Retrieval of Similar Layouts in FABEL using AspecT

Joerg W Schaaf
Angi Voss

German National Research Center for Information Technology (GMD),
Artificial Intelligence Research Division,
Sankt Augustin
GERMANY

In the FABEL project [1] several approaches have been developed and published [2] for retrieving similar cases in the domain of architectural design. In this paper we want to focus on a short description and foremost on integration of these approaches. We will introduce the most important approaches. As a first step towards integration, we suggest decomposing them into their main parts: representation of cases, similarity function and retrieval component. Using the first two parts of each approach in a new search algorithm operating on a generic data structure, we built a shell called AspecT for an easy integration. We are positive that one can never have fully considered all aspects of a design. Therefore, we describe an open framework to define new aspects and to apply them depending on the context. Unfortunately, openness brings some difficulties along with it. One can not predict the importance of an aspect, until the situation assessment of the query [3] is completed. In other words, it is impossible to define a static distribution of aspect weights that fits for all purposes. As a consequence, one can not predefine a static structure on the case base to speed up retrieval processes. Dealing with this problem, we developed and published [4] a special search algorithm that passes through a multidimensional case base, finding the best fitting cases within a short amount of time. This algorithm does not need any predefined structure on the case base except aspect specific relations between cases. Creativity of inventing new aspects of cases should not be hindered but one can ask whether or not a certain aspect are worth being regarded in searching for useful cases. The shell AspecT offers a tool to evaluate aspects. At first, formal criteria influence the initial weights of aspects but later on, the contribution of an aspect to find user accepted cases, determines the survival fitness of an aspect. The rule for living or dying is simple. Seldom used aspects disappear, whereas others become stronger. A short description regarding this work is given in last part.

Keywords: case-based reasoning, retrieval, integration of similarity concepts, architectural design.

1 Some retrieval methods for an architectural domain

1.1 The domain: building design

One goal of the FABEL project is to support architects and civil engineers in designing buildings with complex technical installations. Originally, we focused on Prof. Haller's modular construction kit MIDI, his arrangement methodology armilla, and the A4/Dancer drawing tool developed for it ([5],[6],[7]). Now we are recognizing that our work should be applicable to other types of CAD layout plans as well. In A4/Dancer, a building is represented by up to a hundred thousand objects in a multidimensional space. Beside its position and extent in the three geometric dimensions, each object has semantical attributes spanning further dimensions. Most important are the subsystem (fresh air, used air, construction, ..) an object belongs to, its function or morphology (connection, equipment,

...), its resolution or degree of abstraction (zone, bounding box, concrete component), and its scale (building (= 2), floor (=4), room (=6), levels between floors and ceilings (= 7), ...).

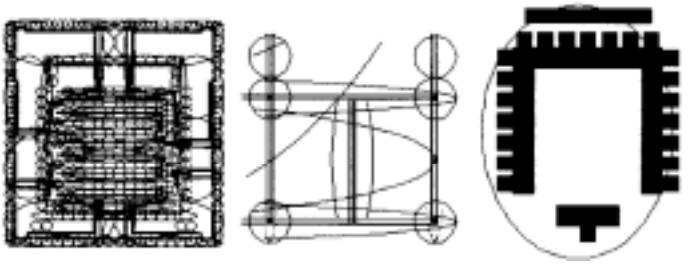

Figure 1: A layout fragment consists of objects with semantic and geometric attributes

Geometrical and semantical attributes allow to define many layers of a construction, e.g., the fresh-air system in a particular room on a particular floor. Each task has its own specific composition of layers. This composition changes with respect to the subproblems being addressed. We call such compositions layout fragments, or fragments for short. See Figure 1 for some examples.

1.2 The need for a retrieval tool

The retrieval of similar sketches and drawings can substantially support the design of buildings, in particular when many subsystems are to be designed and integrated. The retrieved layout fragments provide solutions whose adaptation may be easier than a solution from scratch, and they might even catch individual design styles. Such knowledge is very hard and often impossible to express in rules or any kind of context-independent knowledge.

