TERZO CONVEGNO NAZIONALE.
SISTEMI EVOLUTI PER BASI DI DATI
Ravello, 28-30 Giugno 1995

Techniques for the Analysis of Conceptual Schemas

S.Castano (*), V.De Antonellis (**)
M.G.Fugini (+), B.Pernici (++)

(*) Università di Milano, (**) Università di Ancona
(+) Università di Pavia, (++) Politecnico di Milano

e-mail: [castano,deantone,fugini,pernici]@elet.polimi.it

Abstract

The problem of analyzing and classifying conceptual schemas is becoming more and more important due to the availability of large sets of schemas from existing applications. The purpose of the analysis and classification activities can be that of extracting information on schemas of legacy systems in order to migrate them to new architectures, to build libraries of reference conceptual components to be used in building new applications in a given domain, to analyze large sets of schemas in an organization to identify information flows and possible replication of data. The paper proposes a set of techniques to be adopted for schema classification and analysis: indexing techniques to associate a description with a schema, techniques for abstracting conceptual schemas based on schema clustering, and techniques for schema comparison. The application of these techniques in the context of reuse of conceptual components is briefly presented.

Keywords - Conceptual modeling, Schema analysis, Schema classification, Schema comparison.

1 Introduction

In the last two decades, large sets of conceptual schemas have been constructed, both in conceptual database design projects, and in software engineering projects, to describe data used by applications. In recent times, the need of analyzing and classifying these schemas has arisen in a number of application contexts: to extract information on schemas of legacy systems in order to migrate them to new architectures [1,7], to build libraries of reference conceptual models to be used in building new applications in a given application domain [15,10], to analyze large sets of schemas in an organization to identify information flows and possible replication of data [13]. In the present paper we focus on schema analysis, i.e., on analyzing intensional information associated with schemas; other analysis techniques are being proposed in the literature also to analyze data stored in existing databases, i.e., extensional information (data mining [4]).

The purpose of this paper is to present general techniques for schema analysis: to associate keywords with schemas, to compare schemas, and to create abstract schema representations. We assume to consider conceptual schemas in general, and most of the principles presented below are applicable to a wide set of conceptual schemas commonly used in conceptual database design, in requirements engineering, and in development methods presented in software engineering to support the early phases of application development.

These techniques can be used, separately or in combination, to derive from a schema, with human intervention limited as far as possible, significant properties of the schema. These analysis techniques can be applied for a number of different purposes: for instance, to analyze the quality of a set of schemas, to identify interesting portions of schemas in a repository, to extract reusable subschemas from previous project results.

We distinguish between techniques which are applied for the analysis of one schema, and techniques which are used to compare schemas.

Analysis techniques applicable to a single schema include techniques for:

- associating a set of descriptors with a schema (*schema indexing*);
- grouping elements in a schema according to the principles of high cohesion and low coupling, which are the basis for a good modular decomposition (*schema clustering*);

- creating an abstract representation of a schema, as a schema with less details (*schema abstraction*).

Techniques applicable to pairs of schemas allow the comparison of conceptual schemas, to identify *similarities*. The comparison of schemas is useful to classify schemas, e.g., to insert them in a repository in order to facilitate their retrieval in future projects, or to identify similar schemas which can provide a basis for building reusable components, or to retrieve in a large repository schemas representing similar data, e.g., to reconstruct organizational processes.

Related work in the literature is available in the area of information retrieval. Indexing techniques can be found in [24]. In the paper, we discuss how some existing techniques have been adapted in the conceptual modeling environment. Some specific work on similarity has been performed within the field of reusability. In [25], techniques for analyzing similarity of software and its specification is proposed. The idea of providing an abstract description of conceptual schemas has been proposed in the Entity-Relationship literature, to organize schemas in a repository [3] or to provide an abstract description of schemas [17]. Relevant work in this field is also being performed by [23] to provide generic components for domain modeling. On the other hand, all these approaches are labour intensive, and therefore difficult to adapt to very large sets of schemas. In [10,19], we proposed approaches to automatically cluster conceptual schemas and (semi)-automatically abstract them for building reusable components. In the present paper, this approach is generalized to schema abstraction based on general criteria.

Techniques for classifying reusable components have been presented by the authors in the reuse contexts [8,9,15,14]. The purpose of the present paper is also that of organizing all these techniques in a homogeneous framework, providing a set of techniques which can be applied to schema analysis in general.

The paper is organized as follows. In Section 2, we discuss techniques for associating keywords with schemas. In Section 3, we present techniques for (semi-)automatically building an abstract representation of schemas, based on clustering principles. In Section 4, we discuss problems related to the identification of similarities between schemas. Finally, in Section 5, we discuss applications of our techniques in the domain of reuse, and to the analysis of large sets of schemas.

2 Schema indexing

In this section, we describe our techniques for indexing a schema, that is, for associating with the schema a set of descriptors, capable of describing the contents of the schema at a more abstract level.

To the aim of defining techniques that are applicable to several conceptual models, we consider a conceptual schema S as a set of elements, that is, $S = \{E_1, \ldots, E_n\}$, where elements correspond to the constructs of the conceptual model used for defining S. For example, with reference to the Entity-Relationship model, elements of a schema are the entities and the relationships.

In our approach, descriptors are selected among the names of schema elements, using weight-based techniques. Descriptors can be single names or pair of names, called features, selected for their capability of describing the schema subject and contents in a suitable way. For descriptor selection, we adopt both a syntactic and a contextual approach. We consider the structure of elements in a schema to determine the relevance of elements and, consequently, select the representative elements (syntactic approach). Contextual considerations are taken into account by defining a list of features with an associated fuzzy weight, expressing the relevance of the features. In order to manage synonyms, homonyms, and other situations that can emerge from heterogeneous naming disciplines and jargons, we rely on the availability of a Thesaurus, where terms and relationships among terms are stored in a structured way. The probelms related to the construction and the maintenance of a Thesaurus are discussed in [10,14].

In the following, examples are given using the "Concept Model", an extended Entity-Relationship model defined by SISU within the F^3 project [16]. In Sect. 2.1 we illustrate the technique defined for selection of representative elements, and in Sect. 2.2, the technique used for selecting features for a given conceptual schema.

