Automatic Classification of Semantic Concepts in View Specifications

Ernst Ellmer[1], Christian Huemer[1], Dieter Merkl[2], Günther Pernul[3]

[1] University of Vienna, Institute of Applied Computer Science
Liebiggasse 4, A-1010 Wien, Austria
[2] Vienna University of Technology, Institute of Software Technology
Resselgasse 3, A-1040 Wien, Austria
[3] University of Essen, FB 5 - Information Systems
Altendorfer Str. 97, D-45143 Essen, Germany
<lastname>@ifs.univie.ac.at

Abstract: The design of large database systems often is done by a large number of analysts with different perspectives on the problem domain. That is why the integration of multiple views in database design is a task of tremendous importance. In this paper we report on our experience in using a view comparison tool based on neural network technology. The tool automatically extracts semantic concepts from different view specifications and then transforms them into a vector representation understandable by a neural network. The network is trained and thus performs the job of clustering similar concepts. The output of our tool is a 'first guess' which concepts in views may overlap or which concepts do not overlap at all. The two main contributions of our tool thus are first, that the designer is relieved from the burden of manually comparing each semantic concept of the different view specifications, and second, that human interference into the view comparison process is minimized to the specification of views and the interpretation of concept clusters.

1. Introduction

The development of view integration techniques has now been a recognized research area for some time and significant work has already been done to address involved issues. The work presented in this paper is not yet another general integration methodology, rather, it reports on our experiences in using a specific software tool which was developed to support some sort of database view integration, namely view comparison. The tool assumes as input a set of definitions of potentially overlapping views and classifies their concepts into several categories by using an artificial neural network. From the viewpoint of the tool all concepts grouped into a single category may be overlapping while concepts grouped into different categories are disjoint. The major activity to be performed during view integration is to find corresponding concepts in different views (i. e. concepts that refer to the same real world semantics) and to integrate them into a single representation. The fundamental problem here is the fact that one event of reality may be represented in different views differently. This simply because the same phenomenon may be seen under different aspects, may be modelled by using different levels of abstraction or may be described by using different properties. Our work is based on the following assumption: We assume, that concepts in dif-

ferent views that refer to the same real world semantics will have similar features, such as their names, attributes, types, links, or occur in similar contexts. We assume, that similarities may exist, yet we do not pretend to know them. We make use of this assumption and train a neural network to recognize common patterns in different views and to deliver a 'first guess' to the analyst which concepts may be overlapping with other concepts in other views and which concepts do not overlap at all. For the comparison we use artificial neural networks because of two reasons: First, neural networks have proven capabilities of being robust in the sense of tolerating 'noisy' input data (overlapping classes represented by similar yet not equal features) and as a second reason, we refer to their ability of generalization. The proposed methodology for view comparison consists of several interrelated phases which will be explained in the following subsection.

1.1 A process model for concept classification

We propose a structure for a schema comparison process consisting of four phases. Figure 1 shows the process model. The overall objective is to minimize human interference during the integration.