1.3 Some ways to interpret and compare layout fragments

In layouts, spatial aspects play an eminent role. Thus, retrieval methods suited for data records or texts [8] and standard case-based reasoning methods [3] will not do. Nor will image recognition techniques [9], as we have more than mere images: our data representations already specify the objects presented in the images and their semantics, e.g., the subsystem an object belongs to, scale, or function. One of the major challenges in FABEL is to find retrieval methods focusing on spatial aspects while not ignoring the semantic information available - all this for huge CAD databases, and with response times of a few seconds. Since we are not aware of any single method satisfying these demands, we began exploring several alternatives in parallel. They differ in how they interpret and represent a layout fragment, in their notion of similarity, and in how they organize their case base (see [2] for more details Figure 3.

1.3.1 Attribute vectors with a linear case memory:

It is standard in case-based reasoning [3], to represent a case by an attribute vector. In our domain, a case is a layout, and its features are attribute-value pairs like the number of objects in a layout fragment, different joins of semantical attributes, or the width and height of the occupied area to characterize a layout fragment. The first layout fragment in Figure 2 has the attribute: no of objects: 696, subsystems: (used-air), functions: (equipment connection), resolutions: (zone bounding-box), scales: (4 6 7), width: 5040, height: 5280, flip horizontal: no, flip vertical: no. Two attribute vectors are compared by computing their distance, i.e. the weighted sum of differences in the attributes. The case memory is a simple list.

1.3.2 Keywords with an associative memory:

In a second approach using keywords, we count the number of objects with identical semantic attributes for subsystem, function, resolution, and scale. For the first fragment in Figure 2, which has 696 objects, 14 keywords are produced, among them (10 used-air equipment zone 4), (42 used-air connection zone 6), (200 used-air equipment zone

6), (196 used-air equipment bounding-box 6), (106 used-air connection bounding-box 6), (142 used-air connection bounding-box 7) The tuples are interpreted as mere keywords. The first one means that there is a floor (size 4) with 10 equipment zones for used air. In an associative memory, all keywords are directly linked to the cases they occur in. A query is translated into keywords that activate the case(s) most of them are linked to. So comparison is done on subsets of keywords.

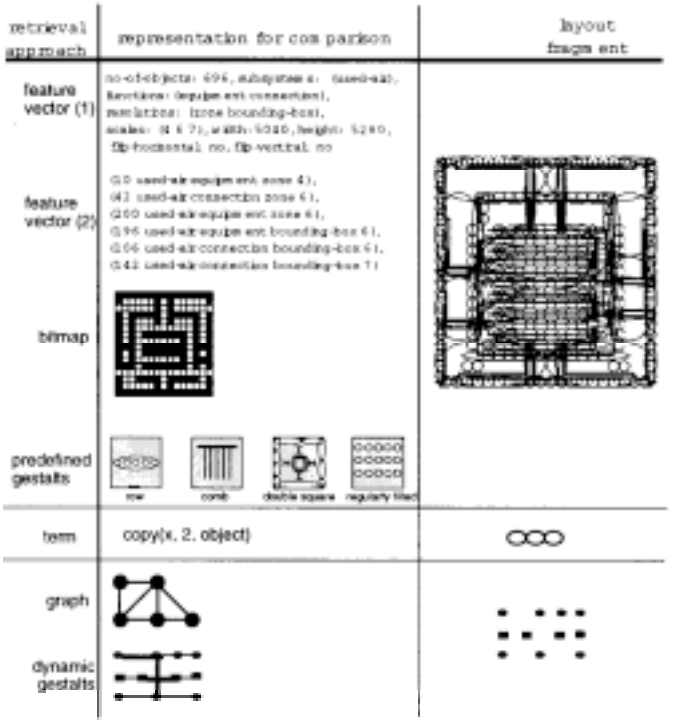

Figure 2: Some layout fragments: the supply and used air systems of a floor (1), a corner of them (2), and the furniture in a room (3).

1.3.3 Gestalts:

Layout fragments contain recurring patterns like a comb, a regular distribution, a rectangle, or a herring-bone. We identified a set of ten such gestalts to be used as additional attributes or keywords in the before mentioned approaches.