2.1 Representative elements

Representative elements are chosen as schema elements able to represent the content of a schema (portion), and their names become schema descriptors [10]. The identification of the representative elements for a given schema S is based on the following steps:

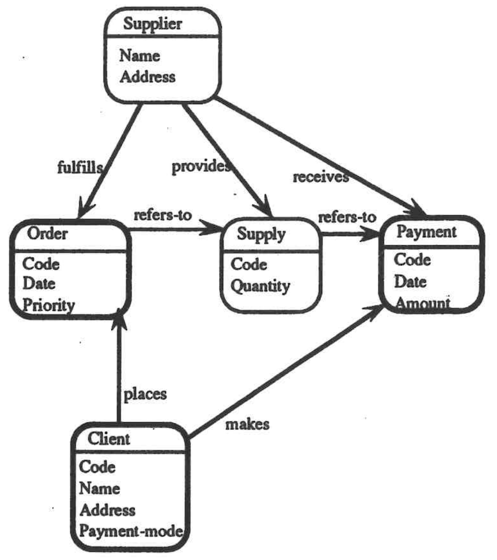

Figure 1: The "Supply Management" schema

- the quantity of information $W(E_i)$ for each element E_i belonging to S is computed;
- a threshold is defined for selecting the representative elements based on the computed quantity of information.

To compute the quantity of information $W(E_i)$ carried by an element E_i in a schema, we consider the following structural aspects:

- Number of *properties* of E_i, $A_{tot}(E_i)$; when considering models like the Entity-Relationship models, the number of attributes is considered. In Fig. 1, properties are shown in the element boxes.
- Number of links of E_i to other elements in the schema, $L_{tot}(E_i)$; when analyzing Entity-Relationship schemas and concept models, relationships linking entities and is-a hierarchies are considered.

$W(E_i)$ is computed as follows:

$$W(E_i) = A_{tot}(E_i) + L_{tot}(E_i)$$

The rationale is that the number of properties and links of an element can be used as a (heuristic) measure of its relevance within the schema. The greater this measure, the higher the relevance, since it is characterized by several properties and it is referred to by several elements of the schema.

To choose the representative elements in a schema S, based on $W(E_i)$, a threshold T is computed for S as the average quantity of information of the elements belonging to S. The elements for which $W(E_i)$ is greater than or equal to the threshold T are selected as representative elements.

Additionally, in conceptual models allowing the representation of is-a links between elements, the choice of representative elements is also based on the participation of elements in is-a links. In particular, a satisfactory criterion is that of choosing all elements which have a descendant in an is-a hierarchy. In fact, usually leaf elements in is-a hierarchies turn out to be too schema specific to be a good basis for schema comparison and abstraction.

For instance, in Fig. 1, an example of conceptual schema is shown, related to "Supply Management", defined according to the concept model of F^3 [16]. This conceptual model is basically a semantic network model, allowing the representation of concepts and relationships between concepts. The "Supply Management" schema in Fig. 1 represents the concepts involved in good supplying (such as suppliers, supplies, orders and clients, payments) and their relationships. When applying the formula previously illustrated to the schema of Fig. 1, we have that, for example, the quantity of information computed for both the concepts `Client` and `Order` is 6 (in fact, `Client` has 4 attributes and 2 links, while `Order` has 3 attributes and 3 links). The threshold T for the "Supply Management" schema is 5.6, which is the average quantity of information (i.e., 28/5).
The concepts that are selected as representative are shown with bold lines in the schema.

2.2 Features

In order to perform contextual analysis of elements and detailed comparison of elements, we introduce the concept of "Features". Features are pairs of keywords with an associated fuzzy weight expressing the relevance of the Feature. The mechanisms of keyword pairing and of

fuzzy weights adds semantics to the description. Correct semantics can be provided only via human intervention by the developer. However, Features and weights can be obtained semi-automatically, with the protocol described here.

Contextual analysis describes the properties, structure and role of schemas and elements taking into account the structure and characteristics of all the schemas stored in the repository.

The output of contextual analysis is a description of schemas through list of Features, allowing schemas to be indexed according to a flexible paradigm, and to be searched and retrieved using imprecise queries. Hence, the developer can navigate in the space of elements and schemas by entering concepts and by finding both perfectly matching candidates, and candidates that "can fit" the current needs. Fuzzy weights are the mechanism supporting imprecision, which is used to compute the degree of confidence of the suitability of schemas.

In detail, the mechanisms for contextual analysis are the following:

- keywords are used in pairs; a keyword pair is called a *Feature*;
- each Feature is weighted with a fuzzy weight, expressing how relevant the Feature is in the description of the schema.
  The *fuzzy descriptor* thus obtained has the following format:

list-of [Feature: fuzzy-weight]

Keywords are paired in Features under the assumption that a two-term structure is more descriptive than usual one-term descriptions [24], while remaining simple enough to avoid the need for a formal description language. The semantics is defined as a mapping between sets of string pairs into fuzzy sets of weights.
A possible interpretation of Features is the following:

- "Action-Object", where Action is a verb and Object is a noun. In this case, we speak of Active Features describing the actions executed by an element on other elements or on itself. An example is "Check-Order", a verb-noun pair, where "Check" is the action executed on the "Order" element.
- "Category-Value", where Category is a noun and Value is an adjective, and "Element-Property", where both are nouns. These are Passive Features describing the properties of an element. Examples are "Priority-High", a noun-adjective pair describing the

value held by the property "Priority" of "Orders", and "Order-OrderCode", a noun-noun pair describing the schema shown in Fig. 1.

In general, a schema is described by a Feature list, according to the *Features extraction protocol* described in the following. Features are extracted from conceptual schemas considering concepts, links, and attributes, and are used as schema descriptors.

