ELECTRONIC WORKSHOPS IN COMPUTING

Series edited by Professor C.J. van Rijsbergen

Rainer Manthey and Viacheslav Wolfengagen (Eds)

Advances in Databases and Information Systems 1997

Proceedings of the First East-European Symposium on Advances in Databases and Information Systems, (ADBIS'97), St Petersburg, 2-5 September 1997

Reusing Analysis Schemas in ODB Applications: a Chart Based Approach

E. Damiani, A. Formica, M.G. Fugini, M. Missikoff, and R. Pizzicannella

Published in collaboration with the
British Computer Society

Reusing Analysis Schemas in ODB Applications: a Chart Based Approach

E. Damiani
Università di Milano
Milano, Italy

M.G. Fugini
Politecnico di Milano
Milano, Italy

A. Formica, M. Missikoff, R. Pizzicannella
IASI-CNR
Roma, Italy

Abstract

This paper presents a method for creating, indexing and reusing analysis schemas in developing Object-oriented Data-base (ODB) applications. Analysis schemas are specified by using *analysis charts*, a user-oriented set of forms structured according to the TQL^{++} Object-oriented specification model, and are classified according to their structural characteristics and content. A set of analysis charts forms a reusable schema, referred to as an *analysis stack*. The developer can retrieve and examine stacks by accessing analysis charts containing relevant entity names and structures. Charts are connected by links reproducing TQL^{++} relationships and connecting 'similar' schemas. The paper presents the measures of similarity between charts and describes the organization of charts in a *reuse repository*. A Thesaurus of relevant terms and synonyms is coupled with the repository. The Thesaurus and the repository are the basis for guiding developers in deriving new ODB applications through a sequence of steps proposed by a *CHarting and Analysis for Reuse Tool (CHART)*. The methodology for reusing analysis schemas, based on navigation in the repository, and the support tool are described.

1 Introduction

Reuse of requirement analysis schemas is receiving attention as a preliminary step in reuse-based application development. It is recognized, in fact, that reuse of specification and design elements can improve the reusability of code [9].

This paper addresses Object-oriented Database (ODB) applications and proposes a method and a tool for the development under reuse of analysis schemas of such applications. The underlying ODB specification model is drawn from TQL^{++} [7], a conceptual language aimed at specifying static and dynamic aspects of complex database applications in an Object-oriented way. In this paper, only static aspects of ODB applications are considered.

The approach consists in supporting the developer in the construction of the application schema, which is organized in a reuse repository [1] under the form of a set of analysis documents called *analysis charts*. Analysis charts, that correspond to schema entities, can be obtained by reusing the specifications of previously completed ODB applications, developed in the same application domain. Engineering of applications for inclusion in the repository is another phase of the proposed approach, which is aimed at creating specific annotations to support the reuse. Here we present a set of analysis criteria based on 'similarity' that are used in schema engineering as well as in chart inspection for reuse. The set of related charts describing a schema is called *analysis stack* that corresponds to a whole reusable TQL^{++} application schema. The idea of having different hierarchical levels of analysis is exploited, analogously to what proposed for instance in [2].

The developer is supported in the reuse by a method and an associated tool. He (or she) can browse the stacks and charts via appropriate links to find in the repository schemas which are similar to the requirements of a new application. The tool, called *CHarting and Analysis for Reuse Tool (CHART)*, allows the developer to inspect the stacks and the individual charts of the repository starting from a Thesaurus of terms. Such terms are entity names, property names and synonyms that have been stored during previous application developments, when analysis stacks and charts have been constructed.

The approach and the associated tool rely on some previous studies conducted by our group. In particular, it relies on a technique for semi-automatic Thesaurus construction [6], and on analysis of Entity-Relationship schemas for information system re-engineering, performed by Politecnico di Milano [3] for the Italian Authority for Informatics in the Public Administration (AIPA). Moreover, the specification language TQL^{++} has been proposed by IASI in [7] and experimented on several industrial ODB development cases, out of which a significant one is a system for railway traffic control [11].

The paper is organized as follows. After briefly reviewing the concepts of TQL^{++} (Section 1.1), a technique for representing TQL^{++} schemas in the form of analysis stacks and charts and for computing similarity indexes between schema entities is presented (Section 2). Then, the methodology for ODB application development under reuse is illustrated and an example is given (Section 3). Finally, the architecture of CHART is described (Section 4).