1.3.4 Silhouettes:

A case or layout is represented by a sequence of silhouettes of different granularity. The case memory is a hierarchy that is traversed top down, performing a bit pattern match at every node. Semantic attributes are mostly ignored. For these four approaches, a case needs to consist of a layout only. They are rather fast and useful for preselection. The following three approaches use cases distinguishing a problem and a solution layout. They are slower, because they do a thorough structural comparison that can be used for a subsequent adaptation.

1.3.5 Graphs:

Layouts are represented by graph whose edges specify elementary structural relations between the objects of the layout. For a set of layouts, similarity is defined by their most common subgraph. A special case memory has not yet been used. At Cebit 94, we presented a prototype combining the attribute vectors and keywords based on silhouettes and gestalts.

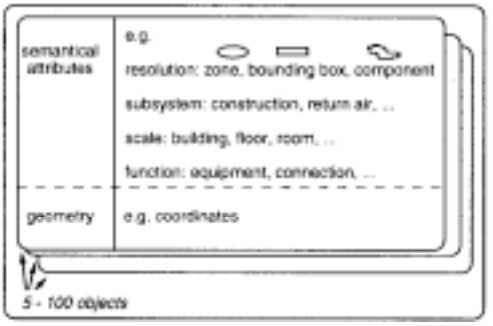

Figure 3: Some layout fragments and derived representations.

2 AspecT - a common roof for retrieval methods

We now present a shell called AspecT to integrate different retrieval methods by creating a common data structure for cases, building a network out of them and defining a general retrieval algorithm. Most retrieval methods (see above) contain at least three main parts. The first part deals with representation of original data of a case. This representation can be very simple; e.g. keywords (e.g. ASM in [2])), but others can be knowledge intensive and very time consuming (e.g. finding the case silhouette [10] or detection of gestalten in [11]). Usually, the generated representations are used to index cases for the support of further retrieval. A significant and context independent representation of a case shall be called an aspect in further description. Whether or not consideration of a certain aspect helps to distinguish useful cases from others, will not be discussed here. An approach to state whether or not considering an aspect leads to useful cases is described in section 2.5. The second main part of retrieval methods is a function which states similarity for two given cases in the previously generated representation. To simplify further description, it will help to replace the notion of similarity from this point on with the equivalent [12] notion of distance. The range of values for distance functions should be normalized to [0,1]. Therefore, whenever we speak about a distance, we mean a normed distance. The used distance functions are often explicitly given by the retrieval approaches but must sometimes be derived from the specific case organization (see associative memory in [13] and search tree of the silhouette approach in [10]). The third part of a retrieval method often consists of a specific case memory and search algorithm. Whenever retrieval methods consists of at least a representation method and a normalized distance function as described above, AspecT can integrate them and offer a common case base and a general and fast case retrieval. Additionally, AspecT offers the opportunity to regard different case representations (aspects) to various degrees. This means that one can watch the cases from different point of views [14] with respect to the actual task, by changing the weights of aspects on cases, while retrieving them from the case base.

2.1 Cases and the case base of AspecT

The case base of AspecT consists of cases connected with labeled edges. Cases contain original representation and derived aspects. Edges relate like aspects of different cases and quantify their distance regarding this aspect. In a pre calculation phase the representation part of retrieval methods is used to calculate every possible representation of a case. It is stored in a record belonging to the case. The second decomposition part of integrated retrieval methods calculates distances between pairs of representations.

For example, if the description of a pieces of a puzzle is a case, one could assume that color, texture and shape are important aspects to find similarity. We first determine color of each pieces and connect parts that are similar in this aspect. Then we do the same with texture and shape. Thus we have three networks describing the neighborhoods in the different aspects.

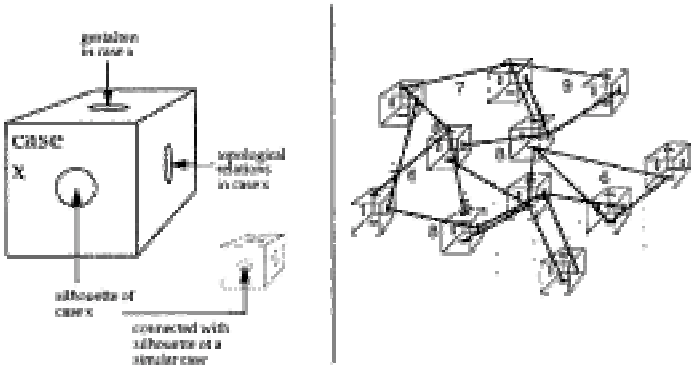

Figure 4: A case object, and a case base out of cases

AspecT should be used if a case base is worth being examined for several different types of queries, to fulfill different tasks and purposes or to serve users of different professions. If a case base is only used to answer very specialized questions regarding no more than one special case representation, approaches with a pre calculated structure may be more effective (e.g. kd-trees in [12] or associative structures in [13]). For our domain in FABEL, and probably more generally in complex design, it makes sense to use same cases to support different tasks in different work flows.