First, representative elements of the schema are identified, using the method described in the previous paragraph. Then, the following indexing steps are applied:

a) All the representative elements belonging to an is-a hierarchy are considered, and, for each element, its key (identifying) attributes are extracted. The following Feature is extracted:

   (Element_Name - Key_Attribute)

   where Element_Name and Key_Attribute are the two keywords paired (using the '- ') in a Feature.

b) A *filter* is applied: by examining for each representative E_i the number of links L_{tot} in which E_i participates, only those elements are filtered whose L_{tot} exceeds the nearest integer greater than half of the maximum L_{tot} of all the representative elements. A Feature of the form:

   (E_i_Name - E_j_Name)

   is added to the descriptor, where E_j varies among all elements connected to E_i. Duplications are avoided, i.e., the directions of links are ignored.

c) The descriptor is completed with Features constructed out of names (if any, meaningful) of the links of the elements extracted in the previous steps, together with their key attributes.

The Features obtained in this way can be mostly created automatically. A manual inspection step can then be performed to correct some ambiguities or odd terms due to the special jargon used in the schema. In general, hand written Features could be more expressive than those automatically extracted; however, since we consider conceptual schemas, we rely on the expressiveness of terms, especially if these schemas are "well written", according to some methodological discipline that influences also the "goodness" of used terms. Instead, the problem of adding semantics to poorer descriptions, e.g., from code

libraries, where the lexicon is bound to hardware/software terms and jargons, can be more serious (this problem is tackled in [14]). Here, we observe also that the Thesaurus is a valid support to the extraction of meaningful Features, since a synonymia function filters out the influence of technical jargons and lexical differences.

Referring to the example of Fig. 1, the extraction protocol would produce the following Features (considering the three representative elements in bold lines):

```
Order-OrderCode
Client-ClientCode
Payment-PaymentCode
Order-Client
Client-Payment
Places-OfferNumber
```

where the first 3 Features derive from step a, the following 2 from step b (considering the connection degrees of the three representative elements), and the last Feature considers that the only relationship with a meaningful name is "places", and where we assume that "places" has the "OfferNumber" attribute (not shown in the figure).

Coming to the fuzzy weighting of the Features, an algorithm assigns fuzzy weights to Features using a function adapted from classical text retrieval systems. The adaptation takes into account that no statistical analysis can be performed on schema terms, since they do not belong to a true corpus of text documents. The formula is the following:

$$w_{i,k} = \frac{\nu_{i,k} \cdot \ln(\frac{N}{n_k})}{\sqrt{\sum_{j=1}^{F} (\nu_{i,j})^2 \cdot (\ln(\frac{N}{n_j})^2}}$$

where $w_{i,k}$ is the weight of the k^{th} Feature in the i^{th} element/schema (in i=1 ...N) and $\nu_{i,k}$ is the frequency of the k^{th} Feature in that element/schema, N is the total number of descriptors in the repository, and n_k is the number of descriptors exhibiting that Feature. F is the total number of distinct Features in the repository.

The fuzzy weight associated with a Feature is a real number in the interval [0,1] representing the relevance of the Feature within the de-

scription, that is, "how well" the Feature describes the element characteristics. The fuzzy weights are used within fuzzy logic [20] with standard operators (*and*, *or*, *logic implication*). Weights are involved in the computation of similarity using the algorithm illustrated in [14] returning the Confidence Value of the similarity.
For example, considering the Features extracted above, the weighted fuzzy descriptor is:

```
Order-OrderCode:0.8247
Client-ClientCode:0.8247
Payment-PaymentCode:0.8247
Order-Client:0.6044
Client-Payment:0.5012
Places-OfferNumber:0.3691
```

In these computations, we have considered $N = 20$, each schema described by 6 Features, a total of 120 features, of which 70 are distinct ($F = 70$). Moreover, n_k has been assumed to be 2, and $\nu_{i,k} = 1$, since we consider only 1 context in the repository.

`Order`, `Client`, and
tt Payment appear as the most frequent keywords and their Features get a high value of the weight, since these elements are the most connected elements of the schema, that is, among the most relevant elements of the repository.

3 Clustering and abstraction of schemas

Schema abstraction has been proposed in the literature (e.g., [3,17]) to represent the contents of a schema in a structured way, according to different levels of detail. We proposed in [10,19] approaches to abstract schemas based on clustering of schema elements. By means of clustering, we partition the elements of each schema according to criteria based on the well-known principle of "high cohesion and low coupling". The objective is to obtain highly cohesive subschemas, each of them representing one basic and indivisible concept. By means of abstraction, for each cluster, one schema element (or, possibly, more than one) becomes a representative element, for the underlying basic concept.

In Fig. 2 an example of schema clustering is shown, for the "Courier Transportation" schema. This schema represents the concepts in-

