Available online at www.sciencedirect.com
ScienceDirect
Knowledge-Based Systems 19 (2006) 480–488
Knowledge-Based SYSTEMS
www.elsevier.com/locate/knosys

The importance of retrieval in creative design analogies ☆

Paulo Gomes *, Nuno Seco, Francisco C. Pereira, Paulo Paiva, Paulo Carreiro, José L. Ferreira, Carlos Bento

CISUC – Centro de Informática e Sistemas da Universidade de Coimbra, Departamento de Engenharia Informática – Polo II, Universidade de Coimbra, Portugal

Received 23 January 2006; accepted 15 April 2006
Available online 12 June 2006

Abstract

Analogy is an important reasoning process in creative design. It enables the generation of new design artifacts using ideas from semantically distant domains. Candidate selection is a crucial process in the generation of creative analogies. Without a good set of candidate sources, the success of subsequent phases can be compromised. Two main types of selection have been identified: semantics-based retrieval and structure-based retrieval. This paper presents an empirical study on the importance of the analogy retrieval strategy in the domain of software design. We argue that both types of selection are important, but they play different roles in the process.

Keywords: Creative design; Analogy; Case-based reasoning; Software design

1. Introduction

Creativity comprises at least four components [2]: process, product, person or entity, and situation. Within this model the creative product is the crucial component. Creativity has no meaning except in relation to the creative product. Thus an important issue in developing computational models of creativity is the evaluation of the creative product. Research done in this domain has come up with several methods used for generation of artifacts considered creative [3–7]. Some methods identified in creative reasoning are: cross-domain transfer of ideas (analogy), combination of ideas, and exploration and transformation of conceptual spaces. Most of these methods are used in creative design [8].

Creative design can be defined as a cognitive process that generates new classes of designs [4]. A cognitive process is considered creative if the product that it generates satisfies certain properties or attributes [1]. These properties determine the product's creativity, thus defining guidelines to the product's evaluator. We identify two main properties: novelty and usefulness.

The first thing that comes into mind when talking about creative design is novelty. This is a mandatory characteristic in any creative artifact. The creative product must be something different from what the evaluator knows or is expecting. Evaluation of a creative product has to do with the confrontation of two sets of knowledge. One is the information conveyed by the design product and the other is the knowledge that the evaluator possesses or uses in the evaluation. If the knowledge from the design product does not make part of the evaluator's knowledge set or is sufficiently different, then it can be said that the product is novel.

During the evaluation of the potential creative design product, the evaluator uses performance-measuring functions, in order to determine to which extent the design satisfies the problem requirements. These measuring functions make part of the evaluator's body of knowledge.

☆ This work was partially supported by POSI – Programa Operacional Sociedade de Informação of Fundação Portuguesa para a Ciência e Tecnologia and European Union FEDER, under Contract POSI/33399/SRI/2000, by program PRAXIS XXI. REBUILDER homepage is http://rebuilder.dei.uc.pt.
* Corresponding author.
E-mail address: pgomes@dei.uc.pt (P. Gomes).

0950-7051/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.knosys.2006.04.006

If minimal performance thresholds are met, then the design product is useful and solves the problem. This is another mandatory characteristic of creative design solutions. The creative design product must be appropriate and useful.

Analogical reasoning [9–12] is a widely used problem solving method. It consists in the transference of knowledge between different domains. This transference is based on similarities between past problems and the new problem. The transferred knowledge is then used to generate solutions for the new problem. The main benefit of analogical reasoning is its capability to transfer knowledge between different domains. This cross-domain transfer enables the generation of new creative designs, and can be considered as a creative process.

Analogical reasoning involves at least three phases: identifying candidates for analogy, mapping the candidate sources with the target, and transferring knowledge between source and target. The first phase is crucial for the generation of creative analogies, without a good set of candidate sources the subsequent phases maybe at stake. Two main retrieval strategies can be used: semantics-based retrieval and structure-based retrieval. In the next paragraphs we describe some of the most important works on analogical retrieval.

Gentner and Forbus [13] have developed the MAC/FAC (Many Are Called/Few Are Chosen) approach to analogy retrieval. It comprises a two-stage structural retrieval model, where in the MAC phase a crude idea of the case structural content is used to retrieve a broad range of cases from memory. Then, in the FAC phase, a complete structural similarity metric is used to eliminate several cases, and to rank the chosen ones.

The Holographic Reduced Representations approach [14] combines semantic and structural similarity. The semantic similarity is based on the common semantic primitives of objects, while the structural similarity is accessed using the various roles of each object. This structural similarity is only a superficial assessment of the structural similarity.