1.1 Elements of the ODB analysis model

In this section, the ODB analysis model, employed in this paper, is briefly presented. It is based on the language TQL^{++} for conceptual modeling of ODB applications. In particular, TQL^{++} is the language on which Mosaico, an environment for the specification and rapid prototyping of ODB applications, is based [12, 13]. It is characterized by all the main features of the Object-oriented paradigm (such as typing, inheritance, object identity, message passing), but avoids all the technical details found in existing development and programming languages. In this paper, we concentrate on the Object Model of TQL^{++} , i.e., the component aimed at modeling the structural aspects of an ODB application. Such an Object Model is compliant with the ODMG-93 [4], a standard for ODBs that is gaining a wide consensus within the database community.

In the Object Model of TQL^{++} , there is a clear separation between the intensional and extensional components. The intension of a database application (i.e., the conceptual model) is represented by a TQL^{++} schema in the form of a collection of types. Types, which are the fundamental elements of a TQL^{++} conceptual model, contain a description of the structure of the objects that will populate the extension of the database (i.e., the actual data). A set of types forming a TQL^{++} schema is shown below.

```
vehicle:= [type:(Wheels,Tracks,Water),
           color:{string}, production_date: integer ]
person:= [name:string, age:(0..150),
          phone:{integer}, address:[street:string, city:string]]
employee:= ISA person
           [salary:string, age:(18..65), drives:car]
car:= ISA vehicle
      [maker:(VW,Fiat,Saab),
       max_speed:integer, owner:person]
```

A type is composed by a label (t_term) and a tuple of typed properties. Types are organized according to a generalization hierarchy, declared by means of the ISA construct. ISA allows the definition of a type in terms of one or more supertypes (multiple inheritance is allowed), from which typed properties are inherited. Properties have a label (p_term) and can be typed in different ways, by using:

1. literal types, such as *integer* or *string*;
2. explicit sets of values, given either by enumeration (in the example, *car.maker*) or specifying interval extremes (as in *person.age*);
3. type labels, as in *car.owner*, establishing an explicit link (or association) between two types;

4. nested tuples, such as in *person.address*. In this case, a structured property is defined, organized as a tuple of subproperties, each of which is typed according to the four listed alternatives.

Properties may be either attributes or associations (relationships, according to ODMG-93). The former are typed according to cases (1) and (2) above, while the latter according to case (3). Associations can form cycles, hence resulting in recursive types. A property is assumed single-valued, unless curly brackets are used. Curly brackets indicate multi-valued properties, on which it is possible to impose cardinality constraints. In absence of cardinality constraints, any number of distinct values is acceptable (as in *person.phone*). In the following definition, the formal syntax of the TQL^{++} Object Model is presented: non-terminal symbols are closed between angular parenthesis, while terminal symbols are in bold. Symbols in italics represent user-defined strings.

Definition. Syntax of the TQL^{++} Object Model.

```

<type> ::= t_term := <type-definition>
<type-definition> ::= <body>
                        | ISA t_term ... t_term <body>
<body> ::= t_term
           | <literal> | ( <value_set> )
           | [ <tp> , ... , <tp> ]
<literal> ::= integer
            | real | boolean | string | TOP
<tp> ::= p_term : { <body> } m,M
    
```

The literal type **TOP** represents the most general type, and $m, M \in N_0$, $m \leq M$.
The formal definition of a TQL^{++} schema follows.

Definition. TQL^{++} schema.

A finite set of types is a TQL^{++} schema (schema for short) iff it results that:

- there are no dangling type labels: every *t_term* declared in a type definition (i.e., the right hand side of the type) is defined in the schema;
- every *t_term* is uniquely defined, i.e., the same *t_term* is not associated with more than one type definition;
- inheritance is acyclic, i.e., a type has not itself as supertype.

As anticipated, TQL^{++} , which is endowed with a rigorous denotational semantics, represents the formal basis on which charts and stacks have been conceived. A deeper analysis of the formal aspects falls outside the scope of this paper. The next sections will concentrate on the user-friendly, reuse oriented version of TQL^{++} , based on charts and stacks.

2 Analysis Stacks and Charts

In this section, analysis charts and stacks are presented, and their characterization in terms of entities and links, based on ‘similarity’ criteria, are given.

An *analysis chart*, in our approach, is a document giving a form-based view of the characteristics of a TQL^{++} entity in terms of its attributes and structure (association and hierarchical links to other entities of the schema). Whenever an ODB application is concluded, the charts produced in the analysis phase are inserted in the repository for reuse purposes.

An *analysis stack* corresponds to a TQL^{++} schema, containing therefore all the information about the schema of an ODB application. Charts in a stack are linked via TQL^{++} links (associations and ISA, described

in Section 1.1) which are here called *design links* because they are defined during the design of an application. Design links can be followed in browsing the repository, when designing a new application, with the aim of finding reusable analysis elements.