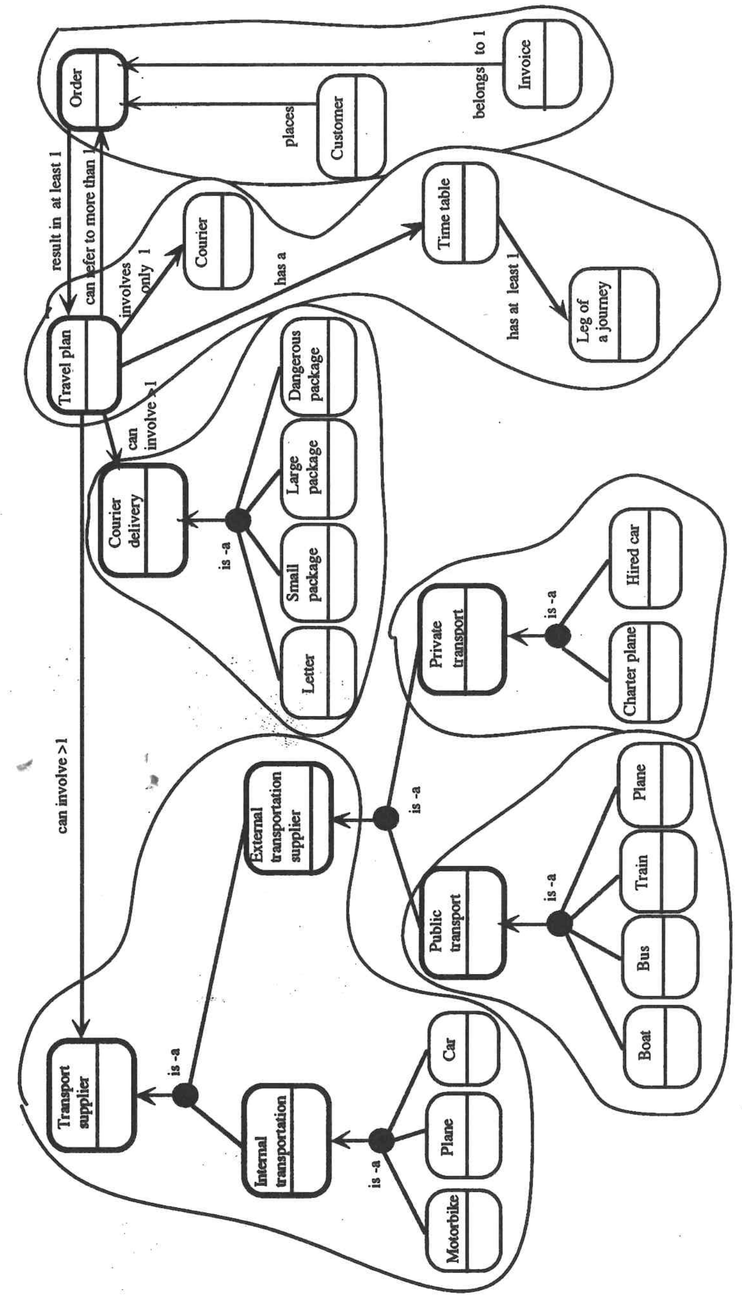

Figure 2: The "Courier Transportation" schema

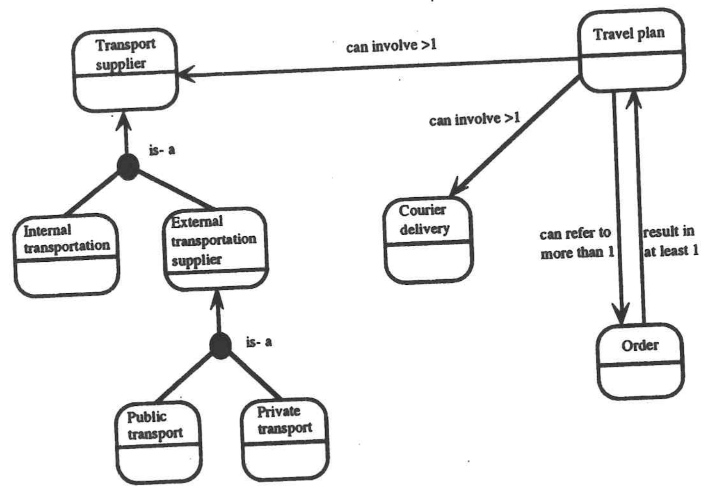

Figure 3: Abstraction of the "Courier Transportation" schema

volved in transportations made by couriers (such as transportation means, delivered goods, orders and travel plans) and their relationships. In Fig. 3 an example of abstraction of the "Courier Transportation" schema of Fig. 2 is shown. The "Courier Transportation" schema is abstracted by first clustering its elements as shown in Fig. 2, then by choosing an abstract representation for each cluster. For instance, the cluster composed by elements `Travel plan`, `Courier`, `Time table`, and `Leg of a journey`, is represented by the element Travel plan. In the following, we present an algorithm for performing schema clustering. Then we discuss criteria for representing clusters with abstract elements and for abstracting links between elements.

In the present paragraph, we generalize to schema analysis the techniques proposed in [10,19] in the context of reuse.

Clustering of schema elements can be performed taking into account the structure of the schema (syntactic approach) or the meaning of the elements (semantic approach). We focus mainly on the syntactic approach - that can be automatically supported - but some semantic considerations are taken into account on the basis of the schema descriptors defined in the previous section. To provide operational criteria for clustering we have introduced basic concepts of

affinity and *closeness* between schema elements. Definitions of coupling and cohesion between clusters are then given in terms of affinity and closeness.

3.1 Affinity

As already said, schemas are described via *descriptors*, which can either be names of representative elements or features (see Sect. 2.1 and Sect. 2.2). These descriptors can be used for computing similarity between elements. In particular, the terms contained in the descriptors of elements E_i and E_j can be examined to establish their level of similarity, with the help of a Thesaurus. We state that two elements E_i and E_j have affinity with respect to their descriptor terms, denoted by *Affinity*(E_i, E_j), if their terms are either equal, or synonyms, or similar, according to the relationship *Synonym*(t_h,t_k), defined in the Thesaurus. For details about the organization of the Thesaurus with affinity measures, we refer the interested reader to [14].

3.2 Closeness

To measure the level of closeness between elements in a schema, we consider the number and the type of links among them. Following [26], different types of links determine different levels of closeness among elements. In particular, "is-a" links implies a high closeness value, due to the parent-child relationship between a supertype and its subtype. According to these considerations, we assign a weight w_k to each type of link, to properly assess the strength of the corresponding relationship.

To measure the level of closeness between elements, we use a Closeness Coefficient, computed as follows:

$$Closeness(E_i, E_j) = \sum_{k=1}^{nl} w_k n_k(E_i, E_j)$$

where nl is the total number of types of links defined between elements in a given model, $\sum_{k=1}^{nl} w_k = 1$, and $n_k(E_i, E_j)$ is the number of links of type k relating elements E_i, E_j.
In computing closeness in concept models and Entity-Relationship schemas, we have chosen $w_{rel} = 0.4$ has the weight for relationship links, and $w_{is-a} = 0.6$ for inheritance hierarchies.

3.3 Coupling

We consider coupling between non overlapping portions of a schema, which are called *clusters* in the following.

Let us consider two clusters C_x and C_y containing n_x and n_y elements, respectively. The measure of coupling between C_x and C_y, denoted by *Coupling*(C_x, C_y), is defined as follows:

$$\mathit{Coupling}(C_x, C_y) = \frac{w_A \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} \mathrm{Affinity}(E_i, E_j) + w_C \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} \mathrm{Closeness}(E_i, E_j)}{n_x n_y}$$

where w_A, w_C, with $w_A, w_C \in [0,1]$ and $w_A + w_C = 1$, are weights introduced to properly assess the importance of affinity and closeness coefficients between elements for cluster coupling computation.