An approach using constraint satisfaction (ARCS – Analogical Retrieval by Constraint Satisfaction [12]) applies three similarity types through the creation of a constraint network. These types are: lexical similarity, structural similarity, and pragmatic similarity.

RADAR was developed by Crean and O'Donoghue [15] with the goal of retrieving semantically distant analogies in a computationally tractable manner. It is based on structural properties to retrieve candidate sources, and the main concern of this approach is that distant analogies would not be rejected by the semantic similarity.

In this paper, we present an approach to software design using analogy, which is integrated in a Computer Aided Software Engineering (CASE) tool named REBUILDER [16]. This tool is based on Case-Based Reasoning (CBR) [17] and comprises a Knowledge Base with several types of knowledge, including WordNet [18], a lexical resource used in REBUILDER as an ontology. We focus on the retrieval phase of analogical reasoning, by studying the influence of different candidate selection strategies in the novelty and utility of generated artifacts. We have used six combinations of semantic retrieval and structural retrieval, to study the adequacy of each strategy to analogical reasoning.

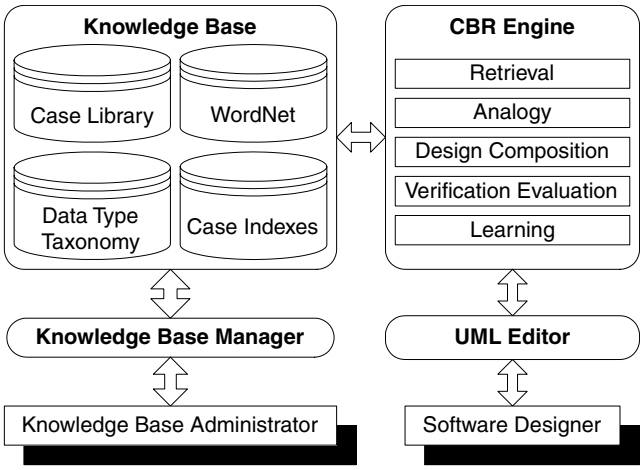

Fig. 1. REBUILDER's Architecture.

The next two sections describe REBUILDER and the used knowledge base, which enable the reader to understand the environment in which the analogical reasoning module is used. Then Section 4 describes how analogy is performed in REBUILDER, along with the different retrieval strategies. Section 5 describes the experiments performed and discusses the achieved results. Section 6 presents other approaches to analogy in design. Finally Section 7 concludes this paper.

2. REBUILDER

The approach carried out in REBUILDER uses CBR as the main reasoning framework, enabling the integration of several different types of knowledge. This section provides an overview on the system's architecture.

Fig. 1 shows the architecture of REBUILDER. It comprises four main modules: the UML[1] editor, the knowledge base manager, the knowledge base (KB), and the CBR engine. It also shows the two different user types: software designers, and KB administrators. Software designers use REBUILDER as a CASE tool, along with the system's reuse capabilities. A KB administrator is responsible for keeping the KB updated and consistent. The UML editor is the interface between REBUILDER and the software designer, while the KB manager is the interface between the KB administrator and the system.

The UML editor is the front-end of REBUILDER and the environment where the software designer develops designs. Apart from the usual editor commands to

[1] Unified Modelling Language [19].

manipulate UML objects, the editor integrates new commands capable of reusing design knowledge.

The KB Manager module is used by the administrator to manage the KB, keeping it consistent and updated. This module comprises all the functionalities of the UML editor, and adds case base management functions. These are used by the KB administrator to update and modify the KB.

The KB comprises four different parts: the case library, which stores the cases of previous software designs; an index memory used for efficient case retrieval; a data type taxonomy; and WordNet [18], which is a general purpose ontology. The KB is described in more detail in the next section.

The CBR Engine is the reasoning module of REBUILDER. As the name indicates, it uses CBR as the reasoning framework. This module comprises five different parts: Retrieval, Analogy, Design Composition, Verification, and Learning. The retrieval module searches the case library for designs or design objects similar to the query. The most similar ones are presented to the user, allowing the designer to reuse these designs or part of them. The analogy module maps designs from the case library, to the query design. The resulting mapping establishes the knowledge transfer from the old design to the query design. The composition module can be used to adapt a past design (or part of it) to the query using design composition. The verification module identifies and corrects errors in the design. The learning module acquires new knowledge from the user interaction, or from the system reasoning. In this paper we focus on the analogy module.