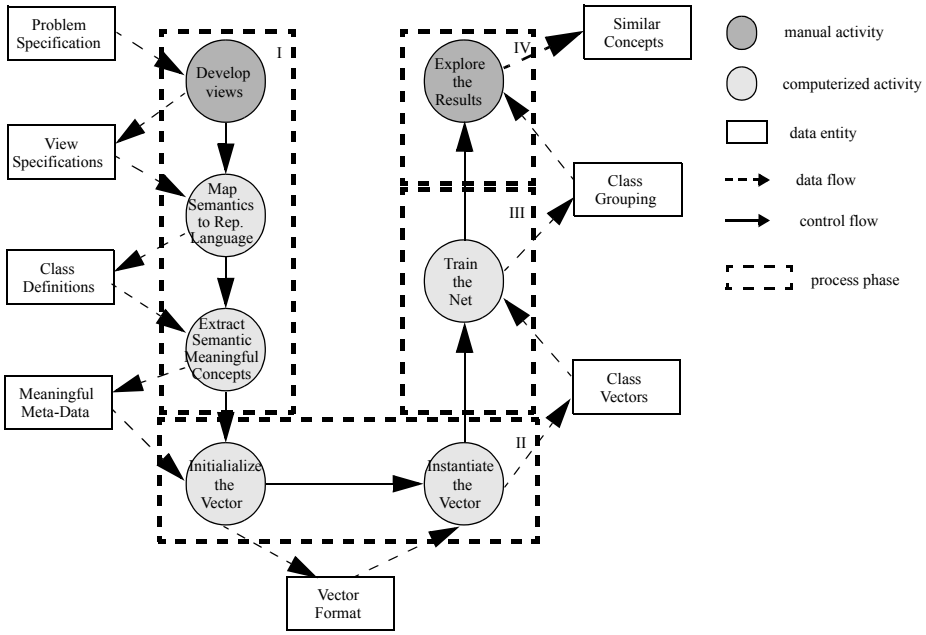

Fig. 1: A process model for concept classification

In the first phase, *schema parsing*, semantic meaningful concepts for view integration are extracted. In a first step, database designers develop views. Their semantics have to be mapped onto a representation language enabling automated processing by our tool. In a next step, data are restricted to concepts useful for view integration. We call this semantic meaningful data classifiers. These classifiers represent the input for phase

two, *vector initialization and instantiation*. The aim of this phase is to transform semantic meaningful meta data into a representation understandable by a neural net. This phase comprises two activities, namely the generation of a vector format as well as the instantiation of a set of vectors representing the input data of the view comparison process in a form processable by the neural net. The third phase, *train the net*, is determined by the neural network technology used and produces a grouping of database view concepts. This grouping has to be explored by humans in order to find similar concepts. *Explore results* thus represents the last phase of our view integration process model.

Our paper is outlined as follows. Section two gives a short introduction into the fundamental concepts of Kohonen nets, the neural network technology we used for our experiments. Section three explains the structure of the case study we carried-out. Section four then shows and discusses the results of this case study. Section five gives hints on related work while section six concludes the paper.

2. Self-Organizing Feature Maps - A Brief Introduction

Kohonen's self-organizing feature map [14] is one of the most prominent unsupervised learning methods in the area of artificial neural networks. The self-organizing feature map consists of a layer of input units each of which is connected with a grid of output units. These output units are arranged in some topological order which is application-dependent. Input units take the input in terms of a data vector and propagate the input to the output units. Each of the output units is assigned a weight vector with the same dimensionality as the input data.The learning algorithm of self-organizing maps can be regarded as an extension to competitive learning. Pragmatically speaking, the key idea of competitive learning is to adapt the unit with the highest activity level with respect to a randomly selected input, i.e. the winning unit, in a way to exhibit even higher activation with this very input in future. Adaptation occurs during each of the learning steps. As an addition to competitive learning the units in the neighborhood of the winning neuron are also adapted to exhibit higher activation with respect to the given input in self-organizing feature maps. This learning rule leads to a clustering of closely related items, i.e. input data with a highly similar vector representation, in the input domain. In general, self-organizing feature maps determine the activity level of the output units by computing the Euclidean distance between the currently presented input and the weight vectors of the various units. Hence, the winning unit is those with the smallest distance between the vectors. This unit and its topological neighbors is adapted in the course of the learning process. The adaptation as such is performed by gradually reducing the difference between the respective components of the unit's weight vector and the current input vector leading to a topographic arrangement of the input data. As a result, similar input data are mapped onto neighboring parts of the output grid. For a more detailled exposition we refer to [14].

3. A view comparison case study

In order to evaluate our approach to view comparison, we carried out a case study.