Note that the measure of affinity captures the knowledge on the semantic affinity between elements, either coming directly from the developer, or from the Thesaurus; conversely, the measure of closeness is based on purely syntactic considerations. As a consequence, different weights assigned to affinity and closeness vary the importance attributed to semantics with respect to syntax. In testing our method we have privileged closeness against affinity, due to the difficulties in gathering the knowledge on schema semantics (we normally assume $w_A = 0.1$ and $w_C = 0.9$).

3.4 Clustering

The goal of the algorithm for clustering illustrated in the following is to optimise a global measure of coupling producing a sub-optimal solution. In the information retrieval context, algorithms have been proposed for document clustering, based on measures of similarity between documents; the most widely used one [24] is based on the construction of a matrix, called similarity matrix, of rank n, where n is the number of documents to be clustered; element i, j of the similarity matrix represents the value of the similarity between documents i and j; in outline, the procedure for clustering is constituted by a series of steps, each of which consists of the merging of the most similar pair of clusters, the deletion from the similarity matrix of the rows and columns corresponding to the merged clusters and the insertion of a new row and a new column corresponding to the new cluster deriving from the merging. The algorithm can be adapted to schema clustering defining a matrix, called coupling matrix, corresponding to the similarity matrix defined for

document clustering. If c_{ij} is the generic element of the coupling matrix, and n_i and n_j are the number of concepts of respectively clusters C_i and C_j, the elements of the coupling matrix are defined as follows:

- c_{ij}= Coupling(C_i,C_j);
- $c_{ii} = 0$.

Initially, the coupling matrix has rank n, where n represents the number of concepts of the schema to be clustered. While the algorithm proceeds and concepts are combined into clusters, matrix elements are updated with the measure of their coupling with the new clusters. The algorithm for schema clustering by grouping is parametrized by the number of clusters k that are created; consequently, the procedure cycles until the initial n concepts have been grouped into k clusters, corresponding to the final rank of the coupling matrix.

The basic procedure for schema clustering by grouping is the following [19,24]:

```
1. Place each of $n$ concepts into a cluster of its own;
2. Compute all pairwise concept-concept coupling coefficients, and
   fill the coupling matrix;
3. While the number of clusters left is greater than $k$ do
       Select the most similar pair of current clusters $C_i$ and $C_j$;
       Combine $C_i$ and $C_j$ into a single cluster $C_{i+j}$;
       Delete from the coupling matrix the rows and columns cor-
       responding to $C_i$ and $C_j$;
       Insert a new row and a new column corresponding to the
       new cluster $C_{i+j}$ and calculate the coupling coefficients for
       the new row and column;
   endwhile.
```

3.5 Cluster abstraction

The purpose of the abstraction process is to provide a description of a schema in terms of more abstract elements. In fact, the availability of abstract elements makes easier both the identification of components shared by different schemas, and the retrieval and understanding of elements and/or schemas, e.g., within a reuse library.

Given a clustered schema, its more abstract (or higher level) representation is obtained according to two steps:

1. definition of abstract elements,
2. link restructuring.

As a result of defining the abstract elements, and of applying link restructuring techniques, we obtain what we call an *abstracted schema*, that is a high-level description of a given schema, composed of representative elements, connected through properly selected links. Abstracted schemas provide a concise representation of the key concepts of given domains.

Criteria guiding the execution of the two steps are discussed in the following, referring to the example shown in Fig. 2 and Fig. 3.

Abstract elements
To obtain abstract elements we select representative elements for each cluster, according to the approach presented in Sect. 2.1, and use their names as cluster descriptors. Descriptors become the label the corresponding abstract element.

Link restructuring
The set of links between abstract elements contains the links between initial elements in the original schema. Sometimes, a link needs to be renamed in order to be generalised. The correspondence between the initial name and the more abstract one are traced in the library to enrich the knowledge about synonyms, hyperonyms, and hyponyms in the domain. Furthermore, some types of links, for instance is-a, do not admit cycles. Including links originally holding between single elements of the initial schema as links between abstract elements can involve the presence of cycles in the final schema. In this case, either the dropping or the renaming of at least one link is compulsory.

The actions that can be performed to modify the initial set of links during abstraction are the following:

- dropping;
- renaming;
- transformation of a subset of the links into a single one, called link abstraction.

In general, the dropping of a link judged scarcely meaningful is suggested as a first action, in order to obtain a more readable schema. Link abstraction consists of the grouping of conceptually related links and of their transformation into a single link. Methodological criteria for the grouping of links are the affinity between their names, or the equality between the sets of participating entities in the source

schema. The definition of the abstract link is mainly based on the names of the original links.

When cardinalities are indicated, they can either be assigned either from scratch, or can be derived from those of the original links, for instance, as the union of their ranges of variability.

4 Similarities between schemas

In this section we describe our technique for analyzing pairs of schemas in order to discover their similarities. Similarities between a pair of conceptual schemas S_i and S_j are computed by considering:

- Descriptors associated with the schemas (*descriptor-based similarity*). This type of similarity is mainly useful when a repository of conceptual schemas is maintained, for classification of schemas according to similarity levels, and for their subsequent retrieval, through query and browsing facilities.
- Structure of elements within the schema they belong to (*element-based similarity*). This similarity is mainly useful when a deep comparison of schema contents is required, such as, for example, for view integration and reuse purposes. In this contexts, the user is interested in analyzing the structure (that is, properties) and the context (that is, the relationships) of the elements, in order to discover overlapping and similar characteristics, to be maintained in the integrated schema [21], or to be properly generalized in a proper reusable component [16].

4.1 Similarity based on descriptors

The descriptor-based similarity of a pair of schemas considers the affinity between descriptors associated with the schemas [8]. The greater the number of descriptors that present affinity, the higher the descriptor-based similarity between schemas. The metric we use to measure descriptor-based similarity between a pair of schemas is the Dice's function [22], which considers the number of descriptors presenting affinity in both schemas multiplied by two, over the total number of descriptors in each schema.