Various stacks of charts, i.e., various reusable schemas, corresponding to various ODB applications previously developed, are then stored in the repository.

Charts and stacks are linked via *similarity links* connecting both similar charts and similar stacks. A chart C_i is connected by a similarity link to the charts C_j that exhibit the highest values of similarity with C_i according to a threshold, e.g., a percentile-based threshold such as the top ten percent. This is obtained by computing the similarity indexes described in Section 2.3 between each chart C_i and all the other charts C_j in the repository. An analogous technique is used to set the similarity links among stacks, by selecting the ‘most connected’ stacks according to the similarity of their component charts. Similarity links between charts and stacks are statically computed applying the techniques described in Section 2.3, and dynamically updated upon a user (analyst) modification of the repository.

According to what described so far, the repository appears as a web of stacks variously connected via similarity links. Similarity links provide a *horizontal dimension* of navigation in the repository, horthogonal to the design links, ISA and associations, which exist within a given stack. A sketch of the repository organization is shown in Fig.1, where stacks, charts and links are depicted.

By following the design links among charts of one stack, and the similarity links among charts of different stacks, CHART iteratively proposes stacks and charts to the developer, supporting the design of a new schema, by reusing as many existing “similar” schemas, or schema fragments, as possible.

A Thesaurus exists for the repository as a navigation support tool containing entity terms, and synonyms. Charts of interest can be “marked” (see the asterisks in Fig.1) in order to re-start a search from a selected set of interesting charts.

In the following, we describe charts, stacks and the Thesaurus. Furthermore, the techniques for computing the similarity between entities and schemas are given.

2.1 Charts and Stacks

A chart is a document describing an entity of the ODB application. In a chart, the description of the entity is given according to a TQL^{++} type. In particular, a chart contains:

- the name of the entity;
- a set of attributes with their domains (typings);
- a set of design links (associations and hierarchies).

The name of the entity corresponds to a TQL^{++} type label, represented by a t_term . The attributes, according to the TQL^{++} Object Model, are properties typed with literals, explicit set of values, or intervals. The design links may be either ISA links or reference links (associations). The former are related to the hierarchy of the schema and are defined by the names of the entities that are generalization or specialization of the entity described in the chart (i.e., they correspond to the TQL^{++} supertypes and subtypes). The latter concern the TQL^{++} associations. In particular, a reference link is defined by a property label (a TQL^{++} association) and an entity name. According to the TQL^{++} Object Model, multi-valued properties and cardinality constraints can be specified in a chart.

In a chart, it is also possible to specify comments, i.e., a short text about the entity, that can be used for documentation purposes. Finally, a chart is linked to a *Thesaurus Entry*, as illustrated below. Before presenting how a Thesaurus Entry is organized, the notion of “stack of charts” is introduced.

Definition. A *stack of charts* is a set of charts describing a TQL^{++} schema, i.e. satisfying the following conditions:

Fig.1: Charts, Stacks, and Links in the Repository (grey arrows represent design links, black arrows similarity links)

- there are no dangling design links, i.e., for each entity name appearing in the design links of a chart there exists a chart in the stack describing such entity;
- different charts describe different entities, with different names and contents;
- ISA links are acyclic, i.e., there are no charts describing entities that are generalization or specialization of themselves.

2.2 Thesaurus

The criteria for establishing similarity links between charts and stacks, illustrated in the next subsection, are based on a Thesaurus of terms. The Thesaurus is organized according to a set of *Thesaurus Entries*. In particular, a Thesaurus Entry is defined by:

- a term t , that is the *key* of the entry;
- a set of *Equivalent Terms* (ET) to t , i.e., terms having the same meaning, for instance, acronym terms and their expansions. Singular terms and their plural, or gender variations are dealt with by means of a stemming and spelling procedure;
- a set of *Synonym Terms* (ST) of t , i.e., terms denoting the same entities;
- a set of *Terms in Hierarchy* (HT) with t , i.e., terms denoting entities participating in generalization or specialization hierarchies with the entity denoted by t ;
- a set of *Related Terms* (RT) to t , i.e., terms having the same roots;
- a set of *User-Defined Similar Terms* (UT) of t , specified by the user or the developer of a new application.

