Neural Network Technology to Support View Integration

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 most difficult and time consuming activity to perform during view integration is to find correspondences between different view specifications. Such correspondences may be the source for conflicts when integrating the views and thus must be detected and resolved. A manual inspection of the class definitions in each view and a comparison with each class definition in the other views may result in an almost endless process. To support a designer we propose a computerized tool to extract the semantics from view definitions, to transform them into a unique vector representation of each class, and to use the class vectors to train a neural network in order to determine categories of classes. The output of the tool is a 'first guess' which concepts in views may be overlapping or which concepts do not overlap at all. This may be of tremendous value because the designers are relieved from manual inspection of all the classes and can direct their focus on classes grouped into the same category.

1. Introduction

For the development of large databases it is difficult to design the whole conceptual database schema at once. Schema integration, i.e. the process of deriving an integrated schema from a set of component schemata, is generally understood as an important activity during database design. This activity may be necessary in two design scenarios. The first is *database schema integration* for building database federations where a uniform canonical database schema is derived from existing, possibly heterogeneous, information systems and necessary to support interoperable access. The second is *view integration*, a process in classical database design, which derives an integrated schema from a set of user views. In general, if requirements on a database were developed by different design teams and are stated on the basis of individual requirements documents and schema specifications, there is a need for integration. Although related to all aspects our main focus in this paper is on the view integration process.

The view integration process can be divided into three main phases, namely view comparison, conflict resolution, and actual view integration. *View comparison* involves analysis and comparison of schemes to determine correspondences, in particular different representations of the same concept. *Conflict resolution* involves modifying one or more of the schemes to be integrated until each conflict is resolved and the conflicting concept is represented equally in each schema. The *actual view integration* generates the integrated schema by merging the individual view schema representations.

The most difficult part of the schema integration process is view comparison. The fundamental problems here lie in the fact that one concept of reality can be modelled in different views differently. This may be because the same phenomenon may be seen in different views from different levels of abstraction or may be represented by using different properties. The key in detecting such conflicts is proper understanding of the intended meaning of object classes and based on certain criteria to identify semantic similarity between classes from different views.

In this paper we try to solve the view comparison problem by assuming that concepts in different views that refer to the same real world semantics will have similar features, such as their names, attributes, domains, links, functions, or occur in similar contexts. Our assumption is 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 designer which concepts may be overlapping with other concepts in other views. For view comparison we use neural networks because of two reasons: First, neural networks are robust in the sense of tolerating 'noisy' input data (overlapping classes represented by similar features) and as a second reason, we refer to their ability of generalization. Pragmatically speaking, a neural network learns conceptually relevant features of the input domain and is thereby able to structure the various concepts accordingly.
This paper is organized as follows: In the remaining part of Section 1 we give an overview of the proposed architecture for view comparison based on neural network technology. Section 2 presents related work performed in the area of view integration. Section 3 contains a description of a case study. Section 4 shows how the view specifications to be compared are transformed in neural net input data and is devoted to the analysis of the neural network, its learning process, and to a discussion of the experimental results achieved. Section 5 contains our conclusions and hints on future research.

1.1 Overview of the proposed architecture for view comparison

For view comparison we propose an architecture consisting of four main building blocks (see Fig. 1) and several major design activities. The main goal of the proposed tool is to minimize human activities during view comparison. In our methodology only manual input of view specifications and human interpretation of results is required. Individual user views are specified using the object model of OMT [17]. The first computerized activity is a translation of the OMT schema definitions into C++-like database class definitions. The class definitions serve as an input to a specific parsing activity during which the semantic meaningful and relevant information for view comparison is extracted from the C++ class definitions. This completes building-block 1 which we call the *data extraction* phase. Building-block 2 is called *data transformation* and is responsible for determining the input vector to train the neural network. We refer to each semantically meaningful unit that is determined while performing building-block 1 as a classifier, and each classifier will serve as an element within the vector representation of a class. The next activity is instantiating the vector and consists of parsing the meta-data to determine for each class which of the classifiers are satisfied. This completes building-block 2. During building-block 3, *training process*, the resulting class vectors are used to train the neural network to recognize categories of classes. Human intervention is again necessary in building-block 4, *cluster interpretation*, to check and confirm the results.

