless, not all federated approaches use the same integration methodology; some use structural integration — defining correspondences between data elements[2] of the systems that have to be integrated; others use operational integration — where correspondences are made between operations at different levels [5]. This last type permits the use of operational mappings when it is difficult to solve the semantic conflicts on a purely structural basis. Even within the structural integration approach, a variety of solutions have been presented. On the one hand, we find those that propose a manual integration [6], where the federated schema results from a PRI-driven (Person Responsible for the Integration) schema-editing process; on the other hand, we have those that claim automatic integration (for example [7], [3],[8], [9], [10]), where the responsibility for generating the federated schema lies partially with the system. Within this framework, our work follows the structural integration approach, and it provides both alternatives: manual and automatic integration. For integration it uses a relatively powerful terminological system.

Database systems integration involves many different problems, such as integrating schemata of local database systems into an integrated schema, mapping actual data from the local databases to the integrated one, mapping queries formulated over the integrated schema into queries against local database schemata, and transaction management [11]. In this paper we concentrate mainly on the first problem. Furthermore, in order to discuss how terminological systems can be used, we decompose this first problem of generating an integrated schema into two subproblems. The first is the translation of each local relational database schema into a terminology. The second is how to integrate semantically heterogeneous terminologies, obtained by a translation process, into a single integrated schema (the

[1] We understand by terminological systems (or systems based on description logics) those defined under the paradigm of KL-ONE [1].

[2] Relations and attributes for the relational model; data items, records and sets for the network, etc.

federated terminology).

The goal of this paper is twofold, on the one hand to show the advantages of using a terminological system to integrate heterogeneous relational databases, and on the other hand to present the features of two of the designed federated system's components that are involved in this integration task.

With respect to the first goal, when working with a terminological system, the specification of concepts using intensional descriptions permits richer translation and integration processes and allows us to create a semantically richer federated terminology. Moreover, the use of the classification mechanism provided by this type of system permits automatic inference of new associations among concepts not explicitly defined by the PRI, both in the translation and integration processes.

Concerning the second goal, the two components of our federated system explained in this paper are the *translator*, which is used to obtain a terminology from a relational schema, and the *integrator*, which generates a federated terminology by integrating several terminologies obtained previously by the translator component. Their main features are: 1) allowing the PRI to select between guiding the translation or integration processes or leaving it to the system; 2) the opportunity for the PRI to express rich translation or integration definitions using a set of high level operations in an editing mode; and 3) supporting the PRI in the process of generating the linking information between the resultant terminology and the underlying databases.

The integration of database systems with terminological systems has already been proposed in other work, such as [12], [13], [14] and [10]. In [12] the use of taxonomic reasoning techniques to support the conceptual design of schemata is proposed. With this aim they have incorporated taxonomic reasoning techniques to some well-known semantic data models. They maintain that the designed schemata will be more consistent. In [13] an existing terminological system (LOOM) is used as a model to describe database schemata and, a system that uses LOOM to provide efficient access to a relational database is described. However, the schemata integration task is somehow limited; they do not allow relations between data elements that belong to different schemata. Our proposal is richer in this sense. In [14] it is argued that the exploitation of pre-existing databases using a terminological system (CLASSIC) allows the generation of new information sources. Nevertheless, they assume that the integrated schema is defined previously and so the integration task is reduced to the mapping process among the local and the integrated schemata. Finally, in [10] CANDIDE, a DBMS based on description logics, is used as a tool to automate a significant part of the schemata integration process. This is the work that has more points in common with our proposal, however, the differences can be summarized in the following aspects: 1) [10] assume that all the schemata to be integrated are defined using CANDIDE, (although, they mention a method that permits the translation from the E/R model to CANDIDE [15]) so a previous translation step is not defined. In our case we deal with heterogeneous relational databases; 2) [10] center their solution around the specification of correspondences among attributes, in our case correspondences among concepts play the main role; and 3) our integration philosophy is more interactive bringing the aspects of specification and exploitation of correspondences among data elements of different terminologies closer to each other.

In the remainder of this paper we present first the general framework for the system components explained in this paper followed by a brief introduction to terminological systems. Next, we explain in detail the translator and integrator components, and then we describe the features of the mapping information.

2. Work's framework

The general framework of the designed federated system that allows the integration of heterogeneous relational databases using a terminological system is presented in this section. In the framework, three main components can be distinguished[3]: Translator, Integrator and Query Processor.

The Translator component produces a terminology from a conceptual schema (or a subset of it, called exported schema) of a component database. The resultant terminology will be semantically richer than the source schema, therefore this component has to capture, with the PRI's help, semantics that are not expressed explicitly.

The Integrator component produces a federated terminology by integrating a set of terminologies previously obtained by the Translator component. During the integration process, a set of correspondences between data elements of the terminologies that must be integrated will be defined by the PRI and new ones can also be deduced by the system.

The Query Processor component obtains the answer to user formulated queries over the federated terminology by accessing the databases. This component has two kinds of modules: the Global Query Processor and the Local Query Processor. The first one optimizes knowledge-base queries, 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 main advantages that the use of a terminological system provides for this module are: a semantic optimization of queries using the classification mechanism[4]; support for defining and identifying cached data; and the possibility of giving intensional answers. Moreover, the goals of the second module are to make local optimizations and to find the answer for the subqueries (a deeper explanation of this component is beyond the scope of this paper).

3. A brief introduction to terminological systems

In general, two different components can be distinguished in this type of system: the *terminological* component, which is used to describe the knowledge (the terminology), and the *assertional* component, which is used to create instance objects that represent the beliefs of the system.

[3] Actually, there is another component, called Controller, that is not discussed in this paper, because it does not take advantage of any particular feature of terminological systems. The Controller responds automatically, i.e., without user intervention, to design changes made in the schema of a component database that affect the federated terminology [16].

[4] The meaning of the classification mechanism for the terminological systems is explained in the next section

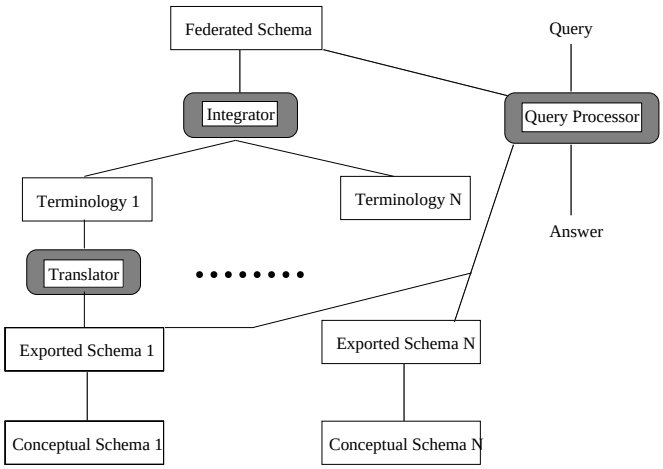


Figure 1: Work's framework

In the terminology, two main types of data elements are included: concepts and roles[5]. A *concept* groups individual elements of the real world. For example, the concept *employee* represents the set of employees of a company. However what distinguishes this notion of concept from the class specification in semantic data models or object-oriented databases is that it is possible to describe concepts using intensional descriptions phrased not only in terms of necessary properties that must be satisfied by their instances (in this case the concept is called a *primitive concept*), but also in terms of necessary and sufficient properties (in this case the concept is called a *defined concept*) [18]. This is accomplished using a language for describing primitive concepts and then combining them into composite ones.

Examples: The concepts *client* and *manager* can be described as primitive concepts in the following way:

client :<[6] *anything*[7].

manager :< *employee and Category:'BS degree'*

This means that a *manager* is an employee who has a BS degree, but there can exist employees with a BS degree who are not managers.

slowpayer can be described as a defined concept:

slowpayer :=[8] *client and all(Payment,ge(40))*

slowpayer are *clients* who make payments in a period greater or equal to 40 days, and because it is a defined concept, any client who makes payments in a period greater or equal to 40 is a *slowpayer*.

[5] We use the BACK system notation. In other systems the components are denoted in a different way, for example as classes and attributes in [17].

[6] :< is used, according to the BACK syntax, to describe primitive concepts.

[7] *anything* is a concept such that any instance can belong to it.

[8] := is used, according to the BACK syntax, to describe defined concepts.

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 have).

Example: The roles *Payment* and *Buys* can be described as
Payment :< *domain(client) and range(number)*,
Buys :< *domain(client) and range(product)*.
Both roles are associated with the concept *client*; *Payment* takes numeric values and *Buys* has as values instances of the concept *product*.

Two important features in terminological systems are the notions of *subsumption* and *classification*. One concept *subsumes* another one if in all possible circumstances, any instance of the second one must be in the first one. In a terminological system, it is possible to know whether one concept *is subsumed* by another one simply by looking at the definition of the concepts, without accessing the instances. The *classification* mechanism consists of discovering the subsumption relationships between concepts when a new concept is declared, i.e. the new concept is automatically located into the hierarchy of terms; therefore, concepts viewed as composite descriptions can be reasoned with and are the source of inferences.

Example: Introducing a new defined concept *government* described as
government := client and all(Payment,ge(60)) in the knowledge base that contains the concepts *client* and *slowpayer* presented above, the system will automatically classify it under the *slowpayer* concept.

When using a terminological system such as BACK to build a federated database system, the resultant federated schema is represented by a terminology. But the actual data are stored in the underlying databases. For this reason it is necessary to define linking information between the federated terminology and the local databases.

Finally, making a brief comparison between the well-known E/R model and the terminological systems[19], we could find a resemblance among the notions of entity and attribute of the E/R model with the notions of concept and role respectively of the terminological systems, but, in the latter case, concepts and roles are attached with intensional descriptions. In the terminological systems a specific constructor does not exist, as it exists in the E/R model, to represent relationships. These are represented through the roles. Moreover, terminological systems provide structuring mechanisms such as specialization and the previously mentioned classification mechanism that are not supported by the basic E/R model. Lastly, terminological systems support an internal identification of the instances which is different than the E/R, which provides an external identification using the notion of key.

4. Example

In this section the general features of the example that will be used in the paper are presented. The input is two different relational databases defined for two distinct companies (see the corresponding schemata in figure 2). The first company has

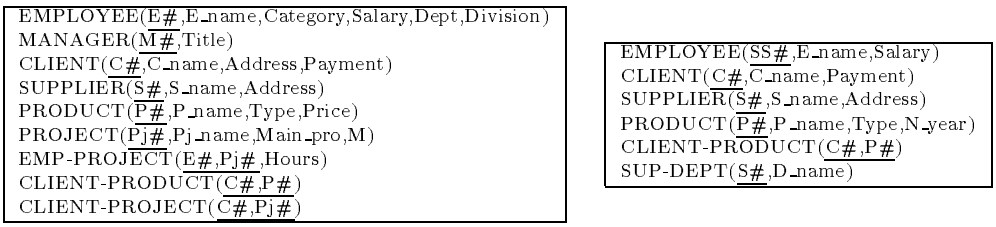


Figure 2: Simplified version of two relational database schemata

two business lines, the manufacturing of electronic components and the execution of engineering projects. The schema, on the left side of the figure, corresponds to this company. In the schema, the employees (distinguishing those that have the category of managers), the clients, the suppliers that they work with, the manufactured products, the executed projects and the main binary associations between employees, clients, projects and products are represented. The second company only manufactures electronic components, the corresponding schema is represented on the right side of the figure.