3. Knowledge base

The KB comprises: the case library, the WordNet ontology, the case indexes and the data type taxonomy. In REBUILDER, a case describes a software design, which is represented in UML through the use of class diagrams. Fig. 2 shows an example of a class diagram representing part of an educational system. Nodes are classes, with name, attributes and methods. Links represent relations between classes. Conceptually a case in REBUILDER comprises: a name used to identify the case within the case library; the main package, which is an object that comprises all the other case objects; and the file name where the case is stored. UML class diagram objects considered in REBUILDER are: packages, classes, interfaces and relations. A package is an UML object used for grouping other UML objects. A class describes an entity in UML and it corresponds to a concept described by attributes at a structural level, and by methods at a behavioral level. Interfaces have only methods, since they describe a protocol of communication for a specific class. A relation describes a relationship between two UML objects.

WordNet is used in REBUILDER as a general ontology. It uses a differential theory where concept meanings are represented by symbols that enable a theorist to distinguish

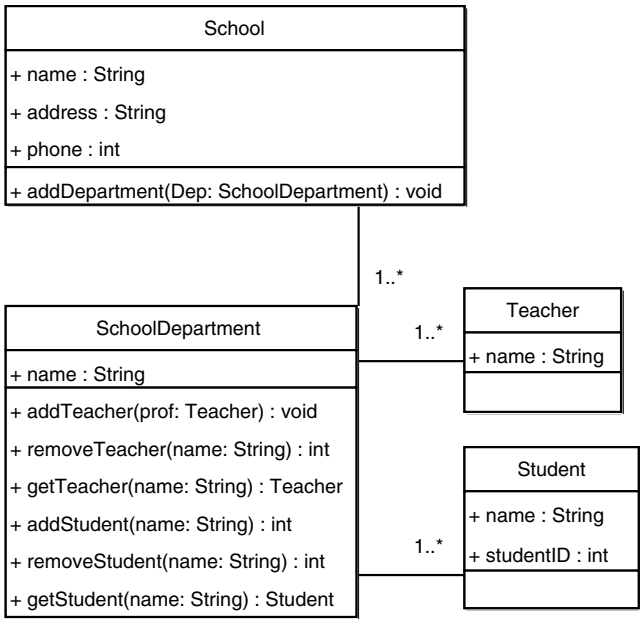

Fig. 2. Example of an UML class diagram (*Case1*), the package classification is *School*.

among them. Symbols are words, and concept meanings are called synsets. A synset is a concept represented by one or more words. If more than one word can be used to represent a synset, then they are called synonyms. The same word can have more than one different meaning (polysemy). WordNet is built around the concept of synset. Basically it comprises a list of word synsets, and different semantic relations between synsets. The first part is a list of words, each one with a list of synsets that the word represents. The second part, is a set of semantic relations between synsets, like *is-a* relations, *part-of* relations, and other relations. REBUILDER uses the word synset list and four semantic relations: *is-a*, *part-of*, *substance-of*, and *member-of*. Synsets are classified in four different types: nouns, verbs, adjectives, and adverbs. REBUILDER uses synsets for categorization of software objects. Each object has an associated synset, which represents the object meaning, called context synset. The object's context synset can be used for computing object similarity (using the WordNet semantic relations), or it can be used as a case index, allowing the rapid access to objects with the same classification.

Design cases can be large, so they are stored in files, which makes case access slower then if they were in memory. To solve this problem we use case indexes. These provide a way to access the relevant case parts without having to read all the case files from disk. Each object in a case is used as an index. REBUILDER uses the context synset of each object to index the case in WordNet. This way REBUILDER can retrieve a complete case, using the case root package, or it can retrieve only a subset of case objects, using the objects' indexes. This allows REBUILDER to provide the user the possibility to retrieve not only packages, but also classes and interfaces. To illustrate this

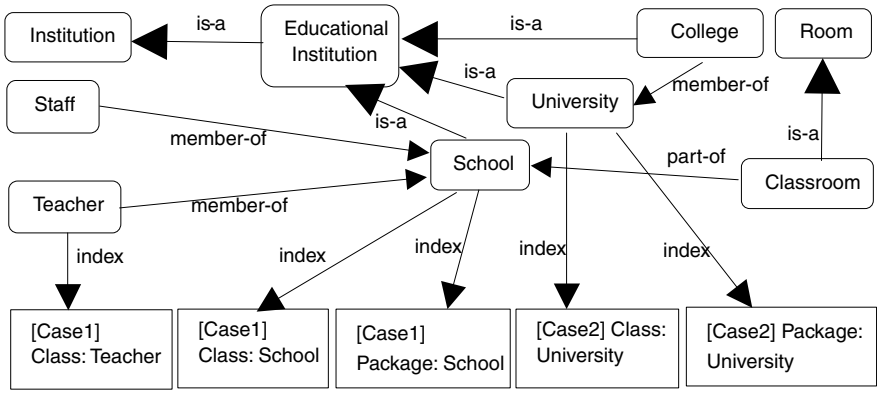

Fig. 3. A small example of the WordNet structure and case indexes.

approach, suppose that the class diagram of Fig. 2 represents *Case1*. Fig. 3 presents part of the WordNet structure and some of the case indexes associated with *Case1*. As can be seen, WordNet relations are of the types *is-a*, *part-of* and *member-of*, while the index relation relates a case object (squared boxes) with a WordNet synset (rounded boxes). For instance *Case1* has one package called *School* (the one presented in Fig. 2), which is indexed by synset *School*. It has also a class with the same name and categorization, indexed by the same synset, making also this class available for retrieval.