2. Related work

View integration has been a research topic for more than fifteen years. A survey of the area can be found in [2] 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 [15] or more recently in the context of the Entity-Relationship model [5] (see for example [12], [19], [8]). Early work has focused on how to merge a given number of schemes 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. [16] was the first who developed a tax-

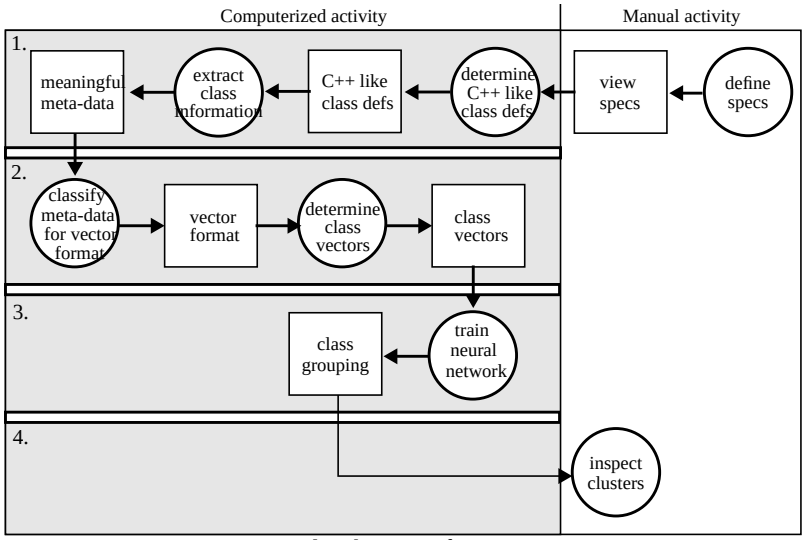


Fig. 1: Proposed architecture for view comparison

onomy for types of conflicts that might occur by integrating different schemes. [1] compares by means of a unification process concepts in different schemes by analysing similarities among names, structures, and constraints. In [18] a method for identifying schema similarities based on using a semantic dictionary and a taxonomy structure is proposed while [9] uses conceptual graph theory to support schema integration. Another idea is to use linguistic tools, such as thesauri and information retrieval techniques to build dictionaries and to support the view comparison. In [4] such experiences are reported. In [7] fuzzy and incomplete terminological knowledge together with schema knowledge is used to determine the similarity of object classes. An innovative and new approach is reported by [13] in which semantic integration in heterogeneous databases is achieved by using neural networks. Although their work is related to the approach reported in this paper there are fundamental differences. While [13] focus on the integration of existing databases we do focus on view integration. Their goal was to find equivalencies between fields in different existing databases while we address to capture a much higher level of real world semantics. Their input into the neural net is driven by data content and a flat table file structure while we use conceptual database structures expressed in a high level data model as input to train the net. Their results may be applicable to develop a canonical data model for a database federation while the results of our efforts may be seen as a ‘first guess’ which concepts in user views may be overlapping in other views. In large design teams and in database environments with a large set of heterogeneous users this information may be of tremendous value because the designers are relieved from manual inspection of a great variety of different object classes.

3. A view comparison case study

In order to evaluate our approach to view comparison, we carried out a case study. Usually, view comparison is necessary if a number of analysts 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 analysts 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 used the well-known Date example [6] 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. Fig. 2 shows the core problem specification from [6] as well as sample solutions to the example in OMT notation [17].

Core problem [6]: "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"