The descriptor-based similarity coefficient of a pair of schemas S_i and S_j, denoted by $DSim(S_i, S_j)$ is computed as follows:

$$DSim(S_i, S_j) = \frac{2 \mid (D(S_i) \cap D(S_j)) \mid}{\mid D(S_i) \mid + \mid D(S_j) \mid}$$

where $\mid X \mid$ indicates the cardinality of set X, and $D(S_i) \cap D(S_j)$ is the set composed of the pairs of descriptors of S_i and S_j which have affinity (see Sect. 3.1). Note that each descriptor of S_i and S_j can participate at most in one pair.

The Dice's function returns a similarity value in the range $[0, 1]$. The value 0 indicates absence of similarity, that is, no descriptors of S_i and S_j present affinity. The value 1 indicates identity, that is, all descriptors of S_i and S_j present affinity. Intermediate values indicate situations of more or less simialrity, depending on the number of descriptors. The more the descriptors with affinity, the greater the similarity coefficient. To determine the descriptor-based similarity coefficient, all descriptors of S_i and S_j are submitted to pairwise affinity comparisons (with the help of the Thesaurus), and their affinity coefficients are computed.

For example, let us consider the "Supply Management" schema, shown in Fig.1. We compute the descriptor-based similarity between the "Supply Management" schema and the portion of the "Courier Transportation" schema in Fig.2 containing the `Order`, `Invoice`, and `Customer` elements. Under the hypothesis that `Order`, `Invoice`, and `Customer` are all representative within the "Courier Transportation" schema portion, the descriptor-based similarity between the "Courier Transportation" and "Supply Management" schema portions is equal to 1. Indeed, all their descriptors have affinity, being equal (i.e., `Order` and `Order`), or synonyms (i.e., `Invoice` and `Payment`, `Customer` and `Client`). Under the hypothesis that only two elements are representative for the the `Order`, `Invoice`, and `Customer` schema portion, the descriptor-based similarity of the considered schema portions would be 0.8.

4.2 Similarity based on elements

The element-based similarity of a pair of schemas considers the affinity between the structure and the context of elements in the schemas [9]. The greater the number of elements that present affinity, the higher the element-based similarity between schemas. To compute the level of affinity between schema elements, we must take into account the structure of elements within a conceptual schema, and, in doing this, we must be as general as possible, in order to define affinity coefficients applicable to several conceptual models. For this purpose, we consider an element in a schema characterized by *structural properties*, corresponding to constructs used in conceptual models for describing real-world object properties; *behavioral*

properties, corresponding to constructs used in conceptual models for describing operations on real-world objects; and *links*, corresponding to constructs used in conceptual models for describing relationships between real-world objects. Affinity between elements is then computed by considering the affinity between their names (see Sect. 3.1, the number of their structural and behavioral properties which have affinity, and the number of their links which have affinity, using again the Dice's metric [22]. To measure the affinity between structural and behavioral properties, and links we define a set of *affinity coefficients*.

Let E_i and E_j two elements belonging to schemas S_i and S_j; let $SP(E_k)$ be the set of structural properties of element E_k; let $BP(E_k)$ be the set of behavioral properties of element E_k; let $L(E_k)$ be the set of links in which element E_k participates. The Structural Affinity coefficient, denoted by $SA(E_i, E_j)$, the Behavioral Affinity coefficient, denoted by $BA(E_i, E_j)$, and the Contextual Affinity coefficient, denoted by $CA(E_i, E_j)$, are computed as follows.

$$SA(E_i, E_j) = \frac{2 \mid SP(E_i \cap E_j) \mid}{\mid SP(E_i) \mid + \mid SP(E_j) \mid}$$

$$BA(E_i, E_j) = \frac{2 \mid BP(E_i \cap E_j) \mid}{\mid BP(E_i) \mid + \mid BP(E_j) \mid}$$

$$CA(E_i, E_j) = \frac{2 \mid L(E_i \cap E_j) \mid}{\mid L(E_i) \mid + \mid L(E_j) \mid}$$

where $\mid X \mid$ indicates the cardinality of set X, and $SP(E_i \cap E_j)$, $BP(E_i \cap E_j)$, and $L(E_i \cap E_j)$ indicate the pairs of structural properties, behavioral properties and links, respectively, which have affinity in both E_i and E_j. To determine the affinity between property and link names we use the name-based affinity coefficient introduced in Sect 3.1. Since properties are characterized also by domains, we have defined affinity conditions for primitive and complex domains, as described in [12]. In brief, two primitive domains have affinity if they are the same domain, or if one of them is included in the other; complex and reference domains affinity is determined on the basis of the affinity of the involved primitive domains, through normalization rules.

Each affinity coefficient can assume a value in the range $[0, 1]$. The value 0 represents absence of affinity (no affinities exist between E_i and E_j for that coefficient), while the value 1 represents a situation of identity (E_i and E_j are equally defined in both schemas, according to the coefficient considered). Intermediate values describe situations of greater or less affinity between E_i and E_j in their respective schemas.

For E_i and E_j the Global Affinity coefficient, $GA(E_i, E_j)$, is computed as the weighted sum of the affinity coefficients, that is,

$$GA(E_i, E_j) = \\ w_A Affinity(E_i, E_j) + w_{SA} SA(E_i, E_j) + w_{BA} BA(E_i, E_j) + w_{CA} CA(E_i, E_j)$$

where w_A, w_{SA}, w_{BA}, w_{CA}, with $w_A, w_{SA}, w_{BA}, w_{CA} \geq 0$, and $w_A + w_{SA} + w_{BA} + w_{CA} = 1$, are weights introduced to properly set the importance of each affinity coefficient within the global affinity.

Given a pair of schemas S_i and S_j, their elements are submitted to pairwise affinity comparisons. An affinity threshold T is defined, and only the element pairs having a Global Affinity greater than or equal to T are selected and considered to compute the element-based similarity of S_i and S_j.