The Thesaurus is coupled with the repository in that the terms collected in it concern the application domain of the stacks stored in the repository. In particular, for each entity name and property (attribute or associations) present in a stack of the repository, there exists a Thesaurus Entry whose key term is the entity or the property name, respectively. The vice versa is not necessarily true, i.e., the Thesaurus may contain additional Entries whose key terms do not appear in the charts of the repository. Furthermore, in the Thesaurus the relationship among Synonym Terms is symmetric, i.e., if a synonym term t' of a key term t is a key in its turn, then t is a synonym term in the Entry of the key term t' . Finally, the Terms in Hierarchy specified in the Entries whose key terms are entity names of a stack include the entity names connected in the stack through ISA links.

The Thesaurus is accessed by a *Thesaurus Manager* integrated by a *String Processor*. Through the stemming facilities and the spelling checker provided by the String Processor, the Thesaurus Manager is able to assign to any unordered pair of terms t_i and t_j a weight $T(t_i, t_j)$ as follows. If t_i and t_j belong to a Thesaurus Entry, then the weight is assigned according to the kind of relationship existing between them, as specified in the following table:

Rel. t_i, t_j	$T(t_i, t_j)$
ET	1
ST	0.8
HT	0.5
RT	0.5
UT	user-defined

where the user-defined weight can range in the closed interval $[0..1]$. In all the other cases, the weight assigned to the pair of terms t_i and t_j is 0. The values of these predefined constant weights derive from experimental evaluations conducted on a large set of conceptual schemas relating to Information Systems in the Italian Public Administration [3]. In the next subsection, we will show how these weights are employed to compute the similarity indexes between charts and stacks of charts.

2.3 Similarity Indexes

The analysis of the similarity between charts and stacks of charts is performed on the basis of the Thesaurus and a set of criteria illustrated below.

1. Similarity between charts.

The similarity between charts is evaluated according to the similarity of the names of the entities they describe and their design links. Taking into account the design links, for each chart of the repository a *Content Vector* is created, consisting in a set of ordered pairs of terms. In particular, for each ISA link and reference link of the chart, an ordered pair $\langle r_i, e_i \rangle$ is extracted as follows. In case of an ISA link, r_i is the keyword *Sup* (or *Sub*), while e_i is the name of the entity that is a generalization (or specialization) of the entity described in the chart. In case of a reference link, r_i is the name of the association, while e_i is the name of the related entity.

Therefore, the similarity index between charts is established on the basis of the names of the entities they describe together with the Content Vectors so far introduced. In particular, let c_i and c_j be two charts belonging to different stacks, and e_i, e_j the names of the entities described, respectively. Furthermore, let $\langle r_{ih}, e_{ih} \rangle, h = 1 \dots n, \langle r_{jk}, e_{jk} \rangle, k = 1 \dots m$, be the Content Vectors associated with c_i and c_j , respectively. Then, the similarity index between the charts c_i and c_j , $som(c_i, c_j)$, is evaluated according to the following formula:

$$som(c_i, c_j) = \frac{1}{2}(T(e_i, e_j) + 2 \sum_{h,k} \frac{T(r_{ih}, r_{jk})T(e_{ih}, e_{jk})}{n+m})$$

where $T(t_i, t_j)$ is the weight assigned to the pair of terms t_i and t_j according to the Thesaurus, as explained in the previous subsection. Note that, in case of the keywords *Sup* and *Sub*, the value of $T(Sup, Sub)$ is assumed to be 0.5, on the basis of the weight assigned to the terms that are in hierarchy (see HT in the previous table). Furthermore, in the above formula the product of the weights appears (rather than the sum), since in our model the similarity between the elements of the Content Vectors (ordered pairs of terms) depends on the similarity of both the first components *and* the second components. If at least one component has similarity index 0, this must be also the similarity index between the pairs.

2. Similarity between stacks.

The similarity between stacks is computed starting from the similarity among all the charts they contain. In particular, in case of stacks that contain a large number of charts, instead of all the charts, only a subset of them is considered, referred to as the set of the *representative* charts of the stack. Intuitively, the representative charts of a stack are the most relevant for describing the content of the stack. Given a stack, the representative charts are selected on the basis of the Content Vectors associated with them. In particular, they correspond to the charts whose Content Vectors have maximum cardinality, and the number of representative charts to select is established by the user, for instance, according to a percentile-based criteria analogous to the one described in Section 2. The motivation is that the number of design links of a chart provides a heuristic measure of the importance of that chart in the stack (i.e., the relevance of an entity in a schema). This means that we interpret a high connection degree of the chart as a high degree of representativeness of the chart in the application.

As already mentioned, the evaluation of the similarity index between stacks is computed starting from the similarity indexes among their charts. In particular, consider two stacks S_p, S_q , containing the (possibly representative) charts $c_{pk}, k = 1 \dots n, c_{qh}, h = 1 \dots m$, respectively. Then, the similarity index between S_p and S_q is computed according to the following formula:

$$sim(S_p, S_q) = 2 \sum_{h,k} \frac{som(c_{pk}, c_{qh})}{n+m}$$

Similarity links between stacks in the repository are once again established considering the maximum values of sim .