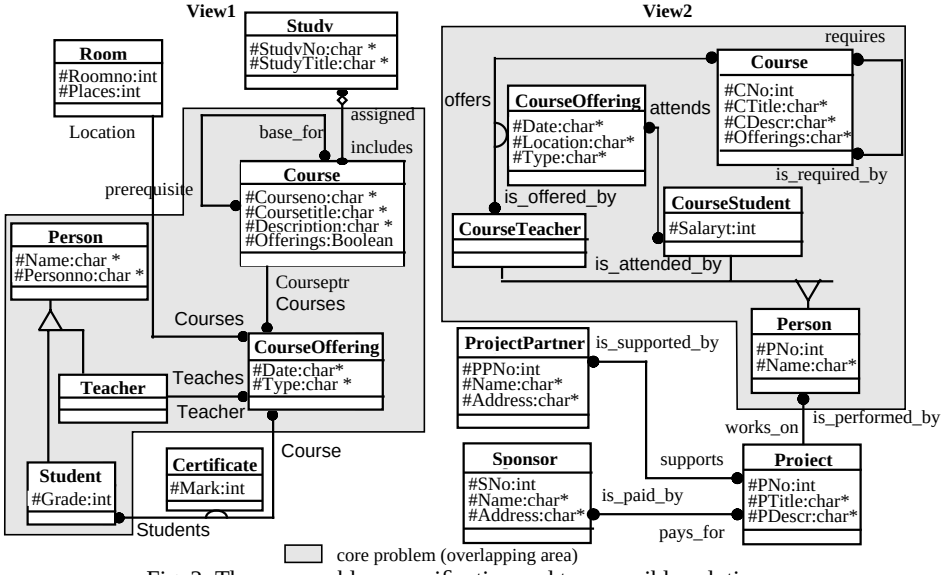


Fig. 2: The core problem specification and two possible solutions

4. A 'neural' tool supporting view comparison

4.1 Data Extraction

Determine C++-like class definitions.

As mentioned in Section 3 we use the OMT technique for the description of the data schema, or more specific the OMT object model, which describes the static structural and relationship aspects of a system. The utilities which the object model provides to capture those concepts from the real world that are important to the system have to be mapped into C++-like constructs: Each class of the OMT model is expressed as a C++ class. The names and types of attributes are expressed as data member names and types in the C++ header file. A generalization is converted according to the C++ syn-

tax 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. Note, at the moment our approach is based on the static structure of the data model and thus, we do not use the notation of operations.
The two view definitions presented in Fig. 3 include a subset of the classes depicted in Fig. 2. Both views include among others a class `CourseOffering`. View 1 includes the class `Student`, whereas view 2 includes a similar class `CourseStudent`. Furthermore, view 1 includes the class `Certificate` and view 2 the class `Project`. Consider the classes of view 1 for a closer inspection. `CourseOffering` directly adopts the attribute names and types from `Date` and `Type` of the OMT model. Since the number of the related `Code`, `Room` and `Teacher` is 1 in each case, pointers to these classes are established and designated by the relationship attribute. Note, the class instance specifying the room is termed `Location` in both views, but is a pointer to a separate class `Room` in view 1 and a simple `char*` in view 2. The same rules are applied to the class `Student`, but since it is a subclass of `Person` the statement '`public Person`' is added at the beginning of the class definition. Since a certificate is issued for each course the link attribute `Certificate` is added to the n:m relationship between `Student` and `CourseOffering`. Therefore, `Students` and `CourseOffering` do not have a set of pointers referencing each other, but each one has a set referencing to `Certificate`. Each `Certificate` includes exactly one pointer to each `Student` and to each `CourseOffering`. The n:m relationship is split up into two 1:n relationships via the link attribute `Certificate`.

View 1:

```
class CourseOffering {
 protected:
  char* Date;
  char * Type;
  Course* Courseptr;
  Room* Location;
  Teacher* Teacher;
  Set<Certificate*> Students; };
```

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

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

View 2:

```
class CourseOffering {
 protected:
  Course* offers;
  CourseTeacher*
          is_offered_by;
  char* Date;
  char* Location;
  char* Type;
  Set<CourseStudent*>
          is_attended_by; };
```

```
class CourseStudent :
          public Person {
 protected:
  int Salary;
  Set<CourseOffering*> attends; };
```

```
class Project {
 protected:
  int PNo;
  char* PTitle;
  char* PDescr;
  Set<Person*>
          is_performed_by;
  Set<ProjectPartner*>
          is_supported_by;
  Set<Sponsor*> is_paid_by; };
```

Fig. 3: C++-like class definitions

Extract class information.
It is difficult to identify and resolve all the semantic heterogeneity among components. In our approach we use three features to classify semantic similarity and dissimilarity: the names of the classes, the names of the attributes and the types of the attributes.
It is assumed that some classes or at least some of their attributes and/or relationships are assigned with meaningful names. Therefore, the knowledge about the terminological relationship between the names can be used as an indicator of the real world correspondence between the objects. However, similar classes and attributes might have different synonyms, which are not detected by inspection. This shifts the problem to building a synonym lexicon. A general purpose synonym lexicon cannot be specified, the synonyms must rather be specified according to the semantics of the problem

description. Thus, such a lexicon cannot be generated automatically and is expensive to build. Consequently, we omitted a synonym lexicon in our approach. Furthermore, it is assumed that given a database design application, different designers tend to produce similar schemata because they use the same technology and have comparable knowledge about designing a 'good' database. Thus, 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.
To extract the semantic meaningful meta-data all C++ header files including the class definitions have to be parsed. The result of the parsing process are sets containing all meta-data: class names set, attribute names set, data types set. Each element of these sets serves as a so-called classifier to characterize a class according to the above mentioned criteria. Fig. 4 shows these sets, if only the views of Fig. 2 were parsed.