Let $E(S_k)$ be the set of elements contained in schema S_k. The element-based similarity coefficient of a pair of schemas S_i and S_j, denoted by $ESim(S_i, S_j)$ is computed as follows:

$$ESim(S_i, S_j) = \frac{2 \mid (E(S_i) \cap E(S_j)) \mid}{\mid E(S_i) \mid + \mid E(S_j) \mid}$$

where $E(S_i) \cap E(S_j)$ is the set composed of the pairs of elements of S_i and S_j whose affinity is greater than or equal to the established threshold T. Note that each element of S_i and S_j can participate at most in one pair.

5 Experimentation

Reuse support

A methodology for creating reference components has been proposed as an example of application of the techniques discussed above in the context of reusability of conceptual level components. A reference component can be used as a starting point for developing new applications, similar to applications developed before, thus extending the reuse paradigm to the early development phases. To derive a reference representation, that is a schema fragment capable of describing similar representative elements in an integrated way, capability of recognizing similarities between components is required. Indeed, it frequently happens that the same concept is described in more than one schema, due to either the overlapping of requirements between schemas, or to different perspectives in the development

of schemas. The idea is that similar components in abstracted schemas should be represented into a single reusable reference component. For the definition of an integrated representation for a given reusable component view integration and restructuring techniques can be applied [2,18,5].

The methods and tools to support reuse have been developed by the authors within two ESPRIT Projects (ITHACA and F^3), and a CNR Project L.R.C. (Infokit). Research on reuse has focused both on storing reusable components in a repository and therefore providing support to retrieve them, and on building reusable conceptual components, from an in-depth analysis of schemas available from previous applications.

Two tools have been developed to support reuse:

- *RECAST* (Requirements Composition and Specification Tool), developed within the ITHACA and Infokit projects, supports application developers in builing object-oriented specifications of applications by composing reusable conceptual components. In RECAST, the techniques for the schema indexing described in Sect. 2.2 are applied; the features are used as a basis for retrieving interesting reusable components, giving their desired characteristics and performing fuzzy queries on a Software Information Base [6].
- *EXTRACT*, developed during the F^3 and Infokit projects, provides support to building reusable conceptual components (using an extended Entity-Relationship model) by classifying schemas and components to find closely related elements [5,11]. In EXTRACT, representative elements (see Sect. 2.1) are associated to schemas to describe them; schemas are then classified according to their similarity computed as in Sect. 4 in schema clusters. From each schema cluster, an analysis is performed on their elements to identify similar components, based on the element comparison techniques between representative elements illustrated in Sect. 4.2.

Analysis of large repositories

An application of techniques for indexing and analyzing schemas is now under study based on a large library of Entity-Relationships schemas in the Public Administration domain in collaboration with AIPA (Autorità per l'Informatica nella Pubblica Amministrazione) [13]. In this project, schemas are classified using representative elements, with the objective of supporting information flow reconstruction. An application of feature extraction and use is also being developed to provide support to navigation in the repository via imprecise queries.

6 Concluding remarks

In this paper we presented criteria and techniques to support schema analysis according to both syntactic and contextual aspects of the contained elements. Similarity criteria and metrics apt to compare schemas and schema elements for defining their degree of closeness have been defined. Clustering techniques to partition elements of a schema according to their level of closeness, and abstraction techniques to represent cluster information contents by means of abstract elements have been presented. These tools have been conceived to be used in the information system engineering and re-engineering processes [5,10,13]. The main challenge in this area is to develop and adopt better abstraction and standardization mechanisms for design and, possibly, reuse. Major motivations for abstraction and standardization are related to the necessity of reducing the development efforts and costs of system evolution by establishing sound reference frameworks. Since in these processes major difficulties are usually related to the requirements formalization efforts, our choice has been to provide criteria and techniques for analysing conceptual schemas and building a library of reference schemas availabe for reuse purposes or for validating schema evolution.

The proposed concepts and techniques together with the supporting tools have been experimented in the reuse area within the ESPRIT Project F^3 (From Fuzzy to Formal) in the "Transportation" domain. Furthermore, their application in the re-engineering area is being partly investigated in the framework of a project with the Information System Authority for Public Administration (AIPA) which has the task of planning and controlling the development of Information Systems by promoting their standardization and integration.

Acknowledgements

The authors wish to thank Chiara Francalanci for her collaboration on the definition of clustering techniques.
Part of this work has been supported by the ESPRIT III Project n. 6612 F^3 (From Fuzzy to Formal) and by Information System Authority for Public Administration (AIPA).

Appendix

Coefficient	Description	value
S	schema	
E_i, E_j	elements in a schema	
$W(E_i)$	quantity of information carried by an element E_i in a schema	
$A_{tot}(E_i)$	number of properties of E_i	
$L_{tot}(E_i)$	number of links of E_i to other elements in the schema	
T	threshold for selection of representative elements	
$w_{i,k}$	fuzzy weight of the k^{th} Feature with respect to the i^{th} element/schema	$[0,1]$
$\nu_{i,k}$	frequency of the k^{th} Feature in the i^{th} element/schema	
N	number of all the descriptors in the repository	
n_k	number of descriptors exhibiting Feature k	
F	total number of Features in the repository	
Affinity(E_i, E_j)	affinity value between elements E_i and E_j	
Synonym(t_h, t_k)	relationship between terms t_h and t_k in the Thesaurus	
Closeness (E_i, E_j)	closeness value between elements E_i and E_j	
w_k	weight associated with link type k	$[0,1]$
w_{rel}	weight associated with ER relationship links	0.4
w_{is-a}	weight associated with ER inheritance hierarchies	0.6
$n_k(E_i, E_j)$	number of links of type k relating E_i and E_j	
nl	total number of types of links	
C_x, C_y	clusters	
n_x, n_y	number of elements in a cluster	
Coupling(C_x, C_y)	measure of coupling between C_x and C_y	
w_A	weight to assess the importance of affinity in the computation of coupling	$[0,1]$ (here 0.1)
w_C	weight to assess the importance of closeness in the computation of coupling	$[0,1]$ (here 0.9)
$DSim(S_i, S_j)$	descriptor-based similarity coefficient of a pair of schemas S_i and S_j	$[0,1]$
$D(S_i)$	set of descriptors of schema S_i	
$SA(E_i, E_j)$	Structural Affinity coefficient	$[0,1]$
$BA(E_i, E_j)$	Behavioral Affinity coefficient	$[0,1]$
$CA(E_i, E_j)$	Contextual Affinity coefficient	$[0,1]$
$SP(E_k)$	set of structural properties of element E_k	
$BP(E_k)$	set of behavioral properties of element E_k	
$L(E_k)$	set of links in which element E_k participates	
$GA(E_i, E_j)$	Global Affinity coefficient	
w_{SA}, w_{BA}, w_{CA}	weights to assess the importance of affinity coefficients	$[0,1]$
$ESim(S_i, S_j)$	element-based similarity coefficient	$[0,1]$
$E(S_i)$	set of elements in a schema S_i	