Usually, schema integration is necessary if a number of database designers model partly overlapping areas of a large real world domain containing hundreds of classes and relationships. The problem we faced was to find an example which first is complex and large enough to show the usefulness and applicability of our approach to view comparison and second is small enough to be presented in this paper. We feel that the central point of such an example problem is not the number of classes and relationships that are necessary to model the problem, but rather to find an example allowing a number of designers to model overlapping as well as very different areas of the same real world problem domain. This is necessary to show that our approach is able to find concepts representing similar areas of the problem domain on the one hand and concepts representing different areas on the other hand. We decided to use an example from [7] as a core problem specification representing the overlapping area to be modeled by each analyst and to enforce the modelers to extend the core problem by at least three concepts of their choice representing the areas which do not overlap. We passed the problem specification to a number of experts and undergraduate students and used their results to evaluate our approach. [7]Figure 2 shows the core problem specification from [7] as well as an example solution to the whole problem in OMT notation [19].

Core problem description [7]: "In this example we are assuming that the company maintains an education department whose function is to run a number of training courses. Each course is offered at a number of different locations within the company. The database contains details both of offerings already given and of offerings scheduled to be given in future. The details are as follows: For each course: course number (unique), course title, course description, details of prerequisite courses (if any), and details of all offerings (past and planned); For each prerequisite course for a given course: course number and title; For each offering of a given course: date, location, format (e.g., full-time or half-time), details of all teachers, and details of all students; For each teacher of a given offering: employee number and name; For each student of a given offering: employee number, name, and grade"

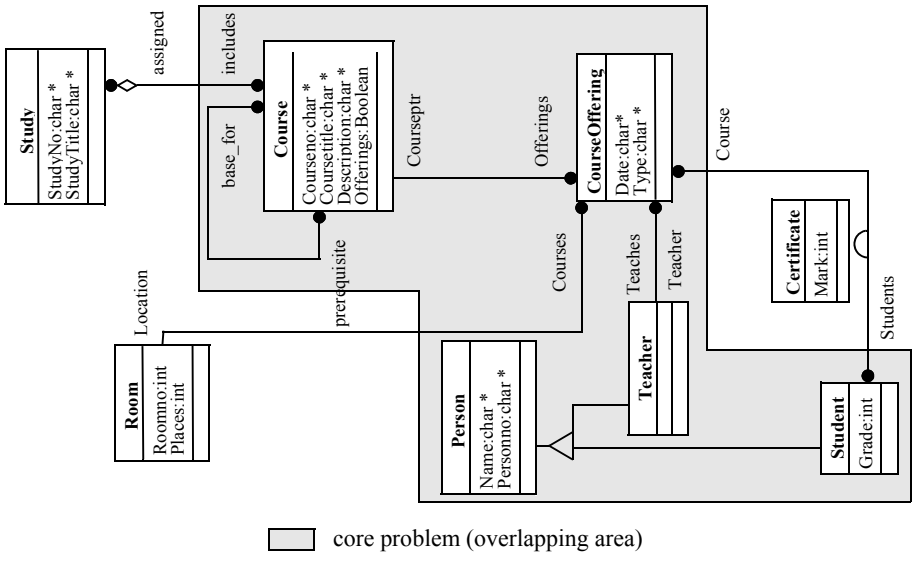

Fig. 2: The core problem specification and a possible solutions to the whole problem

4. Concept Classification

4.1 View Parsing

Mapping the semantics to the representation language

As mentioned in Section 3 we use the OMT object model for the description of the data model. The semantics of the static aspects (object classes, relationships between classes, attributes of object classes and their method signatures) are candidates for integration in a view comparison process. After several experiments we decided not to incorporate method signatures because they do not really contribute to find similarities between structural concepts, which is the task of our investigations. This is because first, one and the same method might be assigned to a number of object classes. Consider the following example. The application working on the data of our example should provide functionality to establish the assignment of teachers to courses and thus creating a specific course to be offered. The resulting method might be assigned equally well to a class representing a teacher or to a class representing a course. Second, a given functionality may be decomposed and distributed over a number of smaller methods which collectively perform this function. Yet, since various different decomposition strategies are possible, the surface representation of the functionality as given by the respective signatures might be rather different. Therefore, method signatures alone are not enough to cover the methods intention or the task of a set of methods representing one functional component.When comparing different views not only for the purpose of database view integration but for application development, functional aspects have to be compared as well. But the application aspects are beyond our approach and thus we have omitted the functional aspects.