class names:	attribute names:			attribute types:	
study	studyno	name	is_supported_by	char*	set<sponsor*>
course	studytitle	personno	is_paid_by	set<course*>	project*
courseoffering	includes	date	works_on	boolean	set<project*>
room	courseno	type	ppnr	set<study*>	
teacher	coursetitle	courseptr	supports	set<courseoffering*>	
student	description	location	sno	int	
certificate	offerings	teacher	address	student*	
person	assigned	teaches	pays_for	courseoffering*	
courseteacher	base_for	grade	cno	course*	
coursestudent	prerequisite	offers	ctitle	room*	
project	roomno	is_offered_by	cdescr	courseteacher*	
projectpartner	places	is_attended_by	is_required_by	set<certificate*>	
sponsor	courses	pno	requires	teacher*	
	students	ptitle	salary	set<coursestudent*>	
	course	pdescr	attends	set<person*>	
	mark	is_performed_by		set<projectpartner*>	

Fig. 4: Classifiers

4.2 Data Transformation

Classify meta data for vector format.

Training a neural network with the class information requires the creation of the format of the vectors describing the classes. In particular, each of the different class names, attribute names and attribute types used as a classifier will be an element of the vector. The classifiers 1 to n (n represents the number of different class names) are used to mark the class name. The elements n+1 to n+m (m represents the number of different attribute names) indicate the use of the corresponding attribute names within the corresponding class. The elements n+m+1 to n+m+p (p represents the number of different attribute types) are used to describe the types of attributes used in the class at hand. Fig. 5 depicts the vector formats resulting from the classifiers in Fig. 4.

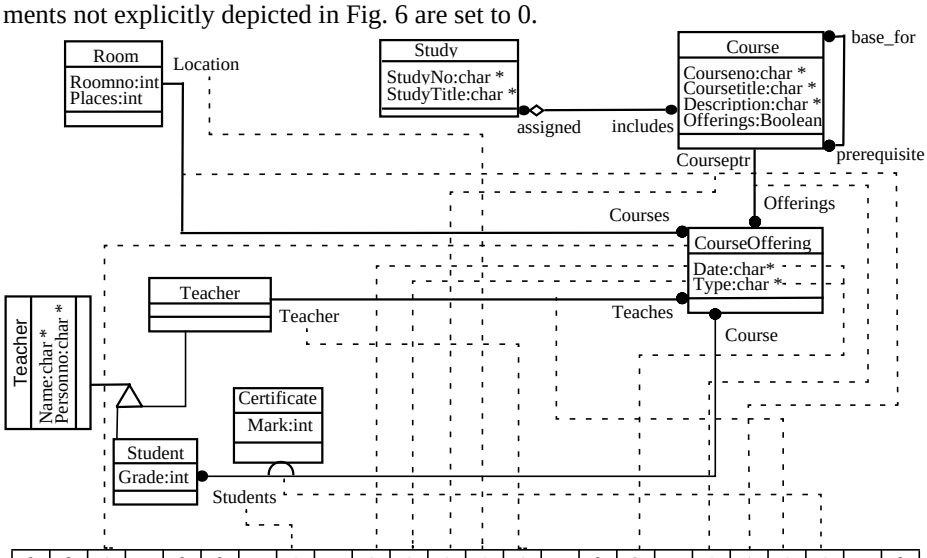
1	2	3	...	13	14
study	course	courseoffering	...	sponsor	studyno
...	27	...	32	33	34
...	students	...	date	type	courseptr
35	36	...	60	61	...
location	teacher	...	attends	char*	...
69	70	71	72	...	79
course*	room*	teacher*	set<certificate*>	...	set<project*>

Fig. 5: Vector format

Determine class vectors.

An additional parsing of the C++ class definitions creates a vector for each class. The vector elements describing the class name are set to 1, if the class is specified with the

name in question or left 0 otherwise. Similarly, the elements describing the attribute names are set to 1, if the class includes an attribute with the corresponding name or are left 0 otherwise. The elements describing the attribute types are set to the number of occurrences of the corresponding type in the class. Fig. 6 shows the transformation of the semantics included in OMT to the corresponding vector representation of the class by using class `CourseOffering` of view 1 as an example. Note, the vector elements not explicitly depicted in Fig. 6 are set to 0.