References

[1] P. Aiken, A. Muntz, R. Richards, "DoD Legacy Systems - Reverse Engineering Data Requirements", *Communications of the ACM,* Vol.37, No.5, May 1994.

[2] C.Batini, M. Lenzerini, S. Navathe, "A Comprehensive Analysis of Methodologies for Database Schema Integration", *ACM Computing Surveys,* September 1986.

[3] C. Batini, G. Di Battista, G. Santucci, "Structuring Primitives for a Dictionary of Entity Relationship Data Schemas", *IEEE Transactions on Software Engineering,* Vol.19, No.4, April 1993.

[4] H.W. Beck, T. Anwar, S.B. Navathe, "A conceptual clustering algorithm for database schema design", *IEEE Trans. on Knowledge and Data Engineering,* Vol. 6, No. 3, June 1994.

[5] R. Bellinzona, S.Castano, V. De Antonellis, M.G. Fugini, B. Pernici "Requirements Reuse in the F^3 project", ISI Journal, Vol. 2, No. 6, 1994.

[6] R. Bellinzona, M.G. Fugini, B. Pernici, "Reusing specification in OO applications, *IEEE Software,* March 1995, in press.

[7] M. L. Brodie, M. Stonebraker, "DARWIN: On the Incremental Migration of Legacy Information Systems", *DOM Technical Report, TM- 0588-10-92-165,* GTE Laboratories Incorporated, November 1992.

[8] S.Castano, V. De Antonellis, B. Zonta, "Classifying and Reusing Conceptual Schemas", in *Proc. of ER'92, Int. Conf. on the Entity-Relationship Approach,* Karlsruhe, LNCS, n.645, Springer Verlag, October 1992.

[9] S. Castano, V. De Antonellis, "A Constructive Approach to Reuse of Conceptual Components, in *Proc. of 2nd ACM/IEEE Int. Workshop on Software Reusability,* Lucca, Italy, March 1993.

[10] S. Castano, V. De Antonellis "Standard-Driven Reengineering of Entity-Relationship Schemas", in *Proc. of ER'94 13th Int. Conf. on the Entity-Relationship Approach,* Manchester, UK, December 1994, Springer Verlag.

[11] S. Castano, V. De Antonellis "The F^3 Reuse Environment for Requirements Engineering", *ACM SIGSOFT Software Engineering Notes,* Vol.19, No.3, July 1994.

[12] S. Castano, V. De Antonellis, "Reuse in Object- Oriented Information Systems Development", in *Proc. of ISOOMS'94, Int. Symposium on Object- Oriented Methodolgy and Systems,* Palermo, Italy, September 1994, Springer Verlag.

[13] S. Castano, V. De Antonellis, B. Pernici "Building Reusable Conceptual Components in the Public Administration Domain", in Proc. of *SSR'95, ACM SIGSOFT Conference on Software Reuse,* Seattle, USA, April 1995.

[14] E. Damiani, M.G. Fugini, "Automatic thesaurus construction supporting fuzzy retrieval of reusable components", in *Proc. ACM SIG-APP Conf. on Applied Computing (SAC95),* Nashville, February 1995.

[15] V. De Antonellis, S. Castano, L.Vandoni, "Building reusable Components Through Project Evolution Analysis", *Information Systems,* Vol.19, No.3, 1994.

[16] F^3 Consortium, "F^3 Reference Manual", *ESPRIT Project Report,* December 1994.

[17] P. Feldman, D. Miller, "Entity Model Clustering: Structuring a Data Model by Abstraction," *The Computer Journal,* Vol.29, No.4, 1986.

[18] C. Francalanci, B. Pernici, "View Integration: a Survey of Current Developments", *Technical Report,* Politecnico di Milano, September 1993.

[19] C. Francalanci, B. Pernici, "Abstraction levels for entity-relationship schemas", in *Proc. of 13th Int. Conference on the Entity-Relationship Approach (ER '94),* Manchester, UK, December 1994.

[20] G.J. Klir, T.A. Floger, *Fuzzy sets, Uncertainty, and Information,* Prentice Hall, Englewood Cliffs, 1988.

[21] P. Johannesson, "Schema Standardization as an Aid in View Integration," *Information Systems,* Vol.19, No.3, April 1994.

[22] Y.S. Maarek, D.M. Berry, G.E. Kaiser, "An Information Retrieval Approach For Automatically Constructing Software Li-

braries", *IEEE Transactions on Software Engineering*, Vol.17, No.8, August 1991.

[23] N.A. MAIDEN, A.G. SUTCLIFFE, "Exploiting Reusable Specifications Through Analogy," *Communications of the ACM*, Vol.35, N.4, April 1992.

[24] G. SALTON, *Automatic Text Processing - The Transformation, Analysis and Retrieval of Information by Computer*, Addison-Wesley, 1989.

[25] G. SPANOUDAKIS, P. CONSTANTOPOULOS, "Similarity for Analogical Software Reuse: A Conceptual Modelling Approach", in *Proc. of CAiSE '93, Int. Conf. on Advanced Information Systems Engineering*, Paris, June 1993.

[26] T. J. TEOREY, G. WEI, D. L. BOLTON, J. A. KOENIG, "ER Model Clustering as an Aid for User Communication and Documentation in Database Design", *Communications of the ACM*, Vol. 3, N. 8, 1989.