Consequently, we only focus on the object classes, relationships between object classes and attributes of object classes in our approach. These concepts have to be translated into a representative language which is able to capture the semantics of the OMT model. In our approach we decided to translate into C++ like class definitions following the rules specified below: Each class of the OMT model builds one C++ class in the header file. The names and types of attributes are also expressed directly as data member names and types in the corresponding C++ class. A generalization is also converted following the C++ syntax for the definition of super- and subclass relationships. Associations and aggregations are both resolved in C++ by pointers if the number of participating classes is 1 or by a set of pointers to the related class otherwise. If a link attribute is attached to a relationship the pointers do not reference to the related class, but to the class which is built from the link attribute. The translation of OMT into C++ header files is illustrated in Figure 3 which includes all classes related to a student.

Extract semantic meaningful concepts

The next task is to identify and resolve all the semantic meaningful concepts of the view definition included in the C++ like representation language. To classify the semantic similarity and dissimilarity between the object classes we parse the header files to extract names of the classes, names of the attributes, types of the attributes and

```
class Person
{
 char* Name;
 char* Personno;
 char* Position;
};
```

```
class Student : public Person
{
 int Grade;
 Set<Certificate*> Course;
};
```

```
class Certificate
{
 Student* Students;
 CourseOffering* Course;
 int Mark;
};
```

Fig. 3: C++ like class definitions

links to other classes. We suppose that classes are assigned with meaningful names. Therefore, the strongest indicator of the real world correspondence between object classes is the equality of their class names. Nevertheless, there might exist different synonyms to identify similar classes, which are not detected in a syntactically analysis. One way to overcome this problem is to use a thesaurus. Since such a thesaurus must be context specific and thus cannot be generated automatically, we omitted to use this kind of synonym lexicon. Therefore, we better have a look inside the class definition. Similar to the concept described above it is assumed that at least some attributes owned by corresponding classes are labeled with equal names. Consequently, knowledge about the terminology used for their attributes names might be an indicator for the similarity between object classes. Additionally, information about the types of attributes used within a class can be regarded as a discriminator to determine the likelihood that two classes describe similar real world objects. This is due to the fact, that database designers using the same technology have comparable know-how in designing a well-structured database and thus incline to create similar views for a similar problem specification. Owing to the same reason, the links to other object classes might conform. Since these links are resolved to pointers or sets of pointers in the representation language, they can be treated like data types. Moreover the naming of the links can be handled like the naming of the class attributes. According to the above mentioned criteria we have to parse the C++ header files to extract the so-called classifiers to characterize an object class: class name classifiers (set of class names), attribute name classifiers (set of attribute names and labels of links), attribute type classifiers (set of attribute types and of types of links to other object classes).

4.2 Vector Initialization and Instantiation

Initialize the vector

In order to train a neural network with the view information, it is necessary to create a vector describing the semantics of each class. Therefore, we first have to initialize the format of such a class vector, which will be made up by the classifiers included in the three classifier sets. Since the equivalence of the class names is a stronger classifier than the equivalence of one attribute name, which is itself a more powerful criteria than the occurrence of an equal data type, the vector has to include a weight value for each sort of vector element that represents the relative importance of the various classifiers. In particular, we used a weight value of 3 for the class name, a weight value of 2 for the attribute names and finally, a weight value of 1 for the attribute types. Notice that such a weighting strategy was selected heuristically and proved to be reasonable during our experiments.