2.2 Case retrieval in AspecT

Given a scene from a CAD plan as a query, we must answer the question of how cases that fit the query in actually relevant aspects can be retrieved. When activated, every retrieval method from above suggests a set of similar cases that bases on comparisons in its regarded aspect. The first idea was to decide which aspect is most important in the actual situation and to accept suggestions calculated on that basis. Often this decision is not very easy because two or more aspects are equivalent according to their actual importance. As a solution, one could run the retrieval method regarding the most important aspect first and apply the second important method to the result. This is how it works in the FABELprototype at the moment. A general retrieval algorithm should take into consideration more than one aspect at a time. The idea is to state the importance of each represented aspect at query time and to influence the case base so that neighborhoods change according to that. Let's say that two puzzle pieces are very similar in color but have different textures. If the aspect color is stressed, they are strong neighbors; however, if texture is stressed this neighborhood disappears. We built a general retrieval algorithm called 'Fish and Sink (FaS)" for different representations and online assignment of importance using these changing neighborhoods. The basic idea of this algorithm is that close neighbors of bad fitting cases can not be expected to fit the query best; therefore, one should check others first Retrieval for similar cases should avoid testing every single case of the case base to enable the user to continue work with results within a few seconds. To still obtain cases of high quality, comparison of one case with the query should deliver more than a single evaluation. Approaches have been proposed to examine one node in a search tree and cut off branches that do not contain adequate solutions. (See e.g., kd-trees of [12], "granulation trees" of [10] etc.) These approaches do not work if users want to change importance of aspects online. Thus, case organizations can not be static and must depend on the users actual tasks and interests. Therefore, the dilemma is that without a structure on a case base, an efficient search is impossible. Unfortunately, the information necessary to structure the case base (actual weights of aspects) does not become available until the time of which the query is given.

Obviously we need a simple and fast way to spontaneously structure the case base. The main idea is not to define relations between cases, but to relate their smallest ingenious interpretations - their aspects - and to combine them spontaneously according to the actual task. The view specific structure on the case base is now a result of elementary operations on aspect specific pre organized bases. Let cases be decomposed in aspects and be related by edges connecting them as described above. Furthermore, let us suppose that the

importance of all aspects can be fixed at query time. The algorithm's aim is to test the probably useful cases before it tests those cases for which negative hints were found. A useless case's neighborhood should be seen as such a negative hint. This statement is based on the belief that for realistic case bases, the following is true: If a retrieved case is not able to support the user in a given situation, then any similar case would not support him either.

Consider the following metaphor (see Figure 5). Imagine all cases being distributed on a plane. You will find that they are without neighbors or within small or large groups. Imagine also, query (A) as a point in that plane. Fishing means taking a close look at one particular test case (T) Selected by catching a point in the plane. First, suppose a test case is selected randomly [15] . After having tested the test case, imagine sinking it according to its distance to the query. Test cases with a small distance to the query will remain closer to the surface, whereas those with greater distances will be sunk deeper. If a test case has neighbors in the area, these neighbors will be influenced by the depth to which the test case is sunk he closer the~ neighborhood is, the greater the influence will be. With each test of a case a crater appears around the location it was sunk. Cases that are part of a crater can not be test cases in the first pass anymore, but can be sunk further by influence of a test case close to them that is being sunk deeper. The closer a case is to the surface, the greater its distance is to every case that was sunk up to that time and the more rises the probability that this case will be tested early in the second pass.

