Similarity for Analogical Software Reuse: A Conceptual Modelling Approach

George Spanoudakis and Panos Constantopoulos

Institute of Computer Science
Foundation of Research and Technology – Hellas

Abstract

We present our approach to defining similarity between software artifacts and discuss its potential exploitation in software reuse by analogy. We first establish properties of similarity which support its role in retrieving and mapping software descriptions. Then we develop a systematic basis for comparison within a fairly general conceptual modelling framework, whereby comparable elements of the descriptions of software objects and corresponding similarity criteria are identified. Finally, a general form of distance metrics for the computation of similarity measures is defined.

1. Introduction: The analogical based approach to software reuse

The reuse based approach to software development has been realized as a form of analogical problem solving, where a reasoner, the software engineer, familiar with a variety of source cases(i.e development histories and systems) attempts to transfer properties, relations and solutions from them to a target system under development[22,24].

This realization brings reuse in a ground rich in theoretical predictions and experimental evidence from a variety of scientific disciplines(AI, Psychology, Cognitive Science and Philosophy). A ground which offers a generic process schema of analogical reasoning, predicting 4 distinct stages:

1. the *retrieval* of a set of source cases given a target description ;

2. the *mapping* of knowledge from some of the retrieved sources to the target on the basis of appropriate analogies between them, established at the same stage ;

3. the *evaluation* of the transferred knowledge in terms of consistency and pragmatic utility ; and

4. the *consolidation* of the final outcome into structures of knowledge, supporting similar forms of reasoning in the future.

It is reasonable to believe that research into reuse can benefit from this conceptual framework. In fact, the solutions to the operational problems of reusability(i.e retrieval, comprehensibility, modifiability and composition[3]) offered so far, can be reviewed in this framework, enhanced with new ones motivated by it and ideally integrated into a generic and theoretically plausible methodology according to the predictions of analogical reasoning.

Adopting this approach, we focus on the problem of defining a quantitative similarity relation between descriptions of software artifacts so as to promote their analogical reuse. The potential of similarity has been pointed out in both the literature relevant to reuse[4,6,7,8] and the literature relevant to analogical reasoning[1,15,28].

As far as analogical reasoning is concerned, the role of similarity is clear during the stage of retrieval. Either as a fine evaluator of source cases initially selected by some coarser search-method(e.g constrained spreading of activation[5]) or as a basic retrieval mechanism, similarity can provide effective trial hints to the stage of mapping that follows retrieval. However, the effectiveness of similarity depends on its very definition and whether it avoids certain undesired effects that are already known(see the discussion about the importance of features below).

In this paper, we present our on-going work on the concept of similarity. The final aim is the formulation of a computational model of similarity, in a way consistent with properties inherent to relevant kinds of human processing(as they are revealed by studies in Psychology[31,34]) and dealing effectively with

reuse-oriented forms of analogical reasoning. To this end we distinguish two main stages. The first stage aims at a generic method for computing similarity from object oriented qualitative descriptions, while the second deals with specific issues of software reuse.

This work is carried out within project NATURE of the ESPRIT Basic Research Action[40], which pursues the development of requirements engineering methodologies and is connected with ongoing work on knowledge organization for software reuse(Software Information Base[35]) carried out within ESPRIT project ITHACA.

2. A general review of Similarity

In general terms, similarity is described as a relation determined by some flexible comparison between the distinct constituents of two entities (i.e situations, cases, natural or nominal-kind objects)[23,29,31,34,36]. From a quantitative viewpoint, the result of this comparison may be interpreted in two basic ways:

i. as a measure of closeness in some abstract space[23,34] and

ii. as a probability that the objects under comparison, would resemble each other even if their possibly missing constituents were considered as well[9,28].

Both the representation of objects and the nature of the comparisons are approached differently in the literature.

As far as the representaion of objects is concerned there exist two basic approaches. The first assumes the existence of a predefined set of features adequate for representing any object within a given domain of discourse. Consequently an object is viewed as possessing a subset of these features by giving appropriate values to them. The second approach assumes that objects are representable through their classification into a set of mutually disjoint and exhaustive classes.