3 Reuse Methodology

Until now, we have described the indexes associated with reusable analysis charts and stacks and the method for similarity computation. This section illustrates how these indexes are employed for the design of ODB schemas under reuse, based on charts and stacks. The methodology emphasizes reuse of schemas stored in a repository during previous ODB development processes. The repository is coupled with a Thesaurus that provides support for accessing analysis charts and associated links. Below, the methodology that allows the design of a new application under reuse of previous analysis results is illustrated.

3.1 Objectives

The objectives of the methodology are:

- to facilitate ODB development by providing a reuse repository, containing reusable analysis schemas which are a starting point for new schemas;
- to improve the quality of the analysis specification, by making “well-defined” and already tested analysis schemas available as a possible starting point;
- to improve the amount of work, i.e., the productivity of the analysis phase, by reducing the need of specifying new charts, shortening therefore the analysis and specification effort.

A collection of reusable schemas (stacks), gathered in a repository, indexed as shown in the previous sections, is an effective means to facilitate the ODB analysis. In fact, developers can start from an organized set of reusable schemas, and proceed by selecting, tailoring, and adapting suitable parts for the new ODB application.

Issues that have been considered in the methodology for schema reuse are:

- The capability of tracing the relationships between analysis charts through a link-based mechanism, following an hypertextual based approach [5]. This provides the developer with a model-driven support for navigating across analysis charts.
- The support for querying and browsing analysis charts and stacks using techniques conceived for software reuse. Different techniques, such as IR [8], faceted schemas [15], knowledge-based systems [14] have been proposed for component retrieval, based on similarity criteria to compute the conceptual distance between the target component, and the candidate components in the repository. IR techniques seem suitable also for higher-level artefacts, such as analysis documentation [10], or requirements specifications [2, 5]. Our ODB analysis methodology under reuse integrates a selection of these techniques, all aiming at improving the specification and validation of an analysis schema.

The hypothesis of the methodology are:

- Annotated schemas are available in the repository, integrated with the information of the Thesaurus: to be reusable, a schema should be self-explicative, with the possibility of being selected, understood, and then tailored and adapted to specific contexts. Accordingly, we define a *reusable schema* RS_i in the repository as a $Stack_i$ and its associated *reuse annotations* containing the knowledge for selecting and tailoring (parts of) RS_i according to the current requirements. $Stack_i$ is composed of the complete set of linked analysis charts specifying an application; *reuse annotations* contain the guidelines associated with RS_i providing the required schema customization knowledge in form of:

- selectable entities described by analysis charts; these are sets of entities connected to a given entity E_i and which become active, or selectable, to be reused with E_i to avoid dangling references (e.g., the ancestors of E_i) or due to reuse histories (e.g., in previous applications $E_i = \text{CAR}$ has been reused with PERSON and DRIVER which are now signaled as selectable).
- keywords associated with the schema RS_i and stored in the Thesaurus;
- comments, in the form of annotated text included by the developer, such as a set of advises and comments coming from other developments, to be considered when reusing a given chart.

In Fig.2, a sample stack of a “CarRental” application is shown; it consists of three analysis charts. Links are shown by arrows; the labels of ISA links denote the direction to a *supertype* or to a *subtype*.

- Focus on an IR-based approach for reusable schema retrieval, with associated capabilities of querying and browsing the repository. We assume to deal with a schema as a document, thanks to its associated analysis charts, and to apply IR techniques for indexing and retrieving schemas and hypertextual techniques to navigate along links. Of course, a schema can not be considered a true corpus of document, and has strong syntactic rules (e.g., no repetition of terms should occur, unless in syntactically controlled positions); therefore, IR techniques have to be tailored to such kind of documents.
- Availability of a Thesaurus. In general, we assume to work within a given application domain (e.g., office, department stores, banking, education), and we analyze schemas in that domain. Since an IR approach is adopted, we need to define and maintain, for each domain, the terms frequently used, their synonyms, homonyms, similar terms, and other relationships between terms (such as standard BT/NT and

RT [16], or user-defined relationships) to assure correct understanding of the schema contents. We refer to a Thesaurus as the base for all this information, which includes, for the representative entities, the Content Vectors. Furthermore, we assume an initialization and incremental semi-automatic tuning of the Thesaurus, along with the indexing and storing of new schemas in the repository.