The data type taxonomy is a hierarchy of data types used in REBUILDER. Data types are used in the definition of attributes and parameters. The data taxonomy is used to compute the conceptual distance between two data types.

4. Analogical reasoning

Analogical reasoning is used in REBUILDER to suggest class diagrams to the designer. The analogy process has three steps:

- Identify candidate diagrams for analogy.
- Map each candidate diagrams with the target diagram.
- Create new diagrams, by knowledge transfer, between the candidate diagram and the target one.

4.1. Candidate selection

Cases are selected from the case library to be used as source diagrams. The selected candidates must be appropriate, otherwise the whole mapping phase can be at risk. Six alternative strategies for candidate selection can be used in REBUILDER. To better describe these strategies, candidate selection is decomposed in two subtasks: retrieval and ranking. While retrieval should be a fast comparison task yielding a subset of candidate cases, ranking is a more complex and computational demanding phase, where cases are ordered accordingly to a defined criteria.

Candidate retrieval returns a subset of cases from the case library. This should be a computational inexpensive task, with the main purpose of identifying a first set of cases that are possible candidates for analogy mapping with the target problem. Case retrieval is based on the WordNet structure, which is used as an indexing structure. The retrieval algorithm uses the classifications of the target problem object as the initial search probe in WordNet. For example, if the designer selects a package as the target problem, the retrieval algorithm uses the package's synset as the initial search probe. Then the algorithm checks if there are any packages indexes associated with the WordNet node of that synset. If there are enough indexes, the algorithm stops and returns them. Otherwise, it explores the synset nodes adjacent to the initial one, searching for package indexes until the number of found indexes reaches the number of objects that the user wants to be retrieved. Suppose that the N best objects are to be retrieved, *QObj* is the query object, and *ObjectList* is the universe of objects that can be retrieved (usually *ObjectList* comprises all the library cases), the retrieval algorithm is depicted in Fig. 4.

Candidate ranking receives a set of cases, which are then ranked based on a metric. This metric incorporates several criteria, which basically assesses semantic and structural similarity. Three different similarity metrics are used: one that combines semantic and structure evaluation, a second one that uses an independence measure, and another that evaluates structural properties.

The first similarity metric compares two UML class diagrams based on the diagram synset (the package's synset, which is a semantic evaluation) and diagram structure (combines structural evaluation of the diagram with object synset distances, is both structural and semantical), for a detailed description of this metric, see [16].

The independence measure is specific to UML and evaluates the independence of each object in the class diagram, based on the structural relations of diagram objects. Each class or interface in the class diagram is assigned a score that reflects the independence level of that object from other objects in the diagram, thus yielding a sense of importance of that object in the diagram.

$ObjsFound \leftarrow \varnothing$
$PSynset \leftarrow$ Get synset of $QObj$ (WSD process)
$PSynsets \leftarrow \{PSynset\}$
$SynsetsExplored \leftarrow \varnothing$
WHILE $(\#ObsFound < N)$ AND $(PSynsets \neq \varnothing)$ AND $(IterationNumber < MSL)$ DO
  $Synset \leftarrow$ Remove first element of $PSynsets$
  $SynsetsExplored \leftarrow SynsetsExplored + Synset$
  $SubSynsets \leftarrow$ Get $Synset$ hyponyms (children nodes)
  $SuperSynsets \leftarrow$ Get $Synset$ hypernyms (parent nodes)
  $SubSynsets \leftarrow SubSynsets - SynsetsExplored - PSynsets$
  $SuperSynsets \leftarrow SuperSynsets - SynsetsExplored - PSynsets$
  $PSynsets \leftarrow$ Add $SubSynsets$ to the end of $PSynsets$
  $PSynsets \leftarrow$ Add $SuperSynsets$ to the end of $PSynsets$
  $Objects \leftarrow$ Get objects indexed by $Synset$ and with the same type as $QObj$
  $Objects \leftarrow Objects \cap ObjectList$
  $ObjsFound \leftarrow ObjsFound \cup Objects$
ENDWHILE
$ObjsFound \leftarrow$ Rank $ObjsFound$ by similarity
RETURN the first N elements from $ObjsFound$

Fig. 4. The retrieval algorithm used in REBUILDER.