In order to enlarge their market share, the first company absorbs the second one, although this last company preserves the autonomy of its management. Therefore, new applications will appear that require an integrated view of the two previously mentioned databases. In the following sections first of all, a BACK Terminology will be obtained for each schema (explaining in detail the steps followed only for the schema on the left side of the figure) and then, the generated terminologies will be integrated in order to obtain the federated terminology.

5. The Translator component

5.1. General Overview

The goal of this component is to generate a terminology taking as input a relational database schema definition and some properties of the corresponding database. Moreover, it must generate the mapping information that relates the terminology data elements to the underlying database. The features of the mapping information are explained in section 7..

Several proposals have been presented in the literature in the area of schema translation. In general, they can be classified in three different groups: 1. Translations from a hierarchical or network model to a relational model; 2. Translations from the Entity-Relationship model (or some of its extensions) to the relational model; and 3. Translations from a relational model to semantic, object oriented or object based models. In the first and third groups translations are made to models that offer abstraction mechanisms of higher level, but in the second group the opposite occurs. The translator presented in this paper belongs to the third group. Its goal is to translate from a relational database schema to a terminology.

By analysing different approaches of the third group[20, 21, 22, 23, 24, 25] in

more detail it can be concluded that: 1) because the translation occurs from the relational model to a semantically richer one, the collaboration of a person responsible for the translation[9] is needed; 2) the translation process is based usually on the definitions of keys (primaries or candidates) and definitions of inclusion dependencies. The use of functional, multivalued and exclusion dependencies is considered in a limited number of cases (e.g. [26, 22]). 3) In some situations, a translation cannot be obtained from a relational schema by using just a general translation process (even with the PRI collaboration). For those situations an *ad-hoc* translation is required. This last point permits the distinction in the group of two strategies [27], one called the manual approach and the other semi-automatic. The first approach develops explicit mappings between the source and the target schema. This means that the PRI must specify mappings that transform the source schema to the target schema, usually by using a predefined set of operations [6]. The second one develops mapping rules to generate the target schema from the source schema, and it has been studied by many authors in different contexts, including [22, 28, 29, 30]. It is our belief that for many practical reasons it is useful to automate the translation process so that the PRI is freed of this responsibility. However, in many situations he may be interested in obtaining a target terminology that does not follow the general translation rules. To provide support for both situations, we have designed and implemented a translator that offers both possibilities: that is, for the PRI to guide the translation process (*editor way*), and, to leave partially the translation process responsibility to the system (the *semi-automatic way*).

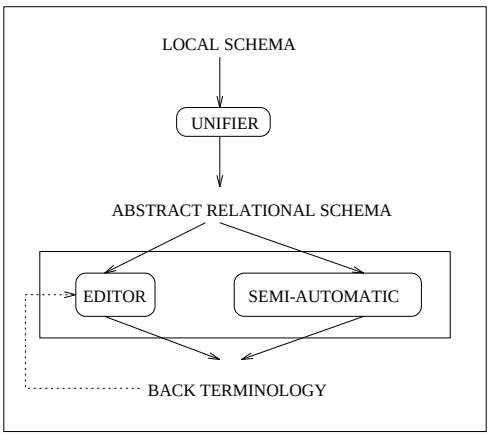


Figure 3: Translator component.

Furthermore, in our case each local schema can be expressed using a distinct relational database management system, and hence, different local schemata may be expressed with different relational systems. In order to avoid designing the whole translation process for each type of relational database management system, we have

[9] The translation and integration processes can be directed by the same person, so hereafter we will call him PRI.

defined an abstract relational schema between the local and the BACK terminology. This schema represents relational data elements (such as relations, attributes or key definitions, etc.) in a form that is independent of the underlying relational systems. *Unifier* is the component that translates the local schema to the abstract relational schema. There will be one unifier for each type of relational database management system (see figure 3). Moreover, this design permits us to extend the applicability of the translation process to other data models, i.e., hierarchical or network, implementing for each one a unifier based on previous work by merely stating the transformation rules of the schemata into the relational model [31, 30].

5.2. Translation Process: Main features

The translation process can be divided in two main steps: in the first one, an abstract relational schema is obtained and in the second one the translation from a relational schema to a terminology, in the strict sense, takes place. This last step can be carried out using the semi-automatic or the editor module. In any case, first of all a default terminology is created which is enriched later on.

The main data elements that will be involved in the translation process, as far as the relational model is concerned, are relations, attributes, and integrity constraints. For terminological systems, they are concepts and roles, which are structured in a taxonomy according to their descriptions.

The following three general transformation rules are used by the Translator component to obtain a default terminology:

- Relations are translated into concepts, e.g., the concept *employee* is generated from the relation *EMPLOYEE*.
- Attributes are translated into roles that have as a domain the concept related to the relation in which they have been defined. The range of the roles will be the type of the corresponding attributes defined in the relational schema. For example, the role *Salary* described as *domain(employee) and range(number)* is obtained from the attribute *EMPLOYEE.Salary*.
- Integrity Constraints will be translated into descriptions associated with the roles, e.g., the integrity constraint that expresses that the salary of all employees must be greater than \$1000 allows the extension of the description of the role *Salary* in the following way:

  $Salary :< domain(employee)\ and\ range(number)\ and\ range(gt(1000))$

5.3. Semi-Automatic module

This module receives as input a default terminology and then enrichs it by using information about dependencies (inclusion, exclusion, and functional dependencies), null values, and semantic properties (domain information for attribute values). This type of information is provided, most of the time, by the PRI, because it is rarely available from the database system. From the operational point of view, when a property is expressed it is immediately exploited by the system in the creation of the terminology. The exploitation can sometimes imply the inference of new relationships due to the reasoning capabilities offered by the terminological system.

(i) **Dependencies.** There are three different dependency types being considered when generating a terminology: *inclusion*, *exclusion*, and *functional*.

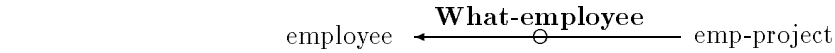
1.1 Inclusion dependencies formalize interrelation constraints [11]. They can represent referential integrity constraints or class/subclass relationships, and have been exploited, in several works, when translating from the relational model to others, such as the object oriented model [22, 24] or an extension of the entity relationship model [21]. In our case, they are used to obtain a terminology from a relational schema. Below, the different situations are presented.

(a) **Dependencies between keys**
Let K and K' be two keys defined for the relations R and R' respectively. The existence of an inclusion dependency, R.K $\subseteq$ R'.K', permits the definition of the concept associated with R as a specialization of the concept associated with R'.

For example, the inclusion dependency *MANAGER.M#* $\subseteq$ *EMPLOYEE.E#* is used to define a taxonomic relationship between the concepts obtained from these relations.

(b) **Dependencies between a subset of a key and a key**
Let K and K' be two keys defined for the relations R and R' respectively. The existence of an inclusion dependency, R.subK $\subseteq$ R'.K' (where subK refers to a subset of the attributes that constitute K) permits the definition of an association between the concepts obtained from the relations.

For example, the inclusion dependency *EMP-PROJECT.E#* $\subseteq$ *EMPLOYEE.E#* between the relations *EMP-PROJECT* and *EMPLOYEE* is used to define an association between the concepts obtained from these relations by incorporating a new role *What-employee* with domain in *emp-project* and range in *employee*.



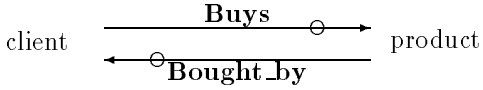
In the case that R has only two attributes that represent a binary association and there exist inclusion dependencies defined for them, the concept generated from the relation R is eliminated from the terminology and the two roles associated with that concept change their descriptions. For example, the inclusion dependencies:

CLIENT-PRODUCT.C# $\subseteq$ *CLIENT.C#* and

CLIENT-PRODUCT.P# $\subseteq$ *PRODUCT.P#*

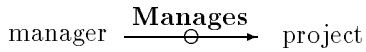
permit the addition[10] of two new roles, *Buys* and its inverse, to the concepts *client* and *product*, respectively. Moreover, the concept *client-product* is eliminated from the terminology.

[10] Although we use specific role names for clarity, the system is not able to generate names of that type automatically.

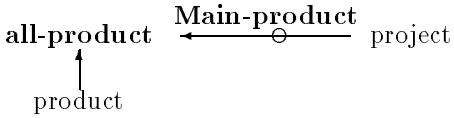


(c) **Dependencies between key and nonkey attributes**
Let K be a key of a relation R and X a set of nonkey attributes of another relation R'. Two different situations can arise:

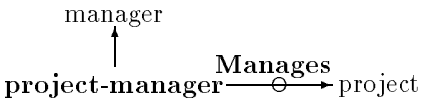
- R'.X $\subseteq$ R.K
  This situation reflects the existence of a foreign key and permits the definition of an association between two concepts. For example, the inclusion dependency $PROJECT.M \subseteq MANAGER.M\#$ is used to establish one association between the concepts *manager* and *project* through the role *Manages*[11].



- R.K $\subseteq$ R'.X
  This means that R' is representing in fact more than one entity. So, first, a new concept is generated that includes the instances that correspond to the values of the nonkey attribute(s) of R' and then, an association is established between the new concept and the concept obtained from the relation R'.
  For example, in the case that the following inclusion dependency could be defined, $PRODUCT.P\# \subseteq PROJECT.Main\_pro$, it would be used to create a new concept *all_product* and an association through the role *Main-product* defined for the concept *project*.



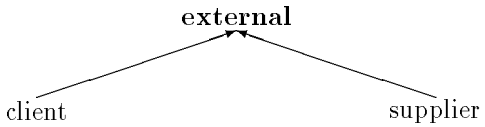
For cases *(b)* and *(c)*, moreover, when the inclusion dependency corresponds to a proper subset, a new concept can be generated. This concept will be a specialization of the concept obtained from the relation that appears in the left-side of the inclusion dependency.
For example, in the case that the inclusion dependency $PROJECT.M \subseteq MANAGER.M\#$ would have been defined as $PROJECT.M \subset MANAGER.M\#$ the result of its application would be:



1.2 Exclusion dependencies. Exclusion dependencies as a useful information source during a translation process appear in [26]. The information about

[11] The existence of the role *Manages* implies the existence of its inverse role *Managed_by*, but it is not added to the terminology unless the user wants to.

an *exclusion dependency* between key attributes of two different relations provides for the addition of information about mutual exclusion to the descriptions of the concepts generated from the relations. For example, if the following exclusion dependency were defined, $CLIENT.C\# \cap SUPPLIER.S\# = \emptyset$, the description of the concept *client* would be increased with the property *and not (supplier)*. Moreover, if a concept in the terminology that generalizes both concepts does not exist, then a new concept as a generalization of the two concepts would be created. For example, the previous exclusion dependency would be used to create a new concept *external* (external-organization) as a generalization of *client* and *supplier*.