3.2 Steps

The proposed methodology for ODB application designing by reuse consists of the steps outlined below. The rationale of this methodology is to seamlessly integrate a *query mode*, where the system is asked for the best reuse candidates, according to some metrics, with a *navigational mode*, where the user follows hypertextual links in order to explore the repository and select reusable components.

The steps are given here in terms of user interaction with the CHART support tool. The objective is to find reusable charts to be used in the design of a new schema.

The repository can be queried at two levels: *entity level* and *content level*. The former requires that the analyst starts by formulating an *Application Lexicon*, essentially a set of relevant entity names, as explained below. The latter requires that the analyst specifies more accurately the application requirements, in the form of a set of ordered pairs, whose elements denote design links and related entities, respectively. We will refer to such a set as a *Requirement Vector*, to highlight the correspondence with the Content Vector associated with the charts in the repository. Both interaction levels are described below.

Entity level.

1. *Application Lexicon initialization*

The developer specifies a tentative *Application Lexicon*, i.e., a repertory of entity names that, in his opinion, characterize the new application. In so doing, the developer is guided by the Thesaurus holding the whole *problem domain lexicon*. Besides being queried for synonyms, the Thesaurus can be browsed by the user to find terms to be included in the new Application Lexicon. However, totally new names can also be specified. If a name specified by the user in the Application Lexicon is not present in the system lexicon, the Thesaurus asks the developer to select a synonym among the existing names or, alternatively, to create a new Entry.

2. *Multiple ranked list retrieval*

If the entered terms exist in the Thesaurus (i.e., the Application Lexicon is properly included in the system lexicon), the developer is proposed with the charts whose entity names are semantically closer to the terms in the Application Lexicon. Returned charts, that may or may not belong to a single stack, are ranked by the values of the semantical relationships of their names with the terms specified by the user. The system answer is thus a *multiple ranked list*. A set of user-controlled thresholds may be specified for filtering purposes.

3. *Granularity tuning*

If no semantically related charts are found for one or more terms specified by the user (i.e., the Application Lexicon is not properly included in the domain lexicon), the Thesaurus is invoked again to assist the user in finding hyperonyms (broader) and hyponyms (narrower terms) of the names in the Application Lexicon. This operation is the lexical counterpart of following the ISA links in the repository. So, even in this case, the analysis may proceed by varying the *granularity* of the search, looking for hierarchically related entities.

4. *Reuse candidate initialization*

In both cases (2) and (3), the developer is proposed with the analysis charts whose entity *names* are semantically related to the ones he specified. Sometimes, this can lead to the reuse of a whole stack, the one that contains most of the similar charts. In other cases, proposed charts belong to several different stacks. The developer can inspect the design links of each candidate, and choose among the charts

connected to it those best fitting the requirements of the new application. In this step, the developer is guided in *completing* each entity chart by observing the properties and associations of the ‘similar’ charts.

5. *Navigation along reuse candidates*

The developer can then enter the navigational mode, following the similarity links and reaching other stacks where, in turn, further reusable charts can be looked for by following the design links.

6. *Steps iteration*

The above steps are iterated until a satisfactory set of reusable analysis charts is obtained.

Content level.

In case of content level interactions, the analyst specifies a tentative list of pairs, forming a *Requirement Vector*. Each pair denotes a potential design link in the application schema, regardless of the entity that owns such a link.

1. *Requirement Vector specification*

A ranked vector of pairs, relevant for the application, is specified and submitted to CHART. Also in this phase, the role of the Thesaurus is central in supporting the analyst in the selection of the terms employed in the construction of the vector, much similarly to what described for the Application Lexicon, at point 1 of the entity level query mode.

2. *Stack retrieval*

The Thesaurus maintains a Content Vector for each stack, consisting of the union of the Content Vectors of the representative entities. The entered Requirement Vector is used to search the Content Vectors of the stored stacks. By applying the *Sim* function, the Thesaurus returns the stack with the highest similarity degree.

3. *Stack inspection*

Once the first candidate is identified, the analyst can retrieve it from the repository and access all its charts. Charts can be viewed by accessing first the most representative and then, using design links, by navigating from one to the next.

4. *Changing the search granularity*

If the Requirement Vector query is not successful, it means that the repository does not contain one specific stack with an acceptable similarity degree. In this situation, the analyst can switch from the stack to the chart granularity. Then, starting from the same Requirement Vector, the Thesaurus retrieves single charts across stacks. In summary, even if there is not a single stack with a significant chance of being reused, there may still be a number of charts, belonging to different stacks, that may be collected and inspected for reuse.

5. *Steps iteration*

The above steps are iterated until a satisfactory set of reusable analysis charts is obtained.