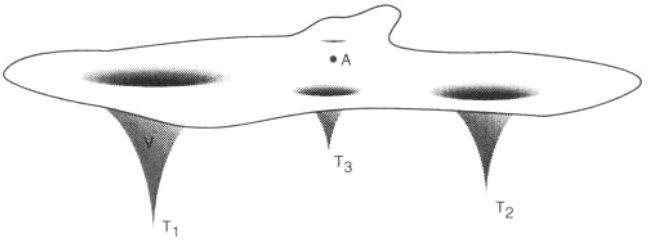

Figure 5: A pictural explanation of what happens in FaS

A formal description of that algorithm called "Fish and Sink", including theoretical background and complexity estimation is published in [16]. Fish and Sink can be activated expecting various results (see [16]), e.g.:

(a) To obtain a sorted list of cases, recommending the most similar cases first.
(b) To obtain the (n) best cases (sorted regarding their similarity to the query)
(c) To obtain all cases with a similarity greater than a predefined threshold.
(d) To obtain the best cases after a predefined run-time,
(e) To obtain the most promising cases after a spontanes user interruption.
(f) To obtain a sorted list of best cases up to the point where the quality of the next one is significantly worse than the previously recommended one.
(g) To obtain only one case of each group of cases where elements are very similar

To summarize: the algorithm Fish and Sink is based on consequent use of detected hints, pointing out that a particular case can no longer be expected to be similar to the query. The algorithm first tests those for which the fewest of such hints where found.

2.3 Context dependent weights for aspects

The presented integration of retrieval methods is only useful if weights of aspects were normally adapted to the actual context at the time the query was given. Actually the user must define them manually according to his needs. The following hints could be used to automatically adjust them according to the context.

(a) [general indicators:] - The costs for representation and storage, the support an aspect gives to distinguish between good and useless cases, the success of an aspect in terms of similarity between the query and the user selected cases regarding this aspect. (Sometimes combinations of hints are more powerful; for example, the ability to classify divided by the complexity to represent and compare.)

(b) [user specific hints:] - Success of an aspect in respect to the actual user, user specific default values e.g. based on the users profession, his access rights, his status etc.

(c) [task specific hints:] - Actual task. (Sometimes it might be difficult to detect which task is going to be performed, but it might be possible to make assumptions based e.g. on the type and number of objects or applied operations. - Previously completed tasks.

(d) [author defined hints]: - The author himself describes a set of hints that indicates application of his similarity concept.

(e) [dependent hints]: - It is possible to find hints to weight aspects in aspect representations which were generated earlier.

For example, if you find that objects in the query are of type supply air, then generate the aspect gestalt to detect the junction pattern. From these, we think the hints from user and task are the most variable and important ones. Automatically generated weights are dominated by any user interaction.

2.4 Adaptation of context dependence

In the following section, we want to deal with the interesting question of how to adapt weights for aspects in respect to the case selection of the user. We believe that any situation in which a user activates retrieval can be classified by some of the hints like these described above. Each class determines the aspects in which cases must be regarded. Our aim is to find the appropriate hints and to derive adequate prediction of aspect weights. The problem with deriving aspect weights from hints can be solved e.g. with artificial neuronal networks (ANN) and back propagation. Detectors that react upon hints can be seen as an input layer. This input results in activation of aspects. Degree of activation means weight for an aspect. Now the task is to find an appropriate distribution of aspect weights for each pattern of hints. This interdependence can be learned using the backpropagation algorithm ([17],[18],[19]). The backpropagation algorithm needs a target vector to train the ANN. A target vector reflects what would have been the correct result. Differences were propagated back to correct weights of edges that connect layers. But how do we obtain such a target vector? Naturally one can not expect to obtain it from the user. If users would have been able to deliver aspect weights, they could have defined them manually in the beginning. Therefore, we must derive the aspect weights in a different way.