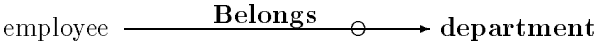
1.3 Functional Dependencies. Partial functional dependencies and transitive functional dependencies enable the translator to generate new concepts in the same way new relations are created when they are found during the normalization process.

For example, if the following functional dependencies are expressed:

$EMPLOYEE.E\# \rightarrow EMPLOYEE.Dept$ and

$EMPLOYEE.Dept \rightarrow EMPLOYEE.Division$

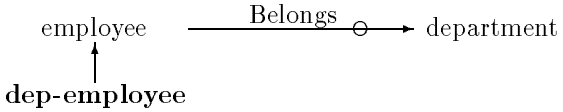
a new concept *department* is created and a new role, *Belongs*, representing the link between *employee* and *department*, is added. Furthermore, the roles corresponding to the attributes functionally dependent on *Dept* become now roles of the new concept. So, the role *Division* of *employee* would become a role of *department* now.



(ii) **Null Values.**

A second type of information requested from the PRI is about *null values*. Tuples of a relation can take a *null value* for one or more of their attributes. Null values can have multiple interpretations [11]; here we refer to the case that *the attribute does not apply to the tuple*. This information enables the creation of a specialization of the concept that contains at least one value for the role supported by the attribute.

As an example, consider that *EMPLOYEE.Dept* can have NULL values, the terminology can be increased with a concept, *dep-employee*, which is a specialization of *employee*, and represents employees that belong to one department at least.

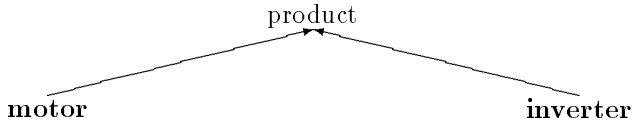


dep-employee := employee and atleast(1,Belongs)

(iii) **Semantic Information.**

Domain information for attribute values can also be used to obtain richer schemata. In this case, we distinguish between properties expressed by enumerating each set of attribute values that defines one conceptual unit and those expressed by giving the condition that characterizes each set of attribute values that defines a conceptual unit.

To illustrate the first case, consider that in the example, *PRODUCT.Type* can have the values ACB, ACA, DCP, DCL, INVA or INVB. The values ACB, ACA, DCP and DCL correspond to products of type *motor*; INVA and INVB correspond to products of type *inverter*. Assuming that the classification of products depending on the value of this attribute is relevant, two new concepts will be added to the terminology.

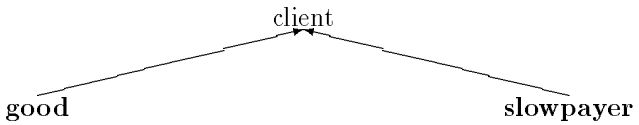


Where *motor* and *inverter* are defined concepts described as:

```
motor := product and all(Type,motortype)
```

```
inverter := product and all(Type,invertertype)
```

To illustrate the second case, let us assume that it is possible to distinguish two kind of clients depending on the value that the attribute *Payment* takes. If this attribute has a value less than or equal to 30, then the client is considered *good*, and if the attribute takes a value greater than or equal to 40, then the client is considered as a *slowpayer*.



The following are the definitions of *good* and *slowpayer*:

```
good := client and all(Payment,le (30)) and atleast(1,Payment)
```

```
slowpayer := client and all(Payment,ge (40)) and
atleast(1,Payment)
```

4. The order for the application of properties

All the previous types of properties are applied in the following sequence of steps:

- First, inclusion dependencies are exploited. Depending on the DBMS, these type of dependencies can be obtained from the system catalog or defined explicitly by the PRI.

- Next, when the input relational schema is not in second or third normal form, functional dependencies, expressed by the PRI, are used to create new concepts.
- Then, exclusion dependencies are exploited. Some of them are presented as candidates by the Translator using as criteria the agreement on names or types of key attributes. New ones can be defined by the PRI.
- Then, the translator presents to the PRI integrity constraints taken out from the database system catalog and the PRI decides which information to exploit in order to enrich the terminology.
- Last, domain information for attribute values is considered. This type of information is defined explicitly by the PRI.

In figure 4, the terminology obtained from the relational schema represented on the left side of figure 2 by exploiting some of the properties[12] explained throughout section 5.3. is shown[13].

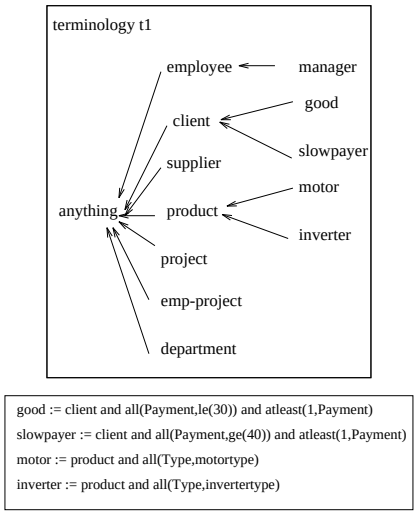


Figure 4: BACK terminology

5.4. Editor module

The goal of this module is to permit the PRI to create a terminology in a flexible way. He can choose to start from scratch, from the default terminology offered by the system (in this case the terminology will be obtained by using the general translation

[12] Only some of the properties have been selected in order to make clearer the integration process.
[13] Only the description of the concepts that help to understand the integration example are included.

OPERATION	*arguments*
ADD-CONCEPT	*name,concept-description,concept-mapping*
ADD-ROLE	*name,role-description,role-mapping*
GENERALIZE-CONCEPT	*list of concept-names,description*
GENERALIZE-ROLE	*list of role-names,description*
SPECIALIZE-CONCEPT	*concept-name,description*
SPECIALIZE-ROLE	*role-name,description*
DELETE-CONCEPT	*concept-name*
DELETE-ROLE	*role-name*
REDESCRIBE-CONCEPT	*concept-name,description*
REDESCRIBE-ROLE	*role-name, description*
MODIFY-CONCEPT-MAPPING	*concept-name,concept-mapping*
MODIFY-ROLE-MAPPING	*role-name,role-mapping*

Figure 5: Set of editing operations provided to the PRI

rules) or from the terminology obtained using the Semi-automatic module. In all cases, the PRI must apply a set of operations to obtain the desired terminology.

It is worth mentioning that most of the properties used by our Semi-automatic module, and in other similar work, are related to the relational database schema. This means that the possibilities of generating a richer target schemata are limited. However, dealing with the Editor module, translations can be expressed using semantic definitions and so the resultant terminology can be semantically richer.

5.4.1. Operations

On top of a terminological system, a set of high level operations oriented to the translation process have been defined, so as to allow the PRI to make the translation directly (see figure 5). Although some of these operations look like the usual knowledge base operations, notice that they manage the mapping information between the terminology and the underlying database, which in any other case would be a user responsibility. The basic features of this mapping information will be explained in section 7..

As mentioned in section 3., two types of terms are distinguished in a terminology: primitive and defined. For this reason first of all, two types of operations were designed, operations to add primitive terms to a terminology and operations to add defined terms. In the former case, the following operations were included: ADD-CONCEPT, ADD-ROLE, GENERALIZE-CONCEPT, and GENERALIZE-ROLE. For the first two operations, the description of the mapping information that relates data elements of the terminology to relational tuples is the PRI responsibility, because he establishes a concrete link. The operation ADD-CONCEPT permits the addition of a new concept based on the input concept mapping information. This new concept will be described as primitive using the input description. GENERALIZE-CONCEPT permits the addition of a new primitive concept, defined as a generalization of the concepts that appear in the input list, with the input description. Its mapping information will be obtained using the mapping information of the concepts that appear in the list. Furthermore, the concepts of the list will

be redescribed according to the new created concept.

Operations to add defined terms, SPECIALIZE-CONCEPT and SPECIALIZE-ROLE, are also available. For these operations, the system automatically generates the mapping information. In order to obtain this, it uses the mapping information descriptions associated with the terms that appear in the definition of the new one.

With the previous operations a terminology can be constructed; however, later on it could be useful or necessary to alter it by removing terms, redescribing terms or modifying the mapping information associated with terms. Taking this into account, the operations DELETE-CONCEPT, DELETE-ROLE, REDESCRIBE-CONCEPT, REDESCRIBE-ROLE, MODIFY-CONCEPT-MAPPING, and MODIFY-ROLE-MAPPING are incorporated. The operation DELETE-CONCEPT permits the deletion of a concept. This means that the concept will be eliminated from all the places where it appears. The REDESCRIBE-CONCEPT operation permits the modification of the description of the input concept, which was described as primitive, maintaining the same mapping information. The MODIFY-CONCEPT-MAPPING operation permits the substitution of the mapping information associated with a concept by new concept mapping information.

Only operations defined for concepts have been explained. The corresponding operations for roles have a similar semantics.

5.4.2. Example

Starting from the terminology obtained by using the Semi-automatic module (see figure 4), the PRI could wish to delete the concept *emp-project* and to add a new role *Participates* to the concept *employee* having this role as a range the concept *project*. With this aim he would use the following operations :

```
DELETE-CONCEPT(emp-project)
ADD-ROLE(Participates, domain(employee) and range (project),
"mapp-inf"[14])
```

Later on, and with the goal of refining the terminology, the PRI could use the operation:

```
SPECIALIZE-CONCEPT(active-employee, employee and
atleast(2,Participates))
```

to obtain a new concept, *active-employee*, for which the terminological definition *employee and atleast(2,participates)* is provided. In this case, the mapping information will be automatically generated because the mapping information for *employee* and *Participates* are known (their concrete expressions will appear in figure 22). Notice that it has been possible to define the concept *active-employee* in terms of the concept *employee* using an intensional description. In general, the specification of concepts using intensional descriptions permits a richer translation process, widening the range of translation possibilities.

In this way, the PRI could continue the modification of the terminology by using one operation at each step.

In figure 6, two terminologies *t1* and *t2* can be seen; *t1* is the terminology obtained after modifying the terminology that appears in figure 4 by using the

[14] This mapping information is defined by the PRI. Its concrete expression appears in figure 22.

editor module and *t2* is a terminology that has been obtained from the database schema on the right side of figure 2.