Instantiate the vector

Having now defined the format of the class vector, we parse the header files of the representation language again to instantiate a vector for each object class. In consideration of the weights of the vector elements we use the following rules for the instantiation: Vector elements describing the class name are set to 3, if the class is specified with the name in question or left 0 otherwise. Vector elements describing the attribute names are set to 2, if the class includes an attribute or a link with the corresponding name or are left 0 otherwise. Vector elements describing the attribute types are set to the number of occurrences of the corresponding type or link in the class. In the case of a generalization hierarchy the vector(s) of the superclass(es) is (are) added to the vector of the subclass to take advantage of the inheritance feature. This means that e.g. a student is also a person and therefore also inherits all the semantics of a person. Consequently, we have to add the semantics of a person to the student specific semantics to describe the student. Figure 4 states an example vector format and shows the transformation of the semantics included in the representation language to the corresponding vector representation of the class `Student.` As mentioned above, the vector representation of `Student` is built by the addition of the vector representation of `Person` and the vector representation resulting from the `Student` specific semantics. Note, that the vector elements not explicitly depicted in Figure 5 are all set to 0.

4.3 Training the net

Based on the basic description of the self-organizing feature map as provided in Section 2 we are now able to describe the learning process in terms of our application domain. The ultimate goal of the learning process is to provide the designer with a classification of the various class descriptions according to their mutual similarities. Obviously, we expect that classes modeling closely related objects of the real world are recognized by the self-organizing feature map. More precisely, the training a self-organizing feature map is a repetitive process consisting of the presentation of a randomly selected class description, i.e. the input vector, and a subsequent adaptation of the weight vectors of units in vicinity of the winning unit, i.e. the unit with the weight vector most closely resembling the current input vector. In other words, the winning unit may be regarded as the unit with the most similar internal representation of the input at hand. Such a point of view is certainly justifiable since the weight vectors may be regarded as approximations to the input vectors. The subsequent adaptation refers to the reduction of distance between the input and weight vectors and thus, to an improved correspondence between the class description at hand and its internal representation.

4.4 Exploring the Results

One typical training result of the self-organizing feature map is depicted in Figure 5. In particular, this figure represents the final arrangement of the various class descriptions mapped onto a 3×3 self-organizing feature map. In this figure we provide the names of the classes represented by the various units. Additionally, each unit is further desig-