Example

An example is the following. A “BoatRental” application is being designed. Let us suppose that a repository of “Loan/ Rental” applications is available as a thematic catalogue. However, no “Boat” Loan/Rental stack is present and the “Boat” term does not exist in the Thesaurus. The analyst browses the Thesaurus and finds “Goods for Rental” as a generalization, and then a “CarRental” as a specialized term to which a stack corresponds. Such stack is selected for reuse. In particular, the analysis chart “Person” has a structure that can be reused with limited effort to design the client of boat rentals (see the chart of “Person” in Fig.2).

Moreover, suppose that in “CarRental” the “Car” chart contains a design link to the structure of a “RentOffice” as shown in Fig.3. The “RentOffice” chart and, in particular, its descendant “HireContractor” are judged by the developer as potentially useful for the current application and are therefore incorporated in the current schema.

Some Charts of CarRental Application

Fig.3: Design links of a Car Rental stack

“HireContractor” carries along its design links to other charts (not shown in the figure) of the “CarRental” stack and its similarity links to charts of other stacks, for instance the “VideotapesLoan” stack. The analyst can choose to enter the navigational mode, to explore the repository both vertically (via design links) and horizontally (via similarity links). He can decide to further inspect the charts of “CarRental” or “VideotapesLoan” to check whether attributes, or associations exist which are a useful hint to complete his “BoatRental” schema.

4 Architecture of the Charting and Analysis for Reuse Tool

The overall architecture of the CHarting and Analysis for Reuse Tool (CHART) is depicted in Fig.4.

The basic modules composing CHART are the following:

- *Description Editor*

This Thesaurus-aware module is a lexical processor that allows the user to list some entity names to be used in the current project, while taking into account the set of keywords, i.e. the lexicon already available in the system. The list of keywords created by the user, referred to as *Application Lexicon*, is in fact a query to the system and can be edited and submitted interactively in the usual process of user-controlled query refinement.

At a more refined level, the analyst creates a Requirement Vector, constructed as described in the previous section. The Application Lexicon first, and then the Requirement Vector, are used to identify candidate stacks and charts for analysis reuse.

- *Thesaurus Manager*

This module is in charge of using the list of terms submitted by the user to access the repository for retrieving the charts that seem to be the best candidates for reuse. This module performs reuse candidates selection by exploiting the semantic relationships’ values stored in the Thesaurus. This process may lead to the retrieval of a whole candidate stack or, more frequently, of several charts taken from different stacks in the repository.

Fig.4: CHarting and Analysis for Reuse Tool Architecture

Search optimizations, based on Thesaurus content, take place in this module. In order to avoid the search on (potentially) all charts in the repository, the system maintains full information only for representative charts of each stack. For example, given a chart name, or a Requirement Vector, the system first considers charts in the repository that are marked as representative and have a Content Vector in the Thesaurus.

- *Chart Editor/Browser*

This editor initially contains a visual representation of the result of a query submitted via the *Description Editor* module, being it a complete stack or several charts coming from different stacks. Thanks to a graphical interface, the user can modify these charts at will, setting their mutual relations and their attributes while consulting the Thesaurus when appropriate. Moreover, the user can follow similarity links in the repository to navigate among possible reuse candidates. Potentially useful charts can be interactively selected and copied to the *Chart Editor* local working area. At any time, it is possible to switch back from this navigational mode to the querying environment of the *Description Editor*.

- *Repository Search Interface*

The actual Stack and Chart Repository can be implemented using several database engines, including relational and full-text retrieval systems (the latter being particularly well-suited for optical memory distributions of the system). The Repository Search Interface decouples CHART from the underlying retrieval technology by presenting to the Thesaurus Interface the abstraction of a searchable indexed library, while the Chart Editor sees the related, but somehow different abstraction of a network of charts connected by similarity and design links, as outlined in previous sections. These two abstractions correspond to two different access techniques, namely the initial positioning by query submission and the subsequent hypertextual navigation.

- *CHART/TQL⁺⁺ Environment Interface*

This interface gives access from the analysis environment to the full set of the powerful syntax and semantic check facilities provided by the Mosaico-TQL⁺⁺ environment, including a just-in-time compiler

for rapid prototyping [13]. Checked charts, that constitute the *Current Stack* of the application being designed, are transformed into TQL^{++} code for checking and rapid prototyping of the application under development.