The third similarity metric is based on the structural properties of the diagrams being compared. This metric is based on the work of O'Donoghue [20]. The main idea of this metric is using the graph-like properties of diagrams to determine the structural similarity of two diagrams. O'Donoghue argues that structural similarity is more important for the identification of good analogical candidates than semantic similarity. But, because the most accurate way to assess structural similarity can only be done using structural mapping, which is computational expensive, he has used structural properties to assess the structural similarity, which are fast to assess [21].

The structural properties that we have used are:

- Number of loops in diagram.
- Average loop size.
- Number of classes.
- Number of interfaces.
- Number of generalizations.
- Number of realizations.
- Number of associations.
- Number of dependencies.
- Independence metric comparison.

Then we have devised four different candidate selection strategies (see also Table 1):

Table 1
The retrieval strategies implemented in REBUILDER

Strategy	Retrieval	Ranking
1	Semantic	Semantic and structural
2	Semantic	Structural (independence)
3	Semantic	Structural
4	No	Semantic and structural
5	No	Structural (independence)
6	No	Structural

Strategy 1: this strategy first uses the semantic retrieval to retrieve a set of cases, which are then ranked by the first similarity metric. This is the first implemented strategy in REBUILDER, but is also the more complex one.
Strategy 2: this strategy uses the semantic retrieval and then uses the independence measure to rank the retrieved cases.
Strategy 3: this strategy uses also the semantic retrieval, but then, retrieved cases are ranked using the third similarity metric.
Strategy 4: this strategy does not use a retrieval algorithm, instead all the cases are feed into the similarity metric, which in this strategy is the first similarity metric. This strategy is the most computational expensive one, it is used only for comparison with the other strategies.
Strategy 5: this strategy is similar to strategy 4 where no retrieval algorithm is used. The ranking is then performed by the independence measure.
Strategy 6: this strategy is similar to the previous one, in the sense that it does not use a retrieval algorithm. It only ranks all the cases in the case library using the third similarity metric. This strategy is also implemented to be compared with the other ones.

4.2. Mapping process

The second step of analogy is the mapping of each candidate diagram to the query diagram, yielding an object list correspondence for each candidate. This phase relies on two alternative algorithms: one based on relation mapping, and the other on object mapping, but both return a list of mappings between objects.

The relation-based algorithm (see Fig. 5) uses the UML relations to establish the object mappings. It starts the mapping selecting a relation from the query diagram, based on an UML heuristic (independence measure), which selects the relation that connects the two most important

Relations ← Get best relation from the query diagram based on the independence measure
MappingList ← ∅
WHILE *Relations* ≠ ∅ DO
    PRelation ← Get best relation from *Relations* based on the independence measure
    CRelation ← Get best matching relation from base case relations that are matching candidates (structural constraints must be met)
    Mapping ← Get the mapping between objects, based on the mapping between *PRelation* and *CRelation*
    Remove *PRelation* from *Relations*
    Add *Mapping* to *MappingList*
    Add to *Relations* all the same type relations adjacent to *PRelations*, which are not already mapped (if *PRelation* connects *A* and *B* then the adjacent relations of *PRelations* are the relations in which *A* or *B* are part of, excluding *PRelation*)
Return *MappingList*

Fig. 5. The relation-based mapping algorithm.

diagram objects. Then it tries to find a matching relation on the candidate diagram. After finding a match, it starts the mapping by the neighbor relations, spreading the mapping using the diagram relations. This algorithm maps objects in pairs corresponding to the relation's objects.

The object-based algorithm (see Fig. 6) starts the mapping selecting the most independent query object, based on the UML independence heuristic. After finding the corresponding candidate object, it tries to map the neighbor objects of the query object, taking the object's relations as constraints in the mapping.

Both algorithms satisfy the structural constraints defined by the UML diagram relations. Most of the resulting mappings do not map all the problem objects, so the mappings are ranked by number of objects mapped (see [22]). An important issue in the mapping stage is: which objects to map? Most of the time, there are several candidate objects for mapping with the problem object. In order to solve this issue, we have developed a metric that is used to choose the mapping candidate. Because we have two mapping algorithms, one based on relations and another on objects, there are two metrics: one for objects, and another for relations. These metrics are based on the WordNet distance between the object's synsets, and the relative position of these synsets in relation to the most specific common abstraction concept (for details on these metrics see [22]).