0	0	1	...	0	0	...	1	...	1	1	1	1	1	...	0	2	...	1	1	1	1	...	0
1	2	3	...	13	14	...	27	...	32	33	34	35	36	...	60	61	...	69	70	71	72	...	79

Fig. 6: Vector representation of class CourseOffering

4.3 Training process

We selected the self-organizing map (SOM) [10], [11] as the architecture to represent similarities within the classes obtained from different designers. The motivation to use exactly the SOM is mainly due to its unsupervised learning rule. Unsupervised learning refers to adaptation of the neural network's structure in order to enable the construction of internal models capturing the regularities of the input domain without any additional information. With additional information we refer to concepts such as a desired output that has to be specified with supervised learning models. For an overview of neural networks in general we refer to [14].

The utilization of neural networks is justified in that they are robust, i.e. they tolerate noisy input data. This fact is of essential importance during view comparison since different designers tend to model the same fact of reality slightly different, e.g. names of classes, names of attributes, types of attributes. These at first sight minor differences constitute major obstacles for conventional approaches, e.g. keyword matching. Moreover, we feel that an adequate representation of these differences for a knowledge-based system is not achievable with satisfactory generality and in reasonable time.

Basically, the SOM consists of a layer of *n* input units and a grid of output units each of which has assigned an *n*-dimensional weight vector. The task of the input units is to propagate the input vectors as they are onto the grid of output units. Each output unit computes exactly one output value which is proportional to the similarity between the current input vector, i.e. the description of one class in our application domain, and the

unit's weight vector. This value is commonly referred to as the unit's activation (the unit's response). Commonly, similarity is measured in terms of the Euclidean distance between the two vectors. In terms of our application domain we may regard the activation of an output unit as the degree of correspondence between the description of a class, i.e. the input vector representing the class, and the internal representation of that very class, i.e. the weight vector of the output unit.
The SOM learning rule may be described in three steps which are to be performed repeatedly. First, one input vector at a time is randomly selected. In other words, one class description is taken as input to the SOM. Second, this input vector is mapped onto the grid of output units of the SOM and the unit with the strongest response is determined. This very unit is further referred to as the winning unit, the winner in short. Notice that in case of Euclidean distance metric the unit with the smallest distance between input and weight vector is selected as the winner. Hence, the winner is the output unit representing the most similar internal representation of the class description at hand. Third, the weight vector of the winner as well as weight vectors of units in topological neighborhood of the winner are adapted such that these units will exhibit an even stronger response with the same input vector in future. This third step refers to the reduction of distance between input and weight vectors of a subset of the output units and thus, to the improved correspondence between the description of a class and its internal representation. The distance reduction may easily be accomplished by a gradual reduction of the difference between corresponding vector components. The adaptation is further guided by a learning-rate in the interval [0, 1] determining the amount of adaptation and a neighborhood-rate determining the spatial range of adaptation. In order to guarantee the convergence of the learning process, i.e. a stable arrangement of weight vectors, the learning-rate as well as the neighborhood-rate have to shrink in the course of time. In other words, the amount of adaptation of weight vectors decreases during the learning process with respect to a decreasing learning-rate. Furthermore, the number of units that are subject to adaptation, i.e. the spatial range of adaptation, decreases as well during the learning process such that towards the end of learning only the winner is adapted. Given these two restrictions it is obvious that the learning process will converge towards a stable arrangement of weight vector entries. Moreover, the SOM will assign highly similar input data, i.e. classes that have a highly similar description, to neighboring output units thanks to the inclusion of a spatial dimension to the learning process. It is the spatial dimension that distinguishes the SOM the most from other unsupervised learning architectures and thus makes it suitable for an application such as view comparison.