Regarding the comparisons, we can also distinguish two basic forms:

(1) the *exact-matching form*, which results into three sets of object-features:

$$\textit{i. the common features}: S1 = F(o1) \cap F(o2)$$

$$\textit{ii.the distinctive features of the first object}: S2 = F(o1) - F(o2) \quad \textit{and,}$$

$$\textit{iii.the distinctive features the second object}: S3 = F(o2) - F(o1)$$

Given these three sets, the evaluation of similarity, in general has one of the following functional forms:

i. F(S1, S2, S3) that takes into account both the common and the distinct features of the objects. Typical instances of this form are[34]:

$$\textit{a.the ratio model}: \frac{wf(S1)}{af(S2) + bf(S3)} \quad \textit{and,}$$

$$\textit{b.the contrast model}: wf(S1) - af(S2) - bf(S3)$$

ii. F(S1) that takes into account only the common features of the two objects. In this category, we can classify the frequency model of similarity[31]:

$$f(|C|, |U|)$$

where

- |C| is the cardinality of the common class of two objects
- |U| is the cardinality of the entire universe of objects and,
- f is a function increasing in |C| and decreasing in |U|

(2) the *distance-based form*, which assumes the existence of metrics measuring the distance of features according to the types of their domains(i.e binary, nominal, ordinal or linear domains). Typical such metrics include:

i. the identity function for nominal domains and

ii. the absolute difference function for totally ordered domains[2,9]:

$$d(v1,v2) = |order(v1) - order(v2)|$$

Apart from the previous issues of object representation and comparison forms, two more aspects of similarity should be pointed out, namely the context effect and the symmetric nature of the relation.

A variety of experiments lead to the conclusion that the similarity between objects is affected by their common context. This effect has been attributed either to the dependence of the salience of the various object features on this context or to variations on the cardinality of the common class that the objects under comparison belong to, which in turn is affected by context changes(see the frequency perspective above).

A more arguable issue is the symmetric nature of similarity. In fact, reported evidence supports both the non-symmetric and the symmetric views[34]. Generally speaking the non-symmetric view regards similarity as the result of a directed comparison, which designates two basic roles for the objects involved. These are the role of the *subject* and the role of the *referent*. The features of the subject are assumed to have a salience greater than the features of the referent. Therefore, differences relevant to the features of the subject are emphasized more than differences relevant to the features of the referent. On the other hand, the symmetric view does not distinguish between these different roles and thus is does not differentiate the salience of features according to them. Hence, the direction of the comparison, does not affect the weights of the differences, resulting in equal similarity estimates.

3. TELOS: A representational framework for Similarity Evaluation

In this section we briefly review the structural part of the TELOS data model[25,35]. This model will be the basis for the representation of objects in our framework. Our choice is primarily motivated by the generic nature of this model, which in turn subsumes an entire family of object oriented data models, at least with respect to their structural elements. In addition, TELOS has been acknowledged as language for describing software artifacts[6].

TELOS provides three basic abstractions, namely classification, generalization and attribution.

Classification defines an infinite dimension along which objects can be classified as Tokens, or Simple Classes, or Meta classes and so on. This is accomplished through the declaration of an object as an instance of the built-in disjoint classes that distinguish between the successive levels of classification(i.e Token class, S_Class class, M1_Class class, M2_Class class and so on). In addition an object can be classified as an instance of other user-defined classes thus being capable of instantiating the attributes attached to these classes.

Objects which are classes(i.e they have not been classified as Tokens) can be generalized and/or specialized into other classes, through Isa-relations. In TELOS, these relations have a set inclusion semantics and are restricted only between classes of the same classification level. A class C may have one or more direct superclasses, whose superclasses are also considered as superclasses of C (Isa transitivity). The Isa relations in general form a directed acyclic graph, the edges of which allow the inheritance of attributes from the superclasses to the subclasses(strict and multiple inheritance).

The attribution mechanism allows the attachment of attributes to objects. It must be emphasized that attributes in TELOS are also objects. Thus they can be classified into attribute classes, generalized, specialized and have attributes of their own. The term attribute captures both classes and tokens of attributes. Attribute classes may be single-valued or multi-valued (i.e an object may have one or more attributes, which are instances of the same attribute class) and necessary or not.

In addition to its modelling according to these abstractions, a TELOS object is associated with two unique identifiers. The first of them is known as *system identifier* and has the role of a surrogate. It is generated automatically by the system, and is associated with a particular object during its entire life-cycle. A system identifier can not be reused for another object and thus it supports a unique identification. The second type of identifiers is the *logical name*. A logical name is attached to an object by its creator and accomplishes logical references to it. Although it may be changed during the life-cycle of an object, a logical name ought to be system-wide unique. The logical names of attribute objects are composed of the logical names of their possessing objects and logical referents attached directly to them, called *labels*. For instance, the logical name of an attribute object with label X, possessed by an object with name O is O.X. O.X must be system-wide unique, while X must be unique only within the scope of object O.