4.3. Knowledge transfer

The last step is the generation of new diagrams using the established mappings. For each mapping the analogy module creates a new diagram, which is a copy of the query diagram. Then, using the mappings between the query objects and the candidate objects, the algorithm transfers knowledge from the candidate diagram to the new diagram. This transfer has two steps: first there is an internal object transfer, and then an external object transfer. In the internal object transfer, the mapped query object gets all the attributes and methods from the candidate object that were not in the query object. This way, the query object is completed by the internal knowledge of the candidate object. The second step transfers neighbor objects and relations from the mapped candidate objects to the query objects. This transfers new objects and relations to the new diagram, expanding it.

5. Experiments

In this section, we present the results obtained from our experiments. The aim of these experiments is to study the correlation between the analogical retrieval strategies and the creative properties of the generated diagrams in the software design domain. The creative properties studied are usefulness and novelty.

Objects ←Get object from the query diagram with the highest independence measure
MappingList ← ∅
WHILE *Objects* ≠ ∅ DO
    PObject ← Get best object from *Objects*, based on the object's independence measure
    CObject ← Get best matching object from the base case objects that are matching candidates (structural constraints must be met)
    Mapping ← Get the mapping between *PObject* and *CObject*
    Remove *PObject* from *Objects*
    Add *Mapping* to *MappingList*
    Add to *Objects* all the objects adjacent to *PObject* which are not already mapped (and adjacent object *B* to an object *A*, are all the objects that have a relation with *A*).
Return *MappingList*

Fig. 6. The object-based mapping algorithm.

5.1. Setup

The results presented in this paper were obtained using a case library compromising 60 cases, each case has one package with 5–20 objects (there is a total number of 586 objects). These cases are from four different domains: banking information systems, health information systems, educational institution information systems, and store information systems (grocery stores, video stores, and others). A set of 25 UML class diagrams were used as problems during these experiments, each problem compromises, on average, four objects. The problems used in these experiments also belong to the four domains formerly presented.

5.2. Method

During these experiments we used the six candidate selection strategies presented in Table 1 for choosing the most promising candidates for reuse through analogy. When semantic retrieval is used as part of one of these strategies, only 10 (the first 10 cases found by our retrieval algorithm) cases are used for ranking, otherwise the whole case library is used. After the ranking phase, the most promising candidate is submitted to the analogy module. Afterwards the output of the analogy module (an UML class diagram) is evaluated for both usefulness and novelty. For these experiments we use the object-based mapping algorithm, since this algorithm has shown better results in previous work [22].

Novelty is assessed using the similarity metric used in case retrieval. We assess the novelty of a solution (the output of the analogy module) using the similarity with the cases in the case base. Three different similarity values are computed using the generated solution:

N1: the average similarity between the generated solution and the cases in the case library.
N2: the similarity value between the solution and the most similar case in the case library.
N3: the similarity value between the solution and the source case chosen for establishing the analogy with the target case.

Since the similarity metric, returns values between 0 and 1 where 1 is an identical match, we are interested in situations where this value is low. The results of this evaluation is presented in Table 2.

Usefulness is evaluated through human judgement of the generated diagrams. Two human judges evaluated these diagrams, identifying:

U1: the percentage of target objects that were mapped to a source case in which the knowledge transferred (methods and attributes from the source case to the target case) was incorrect or useless.
U2: the percentage of target objects that were mapped to a source case in which the knowledge transferred (methods and attributes from the source case to the target case) was partially incorrect.
U3: the number of objects that were transferred into the target diagram and that were considered incorrect or useless.

These values are presented in Table 3. We are aware of the subjectivity involved in these experiments, especially in the judgement of the usefulness. We think that the only faithful evaluation is through a case study in a real software production environment, where the tool is judged by its users. Despite this fact we will present our evaluation in the next section.

5.3. Results

In Tables 2 and 3, we present some of the values computed for our analysis of novelty and usefulness by analogy retrieval strategy.

As can be seen in Table 2 the average distance of the solution in relation to the rest of the case library is practically the same. This observation can result from the fact that the cases in the library are very sparse. So, whatever the solution, on average it will tend to be dissimilar to the case library. Observing the value of similarity between the solution and the most similar case (N2) we can see that strategy 1 and 4 present the highest values, which makes sense, since this metric contains a semantic component. In respect to N3, strategy 5 and 6 obtain solutions that are dissimilar to the source case used for establishing analogies. These values are expected, taken in consideration that neither of these strategies use semantics during the retrieval or ranking phases, and that the similarity metric does. Thus,

Table 2
Experimental results obtained for solution novelty

Strategy	Similarity with the case base (N1)	Similarity with best case (N2)	Similarity with source case (N3)
1	0.277	0.693	0.693
2	0.275	0.654	0.544
3	0.275	0.655	0.505
4	0.277	0.690	0.689
5	0.276	0.646	0.432
6	0.274	0.649	0.304

Table 3
Experimental results obtained for solution usefulness

Strategy	Incorrect mapped objects (U1) (%)	Partially incorrect mapped objects (U2) (%)	Number of Incorrect transferred objects (U3)
1	14	1	0.600
2	14	3	1.640
3	15	40	1.400
4	9	2	0.600
5	20	20	3.560
6	21	36	2.720

these strategies are able to generate more novel solutions than the other strategies, which are by some way using semantics to retrieve and rank source cases.