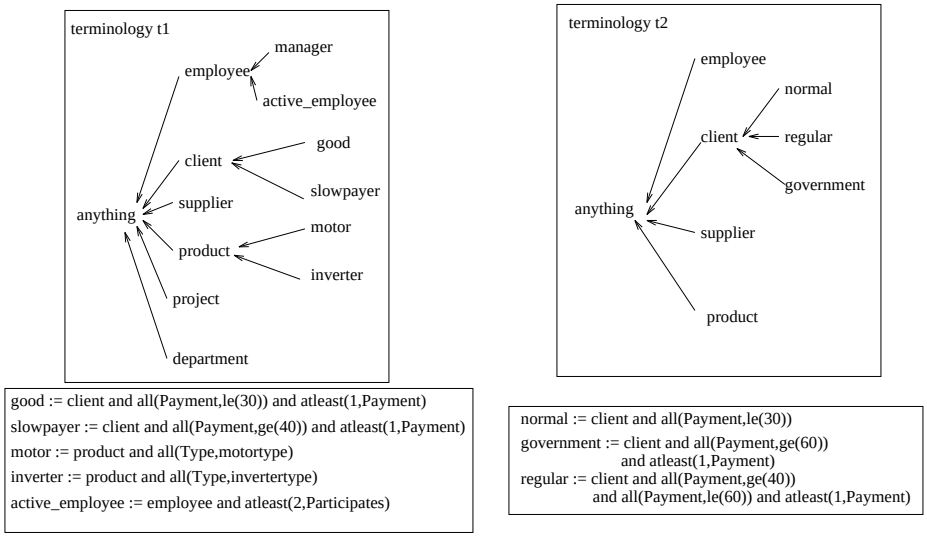


Figure 6: Terminologies obtained after the translation process

6. The Integrator component

The goal of this component is to integrate different terminologies, previously obtained from relational databases by the Translator component, in order to generate the federated terminology. In general, it would have been also possible to integrate the relational schemata directly. However, taking into account that the integration process relies mainly on the semantics associated with data elements of different schemata, it is more advantageous to do it with semantically richer schemata.

For the integration process this component uses a terminological system, unlike other work such as [3] or [8], which use the entity relationship model, or [32], which uses the object oriented data model. The integration process is not based solely on the notion of Real World State (RWS), that is, the semantic of correspondence assertions among data elements of different databases is defined referring not only to the real world counterpart of elements in the schema as in [3], [8]. Finally, although the system, as well as [9, 10], works with a knowledge based system, it does not provide any predefined knowledge base against which to match the schemata to be integrated, as done by [9]. However, the history of the integration process is stored and so that repetition of parts of or the total integration process can be avoided on subsequent occasions. Also in comparison to [10] that base their solution around the specification of correspondences among attributes, in our case correspondences among concepts play the main role. Furthermore, unlike other work in the integration area, this component provides the PRI with the possibility of choosing between

a semi-automatic integration or a manual one. In the former case, the PRI asserts the correspondences among data elements of different terminologies, and then the system applies the integration rules. In the manual approach, the PRI can use a predefined set of high level operations to create a federated terminology of his interest. The semi-automatic approach has the advantage that the main responsibility of the generation process relies on the system but, on the contrary, most of the times the resultant federated terminology will not totally satisfy the PRI. Using the manual approach, qualified users can obtain semantically richer federated terminologies. However, the task of integrating is difficult and hence it could be cumbersome for the PRI to do. In both approaches, due to the use of a terminological system, the system is able to deduce new correspondences, among components of different terminologies, not defined explicitly by the PRI. Moreover, during the integration process the system has also the goal of managing the mapping information that relates the data elements in the federated terminology to the underlying databases.

Briefly described, our integration method generates first a default federated terminology[15] and then gives the opportunity to the PRI to select between the semi-automatic or the manual approach. Depending on the selection, the behaviour is different (the particular features are explained in the following subsections). Once a federated terminology has been obtained, it can always be modified using the manual approach.

It is worth mentioning that, in both approaches, the definition and exploitation of correspondences among data elements of different terminologies are not made separately. Every time the PRI expresses a correspondence, it is exploited and, as a consequence, the terminology is modified and new correspondences can be inferred. Furthermore, before the correspondences are exploited it is verified if they are consistent. The verification of correspondences and the inference of new ones using the features provided by terminological systems are two important aspects that are not supported when using an object-oriented database system or the E/R model to do the integration. These aspects are mainly related to the fact of dealing with defined concepts.

In the following subsections we show the features of the semi-automatic approach and of the manual one.

6.1. Semi-automatic approach

As mentioned before, in this approach the PRI asserts the correspondences among data elements of different terminologies, that represent somehow the conceptualization of the same part of a Universe of Discourse (UoD), and then the system applies the corresponding integration rules. In the following, we present in sequence the integration strategy, the type of correspondences that the approach deals with, and the integration rules.

[15] A default federated terminology is a terminology defined with the *anything* concept as root and the different terminologies that must be integrated under it.

Input: Default federated terminology and the Dictionary with candidate correspondences.
Output: Federated terminology

1 Obtain a set of candidate correspondences
2 **while** the PRI wants to introduce a correspondence **do**
    Verify the correspondence applicability
    If applicable **then**
      Exploit the correspondence
      Integrate the roles
    endif
  endwhile
3 If necessary apply the manual approach

Figure 7: General description of the semi-automatic integration method

6.1.1. Integration Strategy

In figure 7 the general algorithm that describes the integration strategy is shown. In the first step, before the PRI expresses any correspondence between concepts, the system attempts to obtain candidate correspondences, mainly by using a dictionary that contains information of correspondences between concepts and roles that have been defined during the life of the Integrator component. However, notice that in all cases the system handles the selected correspondences as candidates and it is the PRI's responsibility to confirm them. If candidates are not found or if they are not selected by the PRI, then the PRI must provide the properties that relate data elements in terms of the correspondences that will be introduced in the section 6.2..

In the second step, for each established correspondence, the system:

- verifies the applicability of that correspondence. This is accomplished by contrasting the descriptions associated to the concepts with the introduced correspondence.
- exploits the correspondence, by using the proper rules presented in section 6.3. and it continues with the integration of roles defined for those concepts. Where role integration implies the integration of new concepts, this will be defined as a pending task. In this step it is possible for the terminological system (in our case, BACK) to detect new correspondences between concepts automatically. The inference of new correspondences (equivalent, inclusion or disjoint) is another advantage of working with a terminological system.

In the last step the resultant federated terminology can be modified by the PRI by using the set of operations defined for the manual approach. It is at this moment when the user could remove from the terminology redundant information that is difficult to detect automatically.

6.2. Definition of correspondences

The UoD of the underlying local systems plays an important role during the integration process. In particular, it is necessary to know the semantics associated with data elements that must be integrated. In the literature, semantics are defined in terms of Real World State (RWS) [8, 3]. However, the RWS notion is not *absolute* because it defines semantic information, i.e., different authors give it distinct meanings. In our case, we define the RWS in terms of the basic components of a terminology: concepts, roles, and attributive types[16]. On the other hand, it is our belief that the notion of RWS is *not* independent of the final goal: to integrate data elements. Moreover, we think that the RWS definitions depend on the context where the integration takes place. For example, the RWSs of *animal* and *person* could be related in a *biological* taxonomy, and on the contrary, they could be unrelated in a *farm* environment.

Our definitions of RWS notions are the following:

- *RWS of a concept A* (RWS(A)) is the set of real world objects represented by the instances of A at one state of the KB.
- *RWS of an attributive type T.* The RWS of an attributive type T, which serves as range of a role a_i (RWS(T $\rightarrow$ a_i)), is the set of real world objects represented by the values of a_i in the knowledge base at one state of the KB.
- *RWS of a role* a_i (RWS(a_i)), where a_i has as domain A and as range r, is a set of pairs of real world objects <x,y>, where x belongs to the RWS(A), y is an object of the RWS(r) and both are related by a_i.

For example, the terminology *t1* that we deal with, includes the concept *client* and the role *Payment* defined with domain *client* and range *number*. Their RWSs at a given moment could be:

RWS(client) = {FAGOR,BH}

RWS(number $\rightarrow$ Payment) = {25,35}

RWS(Payment) = {<FAGOR,25>,<BH,35>}[17]

Considering that the terminologies have been obtained from local schemata that were designed independently, we can find the conceptualization of the same part of a UoD represented in a different way. For example, in one terminology it can be represented as a concept, while in the other as a role. In general, the types of conflicts among terminologies found in our context are: 1) *Structural conflict.* The same part of a UoD is represented using terminologies with different concepts and roles. 2) *Intensional conflict.* The same real world data element is represented in two terminologies with different descriptions. 3) *Extensional conflict.* The sets of instances of two related data elements are included, overlapping or disjoint. These types of conflicts can occur at the same time and, in fact, discrepancies among terminologies usually show a mix of conflict types. All conflicts are addressed by

[16] Here, we use the BACK system notation. In other systems *attributive types* are denoted as *scalar* or *printable* types.

[17] Notice that *FAGOR* or *25* are representations of the corresponding objects of the real world.

the integrator component. To deal with structural conflicts, another type of correspondence (apart from the natural correspondences between concepts or roles) has been defined between a concept and a role. The other two kinds of conflicts are considered by the integration rules.

Finally, notice that the types of correspondence that we define in the following must be time-invariable properties, because otherwise taking into account the autonomy property of the underlying databases the federated terminology could become inconsistent with respect to the databases.

6.2.1. Types of correspondence between concepts.

Let A and B be two concepts from two different terminologies. The following types of correspondence can be defined between A and B:

- *A is equivalent to B* ($A \equiv B$), means that RWS(A) = RWS(B) is always verified.
- *A is included in B* ($A \subseteq B$), means that RWS(A) $\subseteq$ RWS(B) is always verified.
- *A overlaps with B* ($A \cap B \neq \emptyset$), means that objects can exist in the RWS(A) that are also in RWS(B), but at some moments RWS(A) $\cap$ RWS(B) can be the empty set.
- *A is disjoint with B* ($A \cap B = \emptyset$), means that the real world objects of A and the real world objects of B are of the same kind, but RWS(A) $\cap$ RWS(B) = $\emptyset$ is always verified.

For example, for the concepts *t1_client* and *t2_client* from the two terminologies considered, the following different situations could arise (although only one of them could be valid):

- They could represent the same set of clients, hence *t1_client* $\equiv$ *t2_client*.
- The clients represented by *t1_client* could be a subset of those represented by *t2_client*, hence *t1_client* $\subseteq$ *t2_client*.
- Some of the clients represented for both *t1_client* and *t2_client* could be the same, hence *t1_client* $\cap$ *t2_client* $\neq \emptyset$.
- They could represent disjoint sets of clients, hence *t1_client* $\cap$ *t2_client* $= \emptyset$.

6.2.2. Types of correspondence between roles.

Let a_i and b_j be two roles from two different terminologies, where a_i is a role of the concept A
($a_i :<$ domain(A) and range(r_i)) and b_j is a role of the concept B ($b_j :<$ domain(B) and range(r_j)).

- If A and B are related by one of the following correspondences: A $\equiv$ B, A $\subseteq$ B, B $\subseteq$ A or A $\cap$ B $\neq \emptyset$, then the roles a_i and b_j can verify one of the next properties:

(i) $a_i \equiv b_j$
If $\forall x \in [RWS(A) \cap RWS(B)]\ \forall y \in RWS(r_i) \cup RWS(r_j)$
$((<x,y> \in RWS(a_i) \leftrightarrow <x,y> \in RWS(b_j))$

(ii) $a_i \subseteq b_j$
If $\forall x \in [RWS(A) \cap RWS(B)]\ \forall y \in RWS(r_i)$
$((<x,y> \in RWS(a_i) \rightarrow <x,y> \in RWS(b_j))$

(iii) $a_i \cap b_j \neq \emptyset$
At any time a pair <x,y> can exist such that $<x,y> \in RWS(a_i) \cap RWS(b_j)$.