5 Discussion and Concluding Remarks

In the paper we have presented a technique for the development of ODB applications based on the reuse of analysis schemas. The technique, based on the TQL^{++} underlying strongly typed analysis model, consists in organizing a repository of analysis schemas of previously developed applications, to be reused in a new analysis. The repository can be inspected for selectable elements, which can further be modified and checked for consistency and completeness by the developer. The repository is composed of application schemas stored in the form of *stacks*, linked via similarity links that can be navigated to find also ‘non perfectly fitting’ schemas but good candidates for reuse. At a different granularity level, the repository provides analysis *charts*, that are the components of a stack and describe the schema entities. A link-based navigation is supported also at this level, in order to follow design links (generalizations/specializations, associations, etc.) within a stack, to retrieve related entities to be used altogether, and to follow similarity links among charts.

A methodology for reuse and the architecture of the CHART support tool have been outlined. Parts of this tool have been implemented and experimented such as the Thesaurus and the retrieval interface [6]. Alternative implementation strategies are being evaluated, integrating existing techniques and tools, such as the TeamConnection tool for application development and ObjectStore for the repository.

Acknowledgments

This work has been partially supported by the Italian National Research Council in the framework of “Progetto Coordinato: Progettazione di Sistemi Informativi” and “Progetto Strategico Informatica nella Pubblica Amministrazione”.

References

- [1] Banker R.D, Kauffman R.J., Zweig D., “Repository Evaluation of Software Reuse”, *IEEE Trans. on Soft. Eng.*, vol. 19, n. 4, April 1993.
- [2] Batory D., O'Malley S., “The Design and Implementation of Hierarchical Software Systems with Reusable Components”, *IEEE Trans. on Soft. Eng.*, vol. 18, n. 4, April 1992.
- [3] S. Castano, V. De Antonellis, M.G. Fugini, B. Pernici, “Techniques for the analysis of conceptual schemas”, *Proc. Annual Conf. on Advanced DataBase Applications (SEBD'95)*, Ravello, Italy, June 1995.
- [4] R.G.G.Cattel, “The Object Database Standard: ODMG-93”; Release 1.1, R.G.G.Cattel Ed., Morgan Kaufmann, San Francisco, 1994.
- [5] P. Constantopoulos, M. Doerr, Y. Vassiliou, “Repositories for Software Reuse: the Software Information Base”, *Proc. IFIP TC8/Wg8.4 Working Conf. on Information System Development Process*, Como, Italy, September 1993, in *Information Systems Design Process*, N. Prakash, C. Rolland, B. Pernici (Eds.), North-Holland, 1993.
- [6] Damiani E., Fugini M.G., “Automatic Thesaurus Construction Supporting Fuzzy Retrieval of Reusable Components”, *Proc. ACM SIG-APP Conf. on Applied Computing (SAC'95)*, Nashville, February 1995.
- [7] A. Formica, M. Missikoff, “Correctness of ISA Hierarchies in Object-Oriented Database Schemas”, *Proc. of International Conference EDBT'94*, Cambridge, March 1994, LNCS n. 779, Springer-Verlag, 1994.

- [8] Frakes, W.B., Pole, T.P., "An Empirical Study of Representation Methods for Reusable Software Components", *IEEE Trans. on Soft. Eng.*, vol. 20, n.8, August 1994.
- [9] Krueger C.W., "Software Reuse", *ACM Comp. Surveys*, vol.24, n.2, June 1992.
- [10] Maarek Y.S., Berry D.M., Kaiser G.E., "An Information Retrieval Approach for Automatically Constructing Software Libraries", *IEEE Trans. on Soft. Eng.*, vol.17, n.8, August 1991.
- [11] M.Missikoff, "An Object-Oriented Approach to an Information and Decision Support System for Railway Traffic Control", *Proc. of KES'97 Conference*, Adelaide (Australia), May, 1997.
- [12] M. Missikoff, R. Pizzicannella, "MOSAICO Version 4.0 - User Manual", *Technical Note*, December 1995.
- [13] M.Missikoff, M.Toiati, "Safe Rapid Prototyping of Object-Oriented Database Applications"; *Proc. of IEEE Int'l Workshop on Rapid System Prototyping*, Grenoble, 20-23 June, 1994.
- [14] Ostertag E., Hendler J., Prieto-Diaz R., Braun C., "Computing Similarity in a Reuse Library System: An AI-Based Approach", *ACM Trans. on Soft. Methods*, vol. 2, n.3, July 1993.
- [15] Prieto-Diaz R., Freeman P., "Classifying Software for Reusability", *IEEE Software*, vol.4, n.1, January 1987
- [16] Salton G., Allan J., Buckley C., "Automatic Structuring and Retrieval of Large Text Files", *Comm. of the ACM*, vol. 37, n.2, February 1994.