4.4 Cluster interpretation

Based on the description of the various classes as described above we trained a SOM to structure the classes according to their similarity. In the following we will present one typical arrangement of the classes in Fig. 7. The figure represents the result obtained from a 3×3 SOM. In other words, we mapped the class description onto a square of nine output units. For each unit we provide the list of classes that are represented by the unit. Additionally, each unit is further designated by a number. Please note that there is no formal justification to use a SOM of exactly that size. This size rather turned out to be useful in a number of training runs.
Just to start the description of the result with the units representing only one input, i.e. units 2, 5, 8, and 9, we may arrive at the following conclusion. As described in Section 3 we asked all designers to specify additional classes that may be relevant to the basic example but are not contained in the core problem specification. It is obvious that these classes represent a highly subjective view of the various designers on the problem at

hand and thus no correspondence to other designers may be found. As expected, the SOM successfully identifies these classes and assigns them exclusively to output units. With a simple 'manual' analysis of the problem description we may realize that the most important aggregations of classes refer on the one hand to persons, be it teachers or students, and on the other hand to courses. The latter group is represented by units 1 and 4. Thus, the person who has the responsibility of performing view comparison is pointed to a reasonable subset of the modelled classes that are all related to the real-world concept of a course. Turning now to the 'human aspect' of the problem description, most of the classes that model the real-world concept of a person are mapped onto units 3 and 6. Moreover, unit 3 comprises mostly students whereas unit 6 represents mostly lecturers. However, some distortion in the mapping may be observed. In this sense it is only unit 7 that represents a mix of rather unrelated concepts, namely some course classes and some person classes. Yet, such a distortion is still negligeable since the intention of the classification is to provide the designer with a 'first guess' on classes that may be integrated. Pragmatically speaking, due to the fact that the only alternative is manual inspection of all classes our classification provides substantial assistance to the designer.

CourseOffering.1, PlannedCourse.5, CourseOffering.2, CourseHeld.3, COffering.4 1	Room.1 2	Student.1, CourseStudent.5, CourseTeacherExtern.3, CourseStudent.3, CourseStudent.2 3
Course.1, Course.5, Course.3, Course.2, Course.4 4	Project.2 5	Study.1, Person.1, CourseTeacher.5, Registration.5, CourseRoom.5, CourseTeacher.3, Location.3, Person.2, ProjectPartner.2, Sponsor.2, Person.4 6
Certificate.1, Teacher.1, HeldCourse.5, CoursePlanned.3, CourseTeacherIntern.3, CourseTeacher.2, Contents.4, Workshop.4, Teacher.4, 7	Vehicle.4 8	Department.3 9

Fig. 7: Arrangement of classes within a 3×3 self-organizing map

The advantages of using neural network technology are obvious from a data modelling point of view. The SOM was able to group the potentially overlapping input classes that originate from different view specifications into categories of related object classes. Each category represents a cluster of similar object classes referring to closely related concepts of the real world. Hence, a tool for view comparison 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 their attention on classes grouped into the same category by the SOM. Please note, the main manual effort involved during view comparison is uncovering the correspondences between various view specifications and this process is now reduced to a subset of all possible inspections. By using the 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. Conclusion and further work

In this paper we reported on a novel approach to view comparison by means of neural network technology. As an illustrative example we used the description of an education department modelled by a number of people. Hence, the task of view comparison may be paraphrased as determining the correspondences between the views of differ-

ent designers onto the same reality. The distinguished features of our approach are an automated transformation of an OMT-style class description to a vector based class representation that is further utilized as the input to an unsupervised neural network. The neural network performs a classification of the various input classes based on their mutual similarity. In other words, classes referring to the same real world concept are grouped within the same cluster. As a consequence, the process of view comparison may now be regarded as the manual inspection of a subset of the input classes instead of the whole range of classes that are modelled by various designers. We were able to show that such a classification was performed to a highly promising degree.
Future work will concentrate on improved vector representation of the various input classes. At the moment we achieved the results provided in this paper on the basis of the static structure of the OMT data model. We may arrive at further improvements by an inclusion of the dynamic structure as well. Currently, we are investigating the possibilities to enlarge the vector representation to cover methods that are defined for the various classes as well.

References

[1] 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.

[2] 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.

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

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

[5] P. P. Chen. The Entity-Relationship Model - Toward a Unified View of Data. *ACM Trans. on Database Systems (ToDS)*, Vol. 1, No. 1, pp. 9-36, 1976.

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

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

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

[9] 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.).

[10]T. Kohonen. The Self-Organizing Map. *Proceedings of the IEEE* 78(9). 1990.

[11]T. Kohonen. *Self-Organizing Maps*. Berlin: Springer-Verlag. 1995.

[12]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.

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

[14]P. Mehra and B. J. Wah (Eds.). *Artificial Neural Networks: Concepts and Theory.* Los Alamitos, CA: IEEE Computer Society Press. 1992.

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

[16]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.

[17]J. Rumbaugh, et al. *Object-Oriented Modeling and Design*. Prentice Hall, 1991.

[18]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.).

[19]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.