(iv) $a_i \cap b_j = \emptyset$
If $\forall x \in [RWS(A) \cap RWS(B)]\ \forall y \in RWS(r_i)$
$((<x,y> \in RWS(a_i) \rightarrow <x,y> \notin RWS(b_j))$
As in the case of disjoint concepts, these roles must refer to the same kinds of objects.

For example, for the two different roles, *t1_Buys* and *t2_Buys* with domain in the concepts *t1_client* and *t2_client* described respectively as:

```
t1_Buys :< domain(t1_client) and range(t1_product)
t2_Buys :< domain(t2_client) and range(t2_product)
```

For *each object* of the RWS(*t1_client*) that has its correspondent in the RWS(*t2_client*) the following situations may arise:

- the set of products represented by both roles could be the same, hence *t1_Buys* $\equiv$ *t2_Buys*.
- the products represented by *t1_Buys* could be a subset of products represented by *t2_Buys* hence *t1_Buys* $\subseteq$ *t2_Buys*.
- some of the products represented by both roles could be the same, hence *t1_Buys* $\cap$ *t2_Buys* $\neq \emptyset$.
- all the products represented by both roles could be different, hence *t1_Buys* $\cap$ *t2_Buys* $= \emptyset$.

In the example that will be presented in the subsection 6.3.1. the last situation will be considered.

- $A \cap B = \emptyset$

In general, we have defined a correspondence between roles of different concepts in terms of the correspondence that exists between those roles for the objects that belong to the intersection of the RWSs of the concepts. For this reason, it is pointless to study the cases of role integration when the concepts are disjoint. A special case occurs when the roles of disjoint concepts make reference to the same idea of the real world. For example, consider the following two concepts from the terminologies mentioned previously, *t1_employee* and *t2_employees* with no common instances. Their RWSs intersection would be the empty set but the PRI could wish to generalize them. The integration

of the roles corresponding to *t1_employee* and *t2_employee* will be associated with the new concept. For example, it may be useful to integrate the roles *E#,E_name,Salary* of the concepts *t1_employee* and *t2_employee*. We express this kind of role correspondences as $a_i \cap b_j = \emptyset$.

6.2.3. Types of correspondence between concepts and roles.

Let a_i be a role having the attributive type T as a range. This role is associated to the concept A in a terminology ($a_i :<$ domain(A) and range(T)). Let B be a concept of a different terminology to be integrated with the first one. The types of correspondence that can be defined between a_i and B are the same as those appearing in correspondences between concepts:

- *a_i is equivalent to B ($a_i \equiv B$)*, means that RWS(T $\rightarrow a_i$) = RWS(B) is always verified.
- *a_i is included in B ($a_i \subseteq B$)*, means that RWS(T $\rightarrow a_i$) $\subseteq$ RWS(B) is always verified.
- *B is included in a_i ($B \subseteq a_i$)*, means that RWS(B) $\subseteq$ RWS(T $\rightarrow a_i$) is always verified.
- *a_i overlaps with B ($a_i \cap B \neq \emptyset$)*, means that objects can exist in the RWS(T $\rightarrow a_i$) that are also in RWS(B), but at some moments RWS(T $\rightarrow a_i$) $\cap$ RWS(B) can be the empty set.
- *a_i is disjoint with B ($a_i \cap B = \emptyset$)*, means that the real world objects of a_i and the real world objects of B refer to the same kind of objects, but RWS(T $\rightarrow a_i$) $\cap$ RWS(B) = $\emptyset$ is always verified.

For example, for the concept *t1_department* and role *t2_Dept* with domain in concept *t2_supplier* from terminologies *t1* and *t2* respectively, the following situations could happen:

- They could represent the same set of departments, hence *t1_department* $\equiv$ *t2_Dept*.
- The departments represented by *t1_department* could be a subset of those represented by *t2_Dept*, hence *t1_department* $\subseteq$ *t2_Dept*.
- The departments represented by *t2_Dept* could be a subset of those represented by *t1_department*, hence *t2_Dept* $\subseteq$ *t1_department*.
- Some of the departments represented for both *t1_department* and *t2_Dept* could be the same, hence *t1_department* $\cap$ *t2_Dept* $\neq \emptyset$.
- They could represent disjoint sets of departments, hence *t1_department* $\cap$ *t2_Dept* = $\emptyset$.

In the example that will be presented in the subsection 6.3.4. the lastly situation will be considered.

6.3. Integration Rules

In the previous subsection, the different types of correspondences that can be defined have been shown. In this subsection, the rules that the system applies for those correspondences are explained. We introduce, in sequence, the integration of concepts, then the verification of correspondences, next the integration of roles, and lastly, the integration of concepts with roles. In order to illustrate these notions we will integrate the terminologies (see figure 6) created from the relational schemata of figure 2.

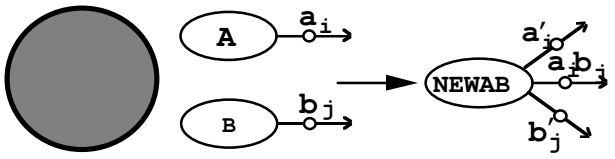
6.3.1. Integration of concepts

For each kind of integration rule we illustrate graphically the result of the integration and the type (primitive or defined) and description of the resulting concepts.

In the following, A and B denote concepts of two terminologies *T1* and *T2* respectively, to be integrated; $a_1 \ldots a_n$ refer to the roles associated with A and $b_1 \ldots b_m$ to the roles associated with B. Moreover, by a_i' are denoted the roles associated with A in *T1* that cannot be integrated with roles of B; by b_j' the roles associated with B in *T2* that cannot be integrated with roles of A; and by $a_i b_j$ the roles obtained by integrating a role of A with a role of B. Finally, C_D is used to denote the description of a concept C, $C_D = d_1$ and...and d_n.

Tables representing the descriptions of the integrated concepts contain references to the notions of *Common Specialization* (CS), *Most Specific Generalization* (MSG) and *Specific Properties* (SP) that are explained in the appendix.

1. A ≡ B (Equivalence)



RWS(A)=RWS(B)

A defined and B defined	A defined and B primitive	A primitive and B defined	A primitive and B primitive
NEWAB defined	NEWAB defined	NEWAB defined	NEWAB primitive

equivalent(A_D,B_D)	subsumes(A_D,B_D)	subsumes(B_D,A_D)	in other case
$NEWAB_D = A_D$ or $NEWAB_D = B_D$	$NEWAB_D = B_D$	$NEWAB_D = A_D$	$NEWAB_D = CS(A,B)$

Figure 8: Case A ≡ B

The result of the integration of A and B is a concept, arbitrarily called NEWAB, that represents the same RWS as that of A and B (see figure 8).

For example, integrating the two terminologies represented in figure 6, suppose that the PRI expresses the following correspondence: *t1_client* ≡ *t2_client*. Because the descriptions of those concepts are compatible with respect to the defined

correspondence (this aspect is analysed with more detail in the following subsection), the integration rule is applied and then the task of integrating their roles is started.

Assuming that the PRI expresses next, one by one, that the following roles of both concepts are equivalent –*code* of *t1_client* and *code* of *t2_client*, *name* of *t1_client* and *name* of *t2_client*, *address* of *t1_client* and *address* of *t2_client*, and last *payment* of *t1_client* and *payment* of *t2_client*–, the integrated concept will have roles[18] *code, name, address and payment*. The restrictions for these roles might come from *t1_client* or *t2_client*.

Moreover, the descriptions of the concepts *good*, *slowpayer*, *normal*, *regular* and *government* will be redefined, since they contain references to the roles *Payment*:

```
t1_good := client and all(Payment,le(30)) and atleast(1,Payment)
t1_slowpayer := client and all(Payment,ge(40)) and atleast(1,Payment)
t2_normal := client and all(Payment,le(30))
t2_government := client and all(Payment,ge(60)) and atleast(1,Payment)
t2_regular := client and all(Payment,ge(40)) and all(Payment,le(60)) and
atleast(1,Payment)
```

As a result of the redefinition process, the terminological system will automatically discover that *good* and *normal* are equivalent concepts, and that *slowpayer* subsumes *regular* and *government*. Notice here the advantage of working with a terminological system.

Later, the PRI wants to integrate the roles *Buys* of *t1_client* and *Buys* of *t2_client*. However, this is not possible at this moment, because it is first necessary to integrate the concepts *t1_product* and *t2_product* that are described as ranges of those roles . The integration of these two concepts becomes a pending task. In summary, the resultant concept after integrating the concepts *t1_client* and *t2_client* and all their roles is shown in figure 9.

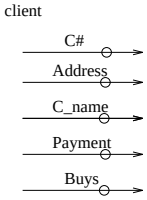


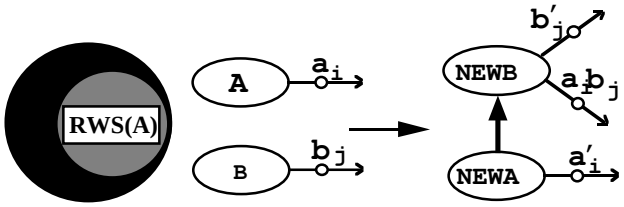
Figure 9: Result of the integration

2. A ⊆ B (Inclusion)

The result of the integration of A and B are two concepts, arbitrarily called NEWA and NEWB, that represent the same RWS as that of A and B, respectively.

In the example, suppose that instead of the previous correspondence defined between the *t1_client* and *t2_client* the PRI expresses the following one: *t2_client* ⊆ *t1_client*. In this case, assuming that the integration of their roles is the same as that presented in the previous case, the result of integrating these concepts is shown in figure 11.

[18] The method for integrating two related roles will be explained in section 6.3.3..



A defined and B defined	A defined and B primitive	A primitive and B defined	A primitive and B primitive
NEWB defined NEWA defined	NEWB primitive NEWA defined	NEWB defined NEWA primitive	NEWB primitive NEWA primitive

equivalent(A_D,B_D)	subsumes(A_D,B_D)	subsumes(B_D,A_D)	in other case
$NEWB_D = B_D$ $NEWA_D$ = NEWB	$NEWB_D = B_D$ $NEWA_D$ = NEWB	$NEWB_D = B_D$ $NEWA_D$ = SP(NEWB,A)	$NEWB_D = B_D$ $NEWA_D$ = CS(A,B)

Figure 10: Case A $\subseteq$ B

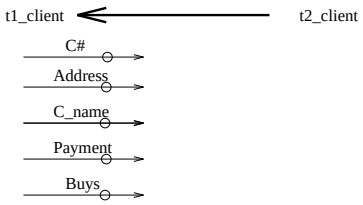


Figure 11: Result of the integration

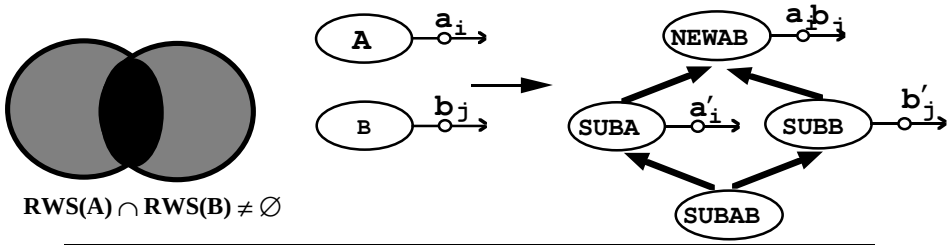
3. A $\cap$ B $\neq \emptyset$ (overlapping)

The result of the integration of A and B are four concepts, arbitrarily named NEWAB, SUBA, SUBB, and SUBAB. SUBA and SUBB represent the same RWS as that of A and B respectively. NEWAB represents the union of the RWSs of A and B, and SUBAB represents the intersection of the RWSs of A and B.