The most important criteria for the correctness of an aspect weight is the role an aspect plays in retrieval success. A case is considered to be useful, if it is selected by the user to support his further work. User selection of a case is used to adapt aspect weights by correcting the interdependence between hints and aspects. Suppose, in a certain situation (with its specific hints) an aspects weight vector W(old) (output vector) is generated by the trained ANN. Based on these weights, the search algorithm Fish and Sink retrieves cases Fl, F2, .., Fi, .. Fn). The user selects the case Fi. It seems to make sense to adapt aspect weights in such a manner, that in a similar situation, case Fi is delivered earlier in the result list. One could change the aspect weight vector W(old) so that the sum of aspect similarities between query and case Fi become maximal. The hereby found target vector W(new) could be used to train the ANN, so that for situations with similar hints the aspect weights would lead to an earlier submission of Fi. To get the target vector, we use a simpler different method. It is enough to adapt the weights for each aspect to the detected similarities between query and Fi. This leads to a reinforcement of detected similarities and differences if the user made his decision regarding only these aspects that are represented in AspecT.We suggest the following formula for W(new)

$$w_i^{neu} = \frac{(K * w_i^{old}) + \dfrac{1 - \delta(A_i, F_i)}{\sum_{i=1}^{n} 1 - \delta(A_i, F_i)}}{K + 1}$$

Figure 6: Formula to get new aspects weights from user decisions

In other words: Given a situation, the weight for aspect (i) is set to be the normed aspect similarity between the query and (Fi). By normalizing (to Sum wi =1), weight moves from aspects with detected differences to those with detected similarities. The change in weights is slowed down by K-times addition of the old weight and division by (K+1). The distance between the query and (Fi) is expressed by delta(A,Fi). Similarity is defined by subtracting the distance from one.

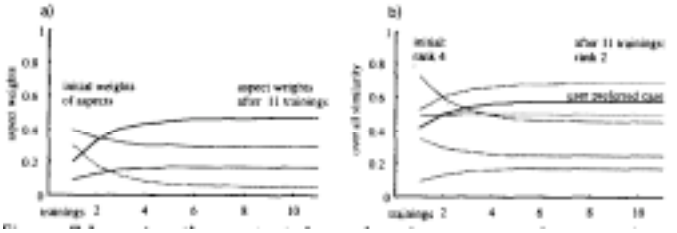

Figure / Learning the context-dependent importance of aspects from previous user decisions

Figure 7 shows how the rank of (Fi) increases (black curve, part b) because the weights of the aspects (part a) change. There is a second way to get a target vector. If the user manually defines aspect weights, we still force an automatic weightening. The differences between the result of the automatic process and the user specification, allows a further correction with backpropagation. To summarize; we use a two-step-learning to adapt weights for aspects to user selection. First, an ANN learns the dependencies between hints and weights of aspects using target vectors to correct its output. Second, the target vector itself is learned from the cases a user selects or from manually defined aspect weights.

2.5 Evaluation of aspects

The benefit of an aspect justifies its complexity of representation and comparison. Therefore, aspects without benefit should not be calculated. Evaluation for us, means to state the benefit of an aspect and to eliminate useless ones. Aspects are useful if they are used and if they help to find cases which are selected by users to support their work. We already reported the conditions for the use of aspects. Hints are associated with aspects by an ANN which is trained using the users case selection or preferences. The problem of evaluating aspects is now reduced to a question about the frequence of its use. Aspects which are not used, can disappear successively. This could happen step by step by disregarding the useless aspects for new cases, deleting them from older ones and finally erasing the edges between cases that are similar in these aspects. This automatic assessment gives experts the freedom to define new aspects, without having to fear that they will

Permanently obtain useless cases due to a wrong idea. Additionally there are conditions or usefulness of aspects which can be precalculated. A new aspect is useless, for example, if an aspect exists already that states the same similarity between each pair of cases of the case base. Such an aspect should be deactivated until a case with differences in this point of view is inserted to the case base.

2.6 Summary and comparison

The presented approach differs from others mainly in the following points:

(a) [Changing views of cases]: The idea of making case bases useful for several different users for performing many different tasks, has not yet been explored consequently as far as I know. Using different and freely definable views on cases makes it possible to use episodical knowledge for tasks that one did not think about while the case base was created. This makes it possible to hitherto existing users to get more information from their cases and ma make it possible for new user groups to participate. Some ideas regarding aspects can also be found in [21]. He uses decomposition of cases to aspects mainly to e storage capacity needs.