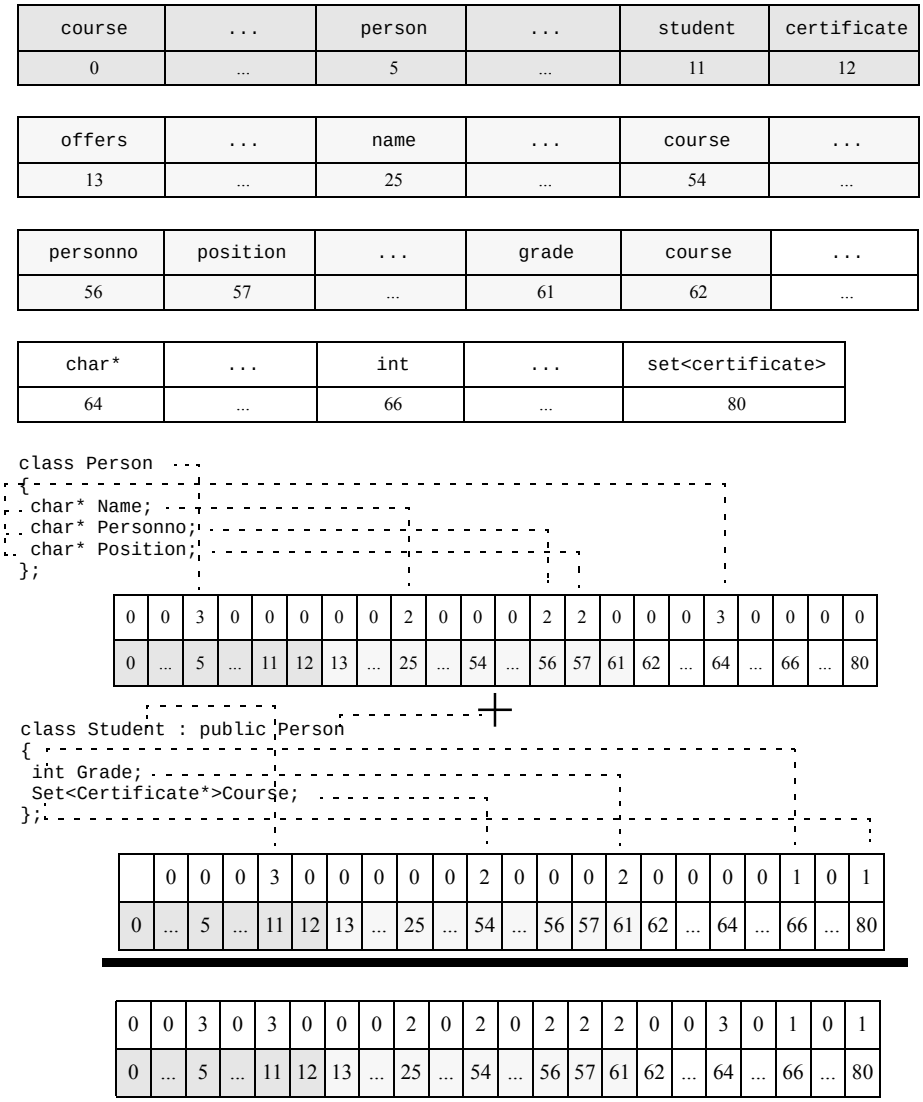

Fig. 4: Vector representation of class Student

nated by a number. At this point we should state that there is no formal justification of using a self-organizing feature map of exactly that size, it rather turned out to be practicable during a number of test runs with various sizes of the map. At first sight we may realize that two units, i.e. 7 and 8, represent a rather large set of classes. As might be expected, these units represent exactly the overlapping part of the case study namely the classes modeling persons and the classes modeling courses. More precisely, unit 7 comprises the persons whereas unit 8 represents the courses. The remaining units are used to represent classes that are not contained in the core problem description and thus, classes that are highly dissimilar since they represent the subjec-

Room.1, CourseRoom.5 1	Place.3 2	Department.3 3
CourseParticipant.5 4	Certificate.2 5	Study.2 6
Person.1, ProjectPartner.1, Sponsor.1, CourseTeacher.1, Teacher.2, Student.2, Person.2, Teacher.3, Student.3, TeacherExtern.3, TeacherIntern.3, Contents.4, Employee.4, Vehicle.4, Teacher.4, Participant.4, 7	CourseOffering.1, Course.1, Course.2, CourseOffering.2, Course.3, CoursePlanned.3, CoursePast.3, Course.4, CourseOffering.4, Workshop.4, CourseOffering.4, Course.5, CourseHeld.5, CoursePlanned.5 8	Project.1 9

Fig. 5: Arrangement of classes within a 3x3 self-organizing map

tive view of the various designers on the description of the case study. The advantages of using the self-organizing feature map are obvious from a view integration point of view. By assigning the input classes to the output units, the self-organizing feature map was able to group the potentially overlapping input classes that originate from different view specifications into categories of related object classes. Thus, the units represent clusters of similar object classes referring to closely related concepts of the real world. Hence, a tool for view integration is now available that facilitates the determination of classes referring to similar real world objects. This apparently is of considerable value because the designers can direct the attention on classes grouped into the same category by the self-organizing feature map. If we reconsider that the main manual effort involved during view integration is uncovering the correspondences between various view specifications and this process is now reduced to a subset of all possible inspections. By using this tool, the designer is able to distinguish between reasonable and unreasonable candidates for integration and thus, integration can be performed with less manual interference and consequently at less time and cost.