Supposing now that integrating the two terminologies of figure 6, the PRI expresses the correspondence $t1\_supplier \cap t2\_supplier \neq \emptyset$. Because the descriptions of those concepts are compatible with respect to the defined correspondence, the integration rule is applied and then the task of integrating their roles is started. The PRI also expresses that the roles corresponding to the codes, names, and addresses of both concepts are equivalent. The result of the integration is shown in figure 13.

4. A $\cap$ B = $\emptyset$ (disjoint)

The result of the integration of A and B are two concepts, arbitrarily called SUBA and SUBB that represent the same RWS that of A and B, respectively,



A defined and B defined	A defined and B primitive	A primitive and B defined	A primitive and B primitive
NEWAB primitive	NEWAB primitive	NEWAB primitive	NEWAB primitive
SUBA defined	SUBA defined	SUBA primitive	SUBA primitive
SUBB defined	SUBB primitive	SUBB defined	SUBB primitive
SUBAB defined	SUBAB defined	SUBAB defined	SUBAB defined

equivalent(A_D,B_D)	subsumes(A_D,B_D)	subsumes(B_D,A_D)	in other case
$NEWAB_D$ = MSG(A,B)	$NEWAB_D$ = A_D	$NEWAB_D$ = B_D	$NEWAB_D$ = MSG(A,B)
$SUBA_D$ = NEWAB	$SUBA_D$ = NEWAB	$SUBA_D$ = SP(NEWAB,A)	$SUBA_D$ = SP(NEWAB,A)
$SUBB_D$ = NEWAB	$SUBB_D$ = SP(NEWAB,B)	$SUBB_D$ = NEWAB	$SUBB_D$ = SP(NEWAB,B)
	$SUBAB_D$ = SUBA and SUBB		

Figure 12: Case A ∩ B ≠ ∅

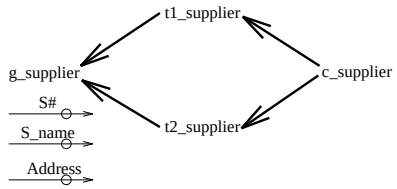
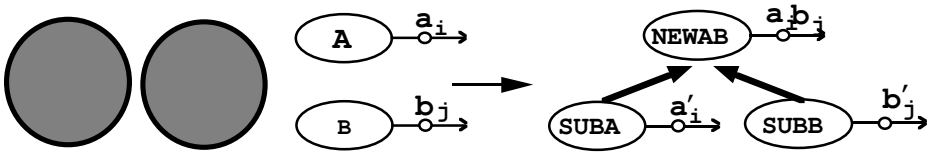


Figure 13: Result of the integration

and another concept, NEWAB that represents the union of the RWSs of A and B. At the same time, SUBA and SUBB are declared as disjoint in the terminology. Working with terminological systems, two different concepts can have common instances unless they are explicitly declared as disjoint.

It is worthwhile to note that A and B must refer to the same kind of objects, otherwise these correspondences could be defined between almost all pairs of concepts. For example, it would be worthwhile to say that concepts *client* and *supplier* are disjoint, taking into account that they are persons or organizations related to the company. But probably, in the same context, it would not be worthwhile to say that concepts *client* and *product* are disjoint.

Continuing the integration process of the input terminologies, the PRI defines that $t1\_employee \cap t2\_employee = \emptyset$. The descriptions of the concepts are compatible with respect to the defined correspondence, and so the integration of the concepts and the corresponding roles takes place. The PRI expresses



RWS(A) ∩ RWS(B) = ∅

A defined and B defined	**A defined and B primitive**	**A primitive and B defined**	**A primitive and B primitive**
NEWAB primitive	NEWAB primitive	NEWAB primitive	NEWAB primitive
SUBA defined	SUBA defined	SUBA primitive	SUBA primitive
SUBB defined	SUBB primitive	SUBB defined	SUBB primitive

equivalent(A_D,B_D)	**subsumes(A_D,B_D)**	**subsumes(B_D,A_D)**	**in other case**
$NEWAB_D$ = MSG(A,B)	$NEWAB_D$ = A_D	$NEWAB_D$ = B_D	$NEWAB_D$ = MSG(A,B)
$SUBA_D$ = NEWAB	$SUBA_D$ = NEWAB	$SUBA_D$ = SP(NEWAB,A)	$SUBA_D$ = SP(NEWAB,A)
$SUBB_D$ = NEWAB	$SUBB_D$ = SP(NEWAB,B)	$SUBB_D$ = NEWAB	$SUBB_D$ = SP(NEWAB,B)

Figure 14: Case A ∩ B = ∅

that the roles corresponding to the codes, names and salaries of both concepts are equivalent. The result of the integration is shown in figure 15.

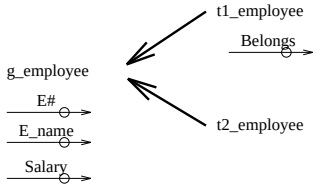


Figure 15: Result of the integration

6.3.2. Verification of Correspondences

The use of a terminological system to do the integration process allows us to use the descriptions associated with the concepts to discover possible errors in the definition of correspondences between data elements to be integrated. In general, the descriptions of the data elements involved in a correspondence can be related or not. If they are related, three different situations are possible: both are equivalent, one subsumes the other, or they are disjoint. Depending on both the situation and the kind of correspondence (see figure 16), the system notifies the PRI with one of the following messages:

– *Error*, the descriptions of the concepts and the correspondence definition are disjoint. For example, if concepts *t1_good* and *t2_government* are said to be equivalent by the PRI, then an error is detected because their corresponding definitions: *client and all(Payment,le(30)) and*

atleast(1,Payment) for *t1_good*, and *client and all(Payment,ge(60)) and atleast(1,Payment)* for *t2_government* are incompatible.

- *Warning*, the descriptions and the correspondence definition can be compatible, but the system believes that there could exist a more precise correspondence. For example, if concepts *t1_slowpayer* and *t2_regular* are said to be equivalent by the PRI, then a warning is detected because the definition of *t1_slowpayer* (*client and all(Payment,ge(40)) and atleast(1,Payment)*) subsumes the definition of *t2_regular* (*client and all(Payment,ge(40)) and all(Payment,le(60)) and atleast(1,Payment)*) and therefore it seems that *t2_regular* is included in *t1_slowpayer*.
- *Ignored*, the definition of the correspondence does not make any new contribution. For example, if concepts *t1_slowpayer* and *t2_regular* are said to be overlapping by the PRI, then it is ignored.

	equivalent (A_D,B_D)	**subsumes** (A_D,B_D)	**subsumes** (B_D,A_D)	**subsumes** (**nothing**,$A_D \wedge B_D$)
A ≡ B	—	warnig	warning	error
A ⊆ B				
A defined	error	error	—	error
B primitive	warning	warning	—	error
A ∩ B ≠ ∅				
A or B are defined	ignored	ignored	ignored	error
other case	—	—	—	error
A ∩ B = ∅				
A or B are defined	error	error	error	—

Figure 16: Cases in which the system can detect anomalies

6.3.3. Integration of roles

This subsection illustrates the method for integrating two related roles (in accordance with the types of correspondence defined in 6.2.2.) and the properties that must be fulfilled for them to be integrated.

Let a_i be a role defined for a concept A of *T1*, and b_j be a role defined for a concept B of *T2* and let their definition be:

(i) a_i :< domain (A) and range (r_i)

(ii) b_j :< domain (B) and range (r_j)

As stated before, for two roles a_i and b_j of different terminologies to be related, their corresponding concepts must also be related.

The resulting role of integrating a_i and b_j, represented as $a_i b_j$, will have the following definition: $a_i b_j$:< domain (C) and range (r). The concept C is the most general concept obtained by integrating A and B, so for the cases A ≡ B, A ∩ B ≠ ∅ and A ∩ B = ∅, C is the concept called as NEWAB in the corresponding integration rules. For the case A ⊆ B, C corresponds to the concept called NEWB. Moreover, the range *r* is obtained by integrating

r_i and r_j, r will depend on the types of r_i and r_j. We can find the following situations:

- If r_i and r_j are attributive types and there exists a function that relates each element of r_i to its equivalent of r_j, then r will be the union of r_i and r_j.
- If r_i is a concept and r_j is an attributive type (or vice versa) and there exists a function that relates each element of r_j to its equivalent instance of r_i, then r will be the result of the integration of the concept r_i and the role b_j.
- If r_i and r_j are both concepts that can be integrated, then r will be the most specific concept of the integrated terminology such as RWS(r_i) $\subseteq$ RWS(r) and RWS(r_j) $\subseteq$ RWS(r).
- Otherwise, a_i and b_j cannot be integrated.

In figures 17 and 18 we show the range r and the RWS of the resulting role $a_i b_j$, respectively, depending both on the correspondence between the roles a_i and b_j and the concepts A and B.