Table 3 presents the values obtained during the evaluation of the usefulness of the generated solution. We remind the reader that these values represent situations that were considered wrong or useless, this means that lower values correspond to solutions that are more useful since these require less corrections from the software designer. As we can observe strategies 5 and 6 seem to generate the diagrams which require the most correction. These high values may be due to the fact that the knowledge transference scheme implemented simply transfers objects from the source domain into the target domain without transforming the source object into an analog in the new target domain. If these source objects are transferred into completely different domains they will usually be considered incorrect.

These results suggest that semantics should be considered during the retrieval phase in order to obtain the most useful diagrams, that is, diagrams that require fewer corrections. On the other hand, novel analogies tend to occur when the ranking phase is based on structural properties. Using a retrieval strategy that combines both aspects can be a good option for analogy retrieval. Examples of this are strategies 2 and 5, which are able to maintain a compromise between strategies 1 and 4, which are more accurate but less aimed to generate novel solutions, and strategies 3 and 6 that are capable of generating more novel solutions but with an increase in the number of errors in the class diagrams.

6. Related work

There are several research works that use analogy to generate designs. From these we have selected the ones that are similar to our work. Most of these are from other domains of application, such as mechanical design, but the last two works presented address the software design domain.

Qian and Gero [23] show how analogy-based design can be implemented using the Function-Behavior-Structure (FBS) knowledge representation. They claim that the FBS path connecting function with behavior, and behavior with structure is the key to analogy in design. They consider that function and behavior are analogical retrieval cues, due to the one to many mapping between function and behavior, and between behavior and structure.

KDSA [24] is an approach to analogy-based design, using a controlled search algorithm in a semantic network. The base idea of this approach is that good analogies stem from artifacts with similar functions, but different behaviors and structures.

Goel and Bhatta [25] developed model-based analogy, a computational theory of analogy-based design that uses domain models about artifacts. These models represent the structure, behavior and function of a design. It uses design patterns as generic design abstractions that are learned during system operation. A design pattern can then be applied to a concrete situation, generating a new analog design.

CADET is a system developed by Sycara and Navinchandra [26] that uses behavior as a thematic abstraction for case retrieval. They use influences as an appropriate thematic abstraction for representing behavior of physical devices.

There are some research works that address the area of analogy applied to software reuse. From these we point out the work of Maiden and Sutcliffe in Ira [27]. Ira is a CASE tool system that enables the reuse of software specifications based on analogy. It provides user support for the task of system specification. Specification reuse involves three processes: categorization of a new problem, selection of candidate specifications, and adaptation of the selected specification to the new domain. Ira addresses these three issues by: obtaining the description of the new target problem from the software engineer; controlling the interaction with the user during selection and adaptation of an analogous specification; and reasoning with critical problem features to match new problems.

Spanoudakis [28] developed a computational model of similarity for analogical software reuse based on conceptual descriptions of software artifacts. This approach is based on semantic similarity of software objects.

7. Conclusions and future work

This paper describes the analogy module of REBUILDER, and presents some preliminary experiments exploring the relation between the retrieval strategy used and the novelty and utility of analogy-generated solutions. From these first experimental results, it can be inferred that semantic retrieval generates more useful class diagrams, but they are less novel than diagrams generated using structural strategies. These findings are in accordance with our idea that there is a trade-off between novelty and usefulness. Using a combination of both retrieval aspects indicates that it is a good retrieval strategy for generation of designs using analogy. Future work will address further variations of the retrieval strategies and different generation mechanisms, like design composition. An empirical study on a real development environment is also a goal of our research.

References

[1] S. Dasgupta, Creativity, invention and the computational metaphor: Prolegomenon to a case study, in: T. Dartnall (Ed.), Artificial Intelligence and Creativity, Kluwer Academic Publishers, Dordrecht, 1994.

[2] R. Brown, Creativity: What are we to measure? in: J. Glover, R. Ronning, C. Reynolds (Eds.), Handbook of Creativity, Plenum Press, New York, 1989.

[3] D. Partridge, J. Rowe, Computers and Creativity, Intellect Books, 1994.