5. Related Work

View integration has already a long tradition as a research discipline. A comprehensive survey of the area can be found in [1] in which the authors discuss twelve methodologies for database or view integration. Early work has been done in the context of the relational model [3], the functional model [17], or more recently the Entity-Relationship model (see for example [15], [21], [10]), or the object-oriented data model (see

for example, [12], [2] or [13]). Most of the early work has focused on how to merge a given number of views by using a given set of correspondences. More recently, the interest of researchers has changed and more work is done for providing assistance during the view comparison phase. [18] was the first who developed a taxonomy for types of conflicts that might occur by integrating different views. [4] compares by means of a unification process concepts in different views by analysing similarities among names, structures, and constraints. In [20] a method for identifying view similarities based on using a semantic dictionary and a taxonomy structure is proposed while [11] uses conceptual graph theory to support view integration. Using concepts from the field of artificial intelligence to support view integration has also been done. [6] proposes an expert system approach to view integration. In [5] experiences are reported in using linguistic tools, such as thesauri and information retrieval techniques to build dictionaries and to support the view comparison process. In [9] fuzzy and incomplete terminological knowledge together with view knowledge is used to determine the similarity of object classes. Another approach is reported by [16] in which semantic integration in heterogeneous databases is achieved by using neural networks. Although their work is closely related to the approach reported in this paper there are fundamental differences. While [16] focus on the integration of populated databases we do focus on view integration and thus make no use of data that might be available. Their goal was to find equivalencies between fields in different databases by analysing the data while we address to capture a much higher level of real world semantics by analysing alternative view specifications.

The work presented in this paper is also related to our earlier work as described in [8]. In particular, the work presented in this paper extends our previous work in the following aspects: We include inheritance in class structures and study their impacts on integration. This is of particular interest in object-oriented views or views expressed in data models supporting generalization or subset hierarchies. We also considered methods of object classes as useful information for view comparison. But we encountered that the information included in the method signatures do not contribute to our comparison process. We experimented with extending the learning process by the following ways: After a first learning phase of the net and inspection of the resulting feature map we reduced the input into the net to only those classes that were grouped into the same category and trained the net again. By doing so we wanted to see if it is possible to increase the granularity of grouping. Unfortunately, the results were not promising. As a consequence we omitted this extension and finally experimented with the size of the output matrix.

6. Conclusion

Many organizations maintain different (often large and complex structured) databases with partly overlapping contents and there is an ever increasing need for supporting interoperable access. To fulfil this strong demand the views of the databases considered for building a federation need to be integrated into a single conceptual representation. In this paper we proposed a semi-automatic integration methodology which

significantly supports the manual integration task. The main contribution is that we use neural network technology to group related classes from different views based on their similarity into categories. The results of our experiments as described in this paper give a strong indication in the direction that our tool successfully identifies related classes as belonging to the same category. For large integration efforts this certainly is of important value because the designers are now relieved from the burden of manual inspection of all the classes and can direct their focus on a considerable subset of all the candidates for integration. Our approach is based on the following major features: Minimal input from a human analyst is required. The static semantic concepts of classes and their attributes supported by the data model under consideration is represented and included. We consider relationships and generalizations in the view comparison process. Extensibility to a variety of data models. To include a new data model only an interface to the representation language is to be developed, whereas the core component of the tool remains the same. Finally, the core component of the methodology proposed in this paper is also applicable to the schema integration process.

References

[1] C. Batini, M. Lenzerini, S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, Vol. 18, No 4, pp. 323-364, 1986.

[2] E. Bertino. Integration of heterogeneous data repositories by using object-oriented views. *Proc. 1st Int. Workshop on Interoperability in Multidatabase Systems*. Kyoto, 1991.