	$\mathbf{a}_i \equiv \mathbf{b}_j$	$\mathbf{a}_i \subseteq \mathbf{b}_j$	$\mathbf{a}_i \cap \mathbf{b}_j \neq \emptyset$	$\mathbf{a}_i \cap \mathbf{b}_j = \emptyset$
$\mathbf{A} \equiv \mathbf{B}$	r_i or r_j	r_j	integ. of r_i and r_j	integ. of r_i and r_j
$\mathbf{A} \subseteq \mathbf{B}$	r_j	r_j	integ. of r_i and r_j	integ. of r_i and r_j
$\mathbf{B} \subseteq \mathbf{A}$	r_i	integ. of r_i and r_j	integ. of r_i and r_j	integ. of r_i and r_j
$\mathbf{A} \cap \mathbf{B} \neq \emptyset$	integ. of r_i and r_j	integ. of r_i and r_j	integ. of r_i and r_j	integ. of r_i and r_j
$\mathbf{A} \cap \mathbf{B} = \emptyset$	—	—	—	integ. of r_i and r_j

Figure 17: Range of the new roles

	$\mathbf{a}_i \equiv \mathbf{b}_j$	$\mathbf{a}_i \subseteq \mathbf{b}_j$	$\mathbf{a}_i \cap \mathbf{b}_j \neq \emptyset$	$\mathbf{a}_i \cap \mathbf{b}_j = \emptyset$
$\mathbf{A} \equiv \mathbf{B}$	RWS(a_i) or RWS(b_j)	RWS(b_j)	RWS(a_i) $\cup$ RWS(b_j)	RWS(a_i) $\cup$ RWS(b_j)
$\mathbf{A} \subseteq \mathbf{B}$	RWS(b_j)	RWS(b_j)	RWS(a_i) $\cup$ RWS(b_j)	RWS(a_i) $\cup$ RWS(b_j)
$\mathbf{B} \subseteq \mathbf{A}$	RWS(a_i)	RWS(a_i) $\cup$ RWS(b_j)	RWS(a_i) $\cup$ RWS(b_j)	RWS(a_i) $\cup$ RWS(b_j)
$\mathbf{A} \cap \mathbf{B} \neq \emptyset$	RWS(a_i) $\cup$ RWS(b_j)	RWS(a_i) $\cup$ RWS(b_j)	RWS(a_i) $\cup$ RWS(b_j)	RWS(a_i) $\cup$ RWS(b_j)
$\mathbf{A} \cap \mathbf{B} = \emptyset$	—	—	—	RWS(a_i) $\cup$ RWS(b_j)

Figure 18: RWS of the new roles

Notice that in all the previous cases of role integration, we have not taken advantage of the role specialization mechanism provided by terminological systems, and therefore only one of the two roles remains in the resulting terminology. The PRI could use that facility by answering affirmatively to the system request about maintaining both roles in some cases. For example, suppose that when integrating two terminologies, *ta* and *tb*, there are two concepts *ta_project* and *tb_project* that the PRI has related by telling the system that *ta_project* $\equiv$ *tb_project*. Moreover, *ta_project* has associated with it a role *Leader*, while *tb_project* has a role *Staff*, where *Leader* is described as *domain(ta_project) and range(doctor)*. The PRI can stay that *Leader* $\subseteq$ *Staff*

and that he wants both roles, *Leader* and *Staff*, to remain in the resulting terminology. Then, as a result of the application of this correspondence, the integrator will redefine *Leader* as *Leader* :< *Staff and range(doctor)*, so the role *Leader* is a specialization of the role *Staff*.

6.3.4. Integration of concepts and roles

Consider a role a_i and a concept B of two different terminologies:

(i) $\mathbf{a_i \equiv B}$ or $\mathbf{a_i \subseteq B}$

- The result of the integration of a_i and B, is a change in the range of the role a_i. The concept B will be the new range of the role a_i.

(ii) $\mathbf{B \subseteq a_i}$ **or** $\mathbf{B \cap a_i \neq \emptyset}$ **or** $\mathbf{B \cap a_i = \emptyset}$

- The result of the integration of a_i and B is a new concept, arbitrarily called NEWA, that verifies RWS(NEWA)= (RWS(T $\rightarrow$ a_i) and a change in the range of the role a_i. The concept NEWA will be the new range of the role a_i. After this, the integration of NEWA and B will behave as the integration of two concepts.

In the example, suppose that the PRI expresses that the role *t2_Dept* of *t2_supplier* and the concept *t1_department* are disjoint (*t2_Dept* $\cap$ *t1_department* = $\emptyset$). First, a new concept *t2_department* is added and the range of the role *t2_Dept* is changed to be *t2_department*. Then, the appropriate integration rule for the correspondence *t1_department* $\cap$ *t2_department* = $\emptyset$ is applied. In figure 19, the resulting concepts are shown.

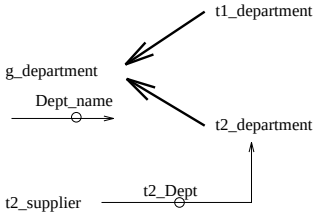
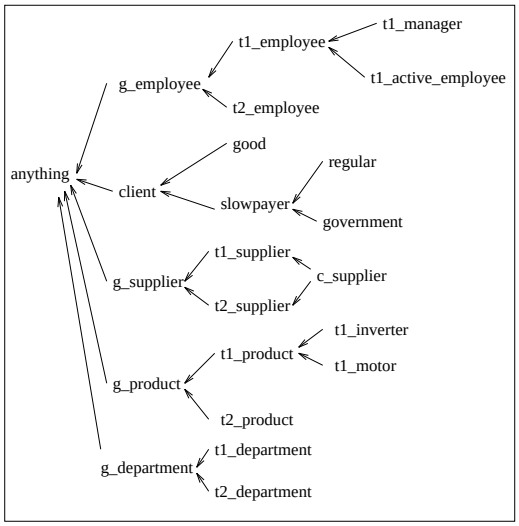


Figure 19: Result of the integration

In figure 20, the federated terminology that has been obtained throughout this section is shown.

6.4. Manual approach

For the situations in which the PRI is interested in obtaining a particular federated terminology that cannot be generated using the semi-automatic approach, he can use the same set of high level operations provided to do the



good := client and all(Payment,le(30)) and atleast(1,Payment)
slowpayer := client and all(Payment,ge(40)) and atleast(1,Payment)
government := client and all(Payment,ge(60)) and atleast(1,Payment)
regular := client and all(Payment,ge(40)) and all(Payment,le(60)) and atleast(1,Payment)
t1_motor := t1_product and all(Type,motortype)
t1_inverter := t1_product and all(Type,invertertype)
t1_active_employee := t1_employee and atleast(2,t1_Participates)
c_supplier := t1_supplier and t2_supplier

Figure 20: The federated terminology obtained by using the semi-automatic approach

manual translation process, increased with two new ones. These new operations are EQUIVALENT-CONCEPT and EQUIVALENT-ROLE. The operation EQUIVALENT-CONCEPT(concept-name1,concept-name2), can be used to remove a redundant concept. All the descriptions that use the deleted concept (concept-name2) will be redefined in terms of the other (concept-name1). The behaviour is similar for the EQUIVALENT-ROLE operation but in this case for removing a role.

The operations basically permit the PRI to obtain a terminology combining elements of the two input terminologies. He can start with the default federated terminology offered by the system or from the terminology obtained by using the semi-automatic module.

For example, starting with the federated terminology obtained as a result in the semi-automatic integration (see figure 20), the PRI could introduce the operations:

```
SPECIALIZE-CONCEPT(motor, g_product and
all(Type,motortype))
```

to add a new defined concept corresponding to a subset of products.

Apart from refining a given terminology, the manual module can also be used to relate concepts of one terminology to concepts of another terminology (both terminologies participating in the integration process) using intensional descriptions. It is not possible to define this type of correspondence by using directly the RWS notion utilized in most automatic approaches [8, 3].

For example, suppose that the common instances of *t1_supplier* and *t2_supplier* verify that they belong to some specific departments, then the concept *c_supplier* can be redescribed in the following way:

```
REDESCRIBE-CONCEPT(c_supplier, t1_supplier and
t2_supplier and all(t2_Dept,spec_depts)
```

Notice that using the RWS notion the PRI could only say that *t1_supplier* and *t2_supplier* overlap ($t1\_supplier \cap t2\_supplier \neq \emptyset$).

In figure 21, the final federated terminology obtained from the terminology in figure 20 by using the manual approach is shown.

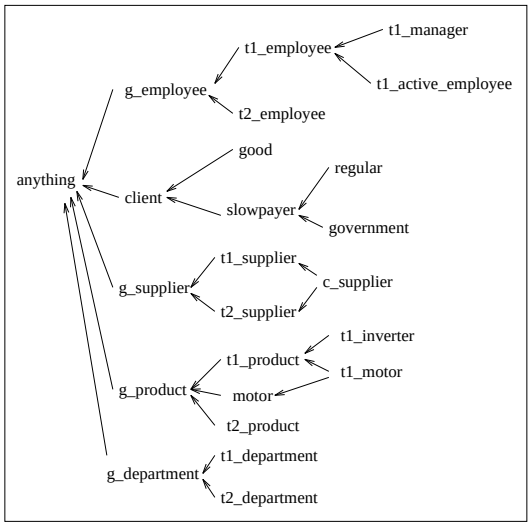


Figure 21: The resultant federated terminology

7. Mapping Information

As we have mentioned before, our general goal is to integrate several relational databases using a terminological system. This means that we will have as a result a federated terminology, which is related to the underlying database

contents. The elements of the terminology will be concepts and roles, and therefore a mapping will be needed that relates these elements to the elements of the relational schemata, namely, relations and attributes. This mapping information will be managed both by the translator and the integrator components.

The goal of the mapping information is to link the terminological and relational data elements in such a way that, given a link 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.

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 *CLIENT* of the schema on the left side of figure 2. The link of this relation with the concept *client*, means that for each tuple in the relation *CLIENT* there is one instance of the concept *client*.

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 interested in defining the concept *slowpayer* that has been defined in such a way that there is an instance of this concept for each tuple of the relation *CLIENT* that satisfies the condition *Payment >40*. 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[19].

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, the link of the relation *EMPLOYEE* and the attribute *Dept* of *EMPLOYEE* with the concept *department*, means that there exists an instance of *department* for each different value that the attribute *Dept* takes.

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 *EMPLOYEE* implies the existence of an instance of the concept *employee* and another one of the concept *department*.

Finally, several relations may determine the instances of a concept. For example, the concept *g-employee* is linked with the relations named *EMPLOYEE* from the two schemata considered, and there will be an instance of *g-employee* for each tuple of each relation. In this case, a link will be established between a concept and a set of relations.

So far we have presented the link types for a concept; we now concentrate on the role values. 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

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

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 *Salary* of *employee* can be linked to the attribute *Salary* of *EMPLOYEE*.
- 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 *D_name* associated with *department* can be linked to the attribute *Dept* of *EMPLOYEE*.

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 *Participates* associated with *employee* will be linked with the attribute *PROJECT.Name* of the derived relation $EMPLOYEE \bowtie_{E\#} EMP\_PROJECT \bowtie_{Pj\#} PROJECT$. In this way the link between instances of *employee* and the value of the role *Participates* is established through the join of three relations.

Here, we introduce briefly the representation used for the mapping information of concepts and roles. Its formal definition is presented in [33].

The mapping information associated with the concepts is represented as a list of n-tuples that have the following format:

$<\mathrm{R},(\mathrm{atr}_1,\ldots,\mathrm{atr}_n),\mathrm{T}>$

where R is either a basic or a derived relation; $atr_1,\ldots,atr_n$ are attributes of R that permit the identification of objects of the real world and T is the type of the attributes.

Moreover, the representation used for the mapping information of roles is represented as a list of n-tuples that have the following format :

$<\mathrm{R}_{rl},(\mathrm{atc}_1,\ldots,\mathrm{atc}_n),\ (\mathrm{atrl}_1,\ldots,atrl_m),\mathrm{T}_C,\mathrm{f}_{rl},\mathrm{T}_{rl}>$

where R_{rl} is either a basic or a derived relation; $atc_1,\ldots,atc_n$ are attributes of R_{rl} that permit the identification of objects of the real world; $atrl_1,\ldots,atrl_m$ are attributes of R_{rl} that define the role values (or contain the role values) corresponding to those objects of the real world; T_C is the type of $\mathrm{atc}_1,\ldots,\mathrm{atc}_n$; f_{rl} is a function defined as $\mathrm{f}_{rl}:\mathrm{D}_1 \times \ldots \times D_m \rightarrow \mathrm{T}_{rl}$, where D_i is the domain of attribute atrl_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).