(b) [Spontaneous structuring of the case base]: To realize the idea of (changing views), cases of the case bases can not be sorted statically to support the retrieval. To solve the retrieval problem without a static case organization, the FaS algorithm has been developed. FaS creates an easy to calculate local structure around a randomly selected case, using only the predefined structure between aspects and a given function to calculate view distances. A similar relationship between cases is used in [21]. He connects entire cases according to similarity to perform his "Hillclimbing on case neighborhoods. [22] describe a method to find cases without searching. At a first glance it looks similar to the procedure performed by FaS, but there are big and far-reaching differences. (See the following item and [16])- [Using exclusion to retrieve]: Many approaches like [21] and [22]

try to make statements about the usefulness of cases that are not directly tested. This has a lot of disadvantages. Retrieval methods based on this are e.g. sensitive towards missing case neighbors. FaS on the other hand, uses indirect tests only to make statements about the uselessness of cases. Usefulness is only stated by direct tests. If an aspect neighbor is missing, no fact stating the uselessness is generated, so the case will be tested early in the process. The only consequence of a missing aspect can therefore be, that the overall run time increases. In [21] and [22] cases can only be found confidently if every neighborhood is accurately existent. Missing neighbors can lead to neglecting the best cases. In FaS, a missing neighbor can not lead to a loss of quality if the user is willing to accept a slight delay. Nevertheless, he should initiate calculation of every known aspect similarity between each pair of his cases.

(c) [A tool to evaluate aspects]: Organization, storage and retrieval of cases help to evaluate the importance of represented aspects. Importance can be derived by a reasoning about aspect weights that guide retrieval and aspect specific similarities between query and the user selected case. This reasoning changes the frequence of use of an aspect which is taken as an indicator for usefulness. The presented approach suggests a concept to deal with different aspects of cases and thereby, to integrate many of todays approaches that deliver similar cases for Case Based Reasoning. In the presented paper we focused on the problem of getting the importance of aspects regarding the actual task.

3 Outlook

The combination of such a variety of retrieval methods is unique in FABEL. This is due to the fact that the retrieval of similar technical drawings, in particular CAD-plans, is a new research area where appropriate methods still need to be developed. Such methods should detect relevant patterns or constellations of objects in a drawing. The approach to combine them context dependent weighted at query time, is completely worked out, formally described in [16] and implemented as a prototype. Creation and maintenance of case bases belong as well to the implementation as a data structure to contain case aspects, a data structure to connect similar aspect occurences and the search algorithm Fish and Sink. Up to now, we have integrated the aspect case silhouette [10] for a case base (in FABEL) in the domain of architectural design. Integration of the aspects "Gestalten" [11] and "topological structure" [23] is in progress. Run-time tests are still to be made. The presented approach has been criticized in the point that the Fish and Sink algorithm (which is essential for) could only work correctly, if the aspect specific distance functions satisfy the "Triangle Inequality". In [16] we show formally that the distance functions should rather satisfy an inequality that is derived from the triangle inequality. A method is described of how to state errors of distance functions that are confronted with this inequality and how to modify Fish and Sink so that these errors can be taken into consideration. As a master thesis [24] a student is actually working on the question of when cases become aspect neighbors. He is following up several approaches e.g., methods of detecting clusters. In the remaining years until June96, the FABEL project will support other subtasks of design like assessing a layout fragment, adapting it to another context, or giving strategic advice on what piece of the design to elaborate next. Wrt. adaptation, we are in contact with the CADRE project in Lausanne [25] that does dimensional adaptation of Autocad plans of buildings.

4 Acknowledgments

We gratefully acknowledge the work of our colleagues in the FABEL project that is presented the survey part of this article. Special thanks also Merinda Johnson who helped with the editing of this paper. Finally, we would like to thank Prof Dr M M Richter for his initial impulse to the work on AspecT and for several important hints.

5 References

[1] This research was supported by the German Ministry for Research and Technology (BMFT) within the joint project FABEL under contract no. 01IW104. Project partners in FABEL are German National Research Center for Computer Science

(GMD), Sankt Augustin, BSR Consulting GmbH, Muenchen, Technical University of Dresden, HTWK Leipzig, University of Freiburg, and University of Karlsruhe.

[2] A. Voss. Similarity concepts and retrieval methods. FABEL-Report -13, GMD, Sankt Augustin, 1994.

[3] J. L. Kolodner. Case-Based Reasoning. Morgan Kaufmann, San Mateo, 1993.