[3] J. Biskup, B. Convent. A Formal View Integration Method. *Proc. ACM Int. Conf. on Managment of Data (Sigmod)*, Washington, DC, 1986.

[4] M. Bouzeghoub, I. Comyn-Wattiau. View Integration by Semantic Unification and Transformation of Data Structures. *Proc. 9th Int. Conf. on the Entity-Relationship Approach*, North Holland, 1990.

[5] W. Bright, A Hurson. A Taxonomy and Current Issues in Multidatabase Systems. *IEEE Computer*, Vol 24, No. 10, 1990.

[6] F. Civelek, A. Dogac, S. Spaccapietra. An expert system approach to view definition and integration. *Proc. 7th Int. Conf. on the Entity-Relationship Approach.* Rome, 1988.

[7] C. Date. *An Introduction to Database Systems*. Addison-Wesley, 1991.

[8] E. Ellmer, C. Huemer, D. Merkl, G. Pernul. Neural Network Technology to Support View Integration. *Proc. 14th Int. Conference on Object-Oriented and Entity-Relationship Modelling* (OO-ER'95). Gold Coast. Australia. Dec 13-15. pp. 181-190. 1995.

[9] P. Fankhauser, M. Kracker, E. Neuhold. Semantic vs. Structural Resemblance of Classes. *ACM Sigmod Record*, Vol. 20, No. 4, Dec. 1991.

[10] P. Johannesson. A Logical Basis for Schema Integration. *Proc. 3rd Int. Workshop on Research Issues in Data Engineering (RIDE)*, IEEE Comp. Society Press, 1993.

[11] P. Johannesson. Using Conceptual Graph Theory to Support Schema Integration. *Proc. 12th Int. Conf. on the Entity-Relationship Approach*, Springer Verlag, LNCS 823, 1994. (R. Elmasri, V. Kouramajian, B. Thalheim, Eds.).

[12] M. Kaul, K. Drosten, E. Neuhold. ViewSystem: integrating heterogeneous information bases by object-oriented views. *IEEE 6th Int. Conf. on Data Engineering (ICDE)*, Los Angeles, 1990.

[13] W. Kent. Solving domain mismatch and schema mismatch problems with an object-oriented

database programming language. *Proc. 7th Int. Conf. on Very Large Databases (VLDB)*, Barcelona, 1991.

[14]T. Kohonen. *Self-Organizing Maps*. Springer-Verlag, Berlin, 1995.

[15]J. A. Larson, S. B. Navathe, R. Elmasri. A Theory of Attribute Equivalence in Databases with Applications to Schema Integration. *IEEE Trans. on Software Engineering (ToSE)*. Vol. 15, No. 4, pp. 449-463, 1989.

[16]W.-S. Li, C. Clifton. Semantic Integration in Heterogeneous Databases Using Neural Networks. *Proc. 20th VLDB Conference*, Santiago, Chile, 1994.

[17]A. Motro. Superviews: Virtual Integration of Multiple Databases. *IEEE Trans. on Software Engineering (ToSE)*. Vol. 13, No. 7, pp. 785-798, 1987.

[18]S. B. Navathe, S. G. Gadgil. A Methodology for View Integration in Logical Database Design. *Proc. 8th Int. Conf. on the Entity-Relationship Approach*, North Holland, 1982.

[19]J. Rumbaugh, et al. *Object-Oriented Modeling and Design*. Prentice Hall, Englewood Cliffs, NJ, 1991.

[20]W. W. Song, P. Johannesson, J. Bubenko. Semantic Similarity Relations in Schema Integration. *Proc. 11th Int. Conf. on the Entity-Relationship Approach*, Springer Verlag, LNCS 645, 1992. (G. Pernul, A M. Tjoa, Eds.).

[21]S. Spaccapietra, C. Parant, Y. Dupont. Model Independent Assertions for Integration of Heterogeneous Schemes. *The VLDB Journal*, Vol. 1, No. 2, pp. 81-126, 1992.