[4] J. Gero, Computational models of creative design processes, in: T. Dartnall (Ed.), Artificial Intelligence and Creativity, Kluwer Academic Publishers, Dordretch, 1994.
[5] J. Gero, M.L. Maher, Modelling Creativity and Knowledge-Based Creative Design, Lawrence Erlbaum Associates, Sydney, 1993.
[6] F. C. Pereira, A. Cardoso, Knowledge integration with conceptual blending, in: 12th Irish Conference on Artificial Intelligence & Cognitive Science (AICS 2001), Ireland, 2001.
[7] P. Gomes, C. Bento, P. Gago, E. Costa, Towards a case-based model for creative processes, in: 12th European Conference on Artificial Intelligence (ECAI'96), John Wiley, Budapest, Hungary, 1996.
[8] J. Gero, Introduction: Creativity and design, in: T. Dartnall (Ed.), Artificial Intelligence and Creativity, Kluwer Academic Publishers, Dordretch, 1994.
[9] D. Gentner, Structure mapping: a theoretical framework for analogy, Cognit. Sci. 7 (2) (1983) 155–170.
[10] R. Hall, Computational approaches to analogical reasoning: a comparative analysis, Artif. Intell. 39 (1) (1989) 39–120.
[11] K.J. Holyoak, P. Thagard, Analogical mapping by constraint satisfaction, Cognit. Sci. 13 (1989) 295–355.
[12] P. Thagard, K.J. Holyoak, G. Nelson, D. Gochfield, Analog retrieval by constraint satisfaction, Artif. Intell. 46 (1990) 259–310.
[13] D. Gentner, D. Forbus, Mac/fac: A model of similarity-based retrieval, in: 13th Conference of the Cognitive Science Society, 1991, pp. 504–509.
[14] T. Plate, Distributed representations and nested compositional structure, Ph.d., University of Toronto, 1994.
[15] B.P. Crean, D. O'Donoghue, Features of structural retrieval, in: IASTED – International Symposia, Applied Informatics, Innsbruck, Austria, 2001, pp. 295–300.
[16] P. Gomes, F.C. Pereira, P. Paiva, N. Seco, P. Carreiro, J.L. Ferreira, C. Bento, Case retrieval of software designs using wordnet, in: F.v. Harmelen (Ed.), European Conference on Artificial Intelligence (ECAI'02), IOS Press, Amsterdam, Lyon, France, 2002.
[17] J. Kolodner, Case-Based Reasoning, Morgan Kaufman, Los Alto, CA, 1993.
[18] G. Miller, R. Beckwith, C. Fellbaum, D. Gross, K.J. Miller, Introduction to wordnet: an on-line lexical database, Int. J. Lexicograp. 3 (4) (1990) 235–244.
[19] J. Rumbaugh, I. Jacobson, G. Booch, The Unified Modeling Language Reference Manual, Addison-Wesley, Reading, MA, 1998.
[20] D. O'Donoghue, B. Crean, Searching for serendipitous analogies, in: European Conference on Artificial Intelligence ECAI'02 Workshop: 2nd Workshop on Creative Systems, Lyon, France, 2002.
[21] B. Crean, D. O'Donoghue, Radar: finding analogies using attributes of structure, in: Proceedings of the 13th Irish Conference on Artificial Intelligence and Cognitive Science (AICS'02), Springer-Verlag, Limerick, Ireland, 2002, pp. 20–27.
[22] P. Gomes, F.C. Pereira, P. Paiva, N. Seco, P. Carreiro, J.L. Ferreira, C. Bento, Experiments on software design novelty using analogy, in: European Conference on Artificial Intelligence ECAI'02 Workshop: 2nd Workshop on Creative Systems, Lyon, France, 2002.
[23] L. Qian, J.S. Gero, Funtion-behaviour-structure paths and their role in analogy-based design, Artif. Intell. Engin. Des. Anal. Manuf. 10 (1996) 289–312.
[24] M. Wolverton, B. Hayes-Roth, Retrieving semantically distant analogies with knowledge-directed spreading activation12th National Conference on Artificial Intelligence (AAAI-94), vol. 1, AAAI Press/The MIT Press, Seattle, Washington, USA, 1994, pp. 56–61.
[25] S. Bhatta, A. Goel, Design patterns; a computational theory of analogical design, in: International Joint Conference on Artificial Intelligence (IJCAI'97), 1997.
[26] K. Sycara, D. Navinchandra, Influences: a thematic abstraction for creative use of multiple cases., in: First European Workshop on Case-Based Reasoning, 1991.
[27] N. Maiden, A. Sutcliffe, Exploiting reusable specifications through analogy, Communications of the ACM 35 (4) (1992) 55–64.
[28] G. Spanoudakis, P. Constantopoulos, Similarity for analogical software reuse: a computational model, in: A. Cohn (Ed.), 11th European Conference on Artificial Intelligence, John Wiley, Amesterdam, The Netherlands, 1994, pp. 18–22.