[4] J. W. Schaaf. Fish and Sink. An Anytime Algorithm to Retrieve Adequate Cases. In Lecture Notes in Computer Science. Springer, New York, 1995

[5] F. Haller. MIDI - Ein offenes System fuer mehrgeschossige Bauten mit integrierter Medieninstallation. USM Bausysteme Haller, Muensingen, 1974.

[6] F. Hailer. ARMILLA - ein Installationsmodell. IFIB, 1985.

[7] L Hovestadt. A4 - digitales bauen: Ein Modell fuer die weitgehende Computerunterstuetzung von Entwurf, Konstruktion und Betrieb von Gebaeuden. PhD thesis, Institut für industrielle Bauproduktion der Universitaet Karlsruhe, 1993.

[8] G. Saiton & M. McGill. Introduction to Modern Information Retrieval. McGraw-Hill, New York, 1983.

[9] H. Niemann. Pattern Analysis and Understanding, 2nd edition. Springer, Berlin, 1990.

[10] C. H. Coulon & R. Steffens. Comparing fragments by their images. In Angi Voss, editor, Similarity concepts and retrieval methods, pages 36-44. GMD, Sankt Augustin, 1994.

[11] J. W. Schaaf. Gestalts in CAD-Plans, Analysis of a Similarity Concept. In J. Gero and F. Sudweeks, editors, AI in Design 94, Kluwer Academic Publishers, pages 437446, Dordrecht, 1994.

[12] S. Wess, J. Paulokat, & K. D. Althoff, editors. Fallbasiertes Schliessen - Ein Ueberblick -, Band I: Grundlagen. Universitaet Kaiserslautern, 1992.

[13] W. Graether. Computing distances between attribute-value representations in an associative memory. In A. Voss, (ed), Similarity concepts and retrieval methods, pages 12-25. GMD, Sankt Augustin, 1994.

[14] The notion of 'views' is based on that used in database systems.

[15] In our implementation, we use the strategy to select the case with the most edges first.

[16] J. W. Schaaf. AspecT: Ueber die Suche nach situationsgerechten Faellen im Case Based Reasoning. Fabel-Report 31, GMD, Sankt Augustin, December 1995.

[17] P.J. Werbos. Beyond regression: New tools for prediction and analysis in the behavioral sciences. Master's thesis, Harvard University, 1974.

[18] D. E. Rumelhart, G.E . Hinton, & R. J. Williams. Learning internal representations by error propagation. In Parallel distributed processing, volume 1, pages 318-362, Cambridge, 1986. MIT press.

[19] P. D. Wasserman. Neural Computing, Theory and Practice. Van Nostrand Reinhold, New York, 1989.

[20] R. Deters. Multidimensional structuring of a case-memory. In Stefan Wess Brigitte Bartsch-Spoerl, Dietmar Janetzko, editor, Fallbasiertes Schliessen - Grundlagen und Anwendungen, pages 21-29, Universitaet Kaiserslautern, Germany, 1995. Zentrum fuer Lernende Systeme und Anwendungen, Fachbereich Informatik.

[21] K. Goos. Fallbasiertes Klassifizieren. Methoden, Integration und Evaluation. PhD thesis, Bayerische Julius-Maxiniilians-Universitaet, Wuerzburg, 1995.

[22] M. Lenz & H. D. Burkhard. Retrieval ohne Suche. In Stefan Wess, Brigitte Bartsch-Spoerl, Dietmar Janetzko, editor, Fallbasiertes Schliessen - Grundlagen und Anwendungen, pp 1-10, Universitaet Kaiserslautern, Germany, 1995. Zentrum fuer Lernende Systeme und Anwendungen, Fachbereich Informatik.

[23] C. H. Coulon. Automatic Indexing, Retrieval and Reuse of Topologies in Complex Designs. Accepted at the ISCCSE-95, 1995.

[24] R. Steffens. Kantenreduktion in AspecT-fallbasen. Master's thesis, Universitaet Bonn, 1995. Diplomarbeit in der Entstehung.

[25] K. Hua, I. Smith, & B. Faltings. Exploring case-based design: CADRE. Artificial Intelligence for Enginering Design, Analysis, and Manufacturing, 7(2): 135-144, 1993.