Figure 22 shows the mapping information, by using this representation, for some concepts and roles presented throughout this paper.

concept	**(<R,(atr$_1$,...,atr$_n$),T>, ...)**
employee	(<EMPLOYEE,(E#),integer>)
active_employee	(<$\sigma_{count>2}$(E# $\mathcal{F}$ count(Pj_name)(EMPLOYEE $\bowtie_{E\#}$ EMP_PROJECT $\bowtie_{Pj\#}$ PROJECT),(E#),integer>)
client	(<CLIENT,(C#),integer>)
slowpayer	(<$\sigma_{Payment>40}$(CLIENT),(C#),integer>)
department	(<EMPLOYEE,(Dept),string>)
external	(<CLIENT,(E#),integer>,<SUPPLIER,(S#),integer>)
role	**(<R$_{rl}$,(atc$_1$,...,atc$_n$), (atrl$_1$,...,$atrl_m$),T$_C$,f$_{rl}$,T$_{rl}$>,...)**
Salary	(<EMPLOYEE,(E#),(Salary),integer,transform_to_ECU,number>)
D_name	(<EMPLOYEE,(Dept),(Dept),character,empty,string>)
Participates	(<EMPLOYEE $\bowtie_{E\#}$ EMP_PROJECT $\bowtie_{Pj\#}$ PROJECT,(E#),(Pj_name),integer,empty,string>)

Figure 22: Example of Mapping Information

8. The advantages of using a terminological system for integrating relational schemata

In general, we see the use of a terminological system to create the federated terminology as an advance on the utilization of a semantic or an object-oriented model for this purpose [34]. The main advantages can be summarized as follows:

From the federated schema generation process point of view

- the specification of concepts using intensional descriptions permits richer translation and integration processes.
- the use of the classification mechanism permits inferring automatically new correspondences among concepts, not explicitly defined by the PRI, both in the translation and integration processes.
- the integration process is more consistent, because the descriptions associated with the concepts can be used to discover errors in the definition of correspondences between data elements expressed by the PRI in order to integrate them.

From the resultant federated schema point of view, it allows for the definition of a semantically richer federated schema, due to the specification of concepts using intensional descriptions expressed in terms of the necessary or necessary and sufficient properties that must be satisfied by their instances. Moreover, those concepts can be used as intensional descriptions of answers, thus providing an alternative to extensional listing of values and therefore eliminating sometimes the cost of accessing to the underlying databases.

Nevertheless, there also exists a limitation. The use of a terminological system does not provide any particular advantage if the schemata that must be integrated are semantically poor and do not allow *defined* concepts to be obtained. But this limitation is mainly related to the semantics of the underlying databases and so other approaches will find similar difficulties.

9. Conclusions

The integration of various information sources is an interesting research topic in which great effort is being expended. We have explained in this paper two components of a federated system that permits the integration of several relational databases using a terminological system, the translator and the integrator.

The goal of the translator is to obtain a terminology from a relational schema with the PRI's help. The main features of this component are: 1) allowing the PRI to select between guiding the translation process or leaving the responsibility for it to the system, 2) the wide range of property types that it takes into consideration in order to perform the translation process semi-automatically, and 3) the opportunity that it gives to the PRI to generate a rich terminology by using all the expressive power of a terminological system.

The integrator allows the integration of different relational databases schemata, previously transformed into terminologies. The main features provided by this component are: 1) it allows the integration of semantically rich schemata with conceptual hierarchies and combination of structural information with the semantics of the Universe of Discourse, 2) it automates as much as possible the integration process by obtaining candidate correspondences between terminological components and also takes advantage of the classification mechanism provided by the terminological system, 3) it permits the integration to advance progressively by working interactively with the system.

Furthermore, both components provide the PRI with support in generating the mapping information that relates the resultant terminology data elements to the underlying databases, relegating the main responsibility of it to the system.

Finally, the two components presented in this paper have been implemented using the BACK system and PROLOG. Both components provide a user friendly interface.

Acknowledgements

We like to thank A. Borgida, J. Patrick, O. Díaz, and the anonymous referees for their valuable comments.

This work was partially supported by the Commission of the European Communities (Esprit project 5210 AIMS) and by the Plan Nacional de Investigación Científica y Desarrollo Tecnológico (TIC91-1246-CE).

References

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

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

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

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

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

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

[7] 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.

[8] 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.

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

[10] A.P. Sheth, S.K. Gala, and S.B. Navathe. On automatic reasoning for schema integration. *International Journal of Intelligent and Cooperative Information Systems*, 2(1):23–50, 1993.

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

[12] S. Bergamaschi and C. Sartori. On Taxonomic Reasoning in Conceptual Design. *ACM Transactions on Database Systems*, 17(3):385–421, 1992.

[13] Y. Arens, C.Y. Chee, C. Hsu, and C.A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal on Intelligent and Cooperative Information Systems*, 2(2):127–158, 1993.

[14] R.J. Brachman, P.G.. Selfridge, L.G. Terveen, B. Altman, A. Borgida, F. Halpner, T. Kirk, A. Lazar, D.L. McGuinnes, and L.A. Resnick. Integrated support for data archaelogy. *International Journal on Intelligent and Cooperative Information Systems*, 2(2):159–185, 1993.

[15] A. Savasere. *An approach to integration using classification.* PhD thesis, University of Florida, Gainesville, 1990.

[16] J. M. Blanco, A. Illarramendi, J. M. Pérez, and A. Goñi. Making a federated database system active. In *Database and Expert Systems Applications*. Springer-Verlag, 1992.

[17] A. Borgida, R.J. Brachman, D.L. McGuinness, and L.A. Resnick. CLASSIC: A structural data model for objects. In *Proceedings ACM SIGMOD-89, Portland, Oregon*, 1989.

[18] A. Borgida. From type systems to knowledge representations: Natural semantics specifications for description logics. *Intelligent and Co-operative Information Systems*, 1(1), 1992.

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

[20] K.H. Davis and A.K. Arora. Converting a relational database model into an entity-relationship model. In *Entity-Relationship approach (Proceedings 6th ER Conference, New York, 1987).* North Holland, 1988.

[21] V. M. Markowitz and J. A. Makowsky. Identifying extended entity-relationship object structures in relational schemas. *IEEE Transactions on Software Engineering*, 16(8), 1990.

[22] M. Castellanos and F. Saltor. Semantic enrichment of database schemas: An object oriented approach. In *First International Workshop on Interoperability in Multidatabase Systems*, 1991.

[23] S. McKearney, D.A. Bell, and R. Hickey. Inferring abstract objects in a database. In *[36]*, 1992.

[24] L-L. Yan and T-W. Ling. Translating relational schema with constraints into OODB schemas. In *Proc. of the International Conference Semantics of Interoperable Database Systems. Lorne, Australia.*, November 1992.

[25] D.A. Keim, H.P. Kriegel, and A. Miethsam. Integration of relational databases in a multidatabase system based on schema enrichment. In *[35]*, pages 96–104, 1993.

[26] J. Biskup and B. Convent. A formal view integration method. In *ACM SIGMOD International Conference on Management of Data, Washington*, 1986.

[27] P. Sheth and J. A. Larson. Federated database systems for managing distributed heterogeneous and autonomous databases. *ACM Computing Surveys*, 22(3), September 1990.

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

[29] S. Navathe and A. Wong. Abstracting relational and hierarchical data with semantic data model, entity-relationship approach. In *Proc. 6th. ER Conf.*, 1987.

[30] C. Zaniolo. Design of relational views over network schemas. In *Proceedings of the ACM SIGMOD Conference*, 1979.

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

[32] M.A. Qutaishat, N.J. Fiddian, and W.A. Gray. Association merging in a schema meta-integration system for a heterogeneous object-oriented database environment. In *Lecture Notes in Computer Science Proc. 10th British National Conference on Databases*, 1992.

[33] J.M. Blanco. *Integración de bases de datos relacionales por medio de un sistema terminológico: una propuesta utilizando BACK.* PhD thesis, Basque Country University, April 1994.

[34] J.M. Blanco, A. Illarramendi, A. Goñi, and J. Bermúdez. Advantages of using a terminological system for integrating databases. In *Proc. of the International Workshop on Description Logics. Bonn. Germany*, 1994.

[35] H.J. Scheck, A.P. Sheth, and B.D. Czejdo, editors. *Third International Workshop on Research Issues in Data Engineering: Interoperability in Multidatabase Systems, Vienna, Austria*, 1993.

[36] Y. Yesha, editor. *Proceedings of the ISMM International Conference on Information and Knowledge Management CIKM-92, Baltimore, MD, U.S.A., November*, 1992.

Appendix

- Let A be a concept and let A_D be its description, $A_D = d_1$ and...and d_n. The *fully concept description* of A, denoted by A_D^f, is a longer equivalent description giving all superconcepts plus all role restrictions which apply for the concept A, $A_D^f = p_1$ and...and p_m.
- The *Most Specific Generalization (MSG)* of two concepts A and B with A_D^f=p_{11} and ... and p_{1r} and $B_D^f = p_{21}$ and...and p_{2s} respectively, is a description
  G = p_1 and...and p_p that satisfies:
  (i) for all p_j in G, p_j must be in A_D^f or in B_D^f
  (ii) G subsumes A_D and subsumes B_D.
  (iii) G is not redundant, that is, for all p_j in G, G does not subsume G without p_j.
  (iv) G is the most specific description composed by properties of A_D^f and B_D^f, such that any G' obtained by adding a new property to G does not satisfy the above conditions.
- The *Common Specialization (CS)* of two concepts A and B with $A_D^f = p_{11}$ and...and p_{1r} and $B_D^f = p_{21}$ and...and p_{2s} respectively, is a description S = p_1 and...and p_p that satisfies:
  (i) for all p_j in S, p_j must be in A_D^f or in B_D^f
  (ii) A_D subsumes S and B_D subsumes S.
  (iii) S is not redundant.
- The *Specific Properties (SP)* of two concepts A and B is a description S = A and p_1 and...and p_p where for all p_j:
  (i) p_j must be in B_D^f
  (ii) p_j does not subsume A_D

Example. Let *person*, *man*, *woman* be concepts of a terminology. The primitive concept *man* is described as *person and atmost(20,child) and all(age,le(120))*, which means that all instances of *man* will be persons younger than 120 and with less than 21 children and *woman* is a primitive concept described as *person and atmost(10,child) and all(age,le(140))*.

MSG(man,woman)={person,atmost(20,child),all(age,le(140))}
CS(man,woman)={person,atmost(10,child),all(age,le(120))}
SP(man,woman)={man,atmost(10,child)}