According to the previous overview, four basic forms of TELOS objects can be distinguished:

1. Entity Tokens: ETi = [Id(i), In(i), A(i)]
2. Attribute Tokens: ATi = [Id(i), Id(From(i)), In(i), A(i), Id(To(i))]
3. Entity Classes: ECj = [Id(j), In(j), Isa(j), A(j)]
4. Attribute Classes: ACj = [Id(j), Id(From(j)), In(j), Isa(j), A(j), Id(To(j))]

where

i. Id(x) is a system identifier denoting the object Ox

ii. In(x) is a set of system identifiers denoting the classes of the object Ox

iii. Isa(x) is a set of system identifiers denoting the direct superclasses of the class-object Ox

iv. A(x) is a set of system identifiers denoting the attributes attached to the object Ox

v. From(x) denotes the object possessing the attribute Ox and,

vi. To(x) denotes the object pointed to by the attribute Ox

Two more issues regarding the topology of the Isa-graphs and the inherent assumptions of the inheritance mechanism of TELOS are of interest from the similarity perspective.

In TELOS, an Isa-graph generally consists of M disjoint subgraphs, where M is the number of classification levels used in the particular schema(see figure 1). The Isa graphs are disjoint because of the restriction of the Isa relations only between classes of the same classification level.

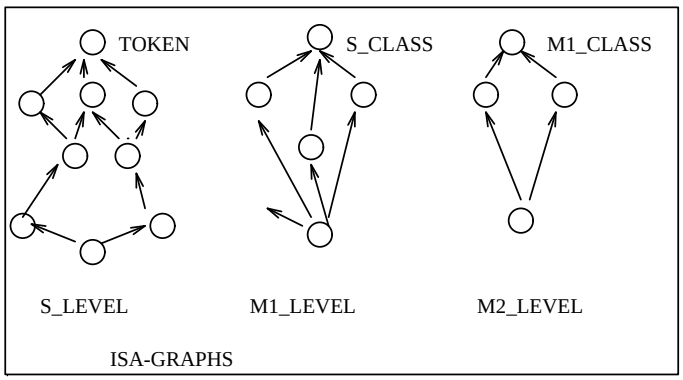

Figure 1: Distinct TELOS Isa-graphs

The set-inclusion semantics of the Isa-relation make it a partial-order relation[20]. Moreover, at each level of classification TELOS provides a most general class, whose extension contains all the objects of the lower level (see Token, S_Class and M1_Class in figure 1). Therefore the extensions of the rest of the classes at the same classification level are subsets of its extension and due to the set-inclusion semantics of the Isa-relations, these classes are regarded as subclasses of it. Thus two classes in an Isa subgraph will always have a common superclass, no matter if this is the built-in most general class for the relevant level of classification.

As far as the strict multiple inheritance mechanism is concerned, the key underlying hypothesis is that the identity of attribute-labels along an Isa-path implies the semantic identity of the relevant attribute-objects. This, due to the strict inheritance, has the consequence that attributes in subclasses, having the same labels with attributes in their superclasses, can only specialize them(see figure 5).

4. A scenario of exploiting similarity in requirements reuse

To illustrate the role of similarity and analogical reasoning in software reuse, we present a hypothetical scenario of incremental system specification in which requirements of existing information systems are reused.

These requirements are organized according to a three-level conceptual schema. The higher level of it includes models for describing the so called[25] Subject, System, Usage and Development Worlds of an Information System. Roughly speaking, these worlds correspond to the application domain, the functionality, the operational environment and the development process of an information system respectively. The next level consists of abstract specifications of system types, built according to these four perspectives(e.g LibrarySystem in figure 2). The lower level comprises descriptions of specific systems, classified into one or more of the mentioned abstract specifications(e.g MunicipalLibrary1 in figure 2).

Rather than presenting user-system dialogs, we give a rough process description, including steps where similarity and analogical reasoning are expected to enable reuse. Also, we refer to other generic reasoning and searching techniques, which could be used for carrying out particular actions of specification.

Our scenario is by no means complete and is based on an interactive view on reuse, where a software engineer uses a set of tools supporting reuse (e.g an information base, a browser, a similarity computation module, retrieval methods, integrity checking) which do not fully automate the entire process nor do they assume a unique reuse strategy(e.g templates approach).

Let us assume that an engineer wants to specify the requirements for a Car Rental Agency system and proceeds through the following steps:

STEP 1:

The software engineer asks for a requirements description, named as CarRentalSystem. The search fails and consequently he is prompted to choose one of the two complementary models of requirements descriptions, namely, the *SubjectWorldModel* and the *SystemWorldModel* to start the specification of the new system.

Assuming that the SystemWorldModel is selected, the system presents it to the engineer(i.e SystemWorldModel in the Schema of figure 2) and prompts him to specify the top-level *transactions* and *data classes* of the new system by names and other semantic terms.

STEP 2:

In response to this prompt, the engineer specifies the following transactions and data classes, accompanied by appropriate terms.

a. Transactions:

1. rentACar (hire, lend)

b. Data Classes

1. Car (Vehicle, automobile)

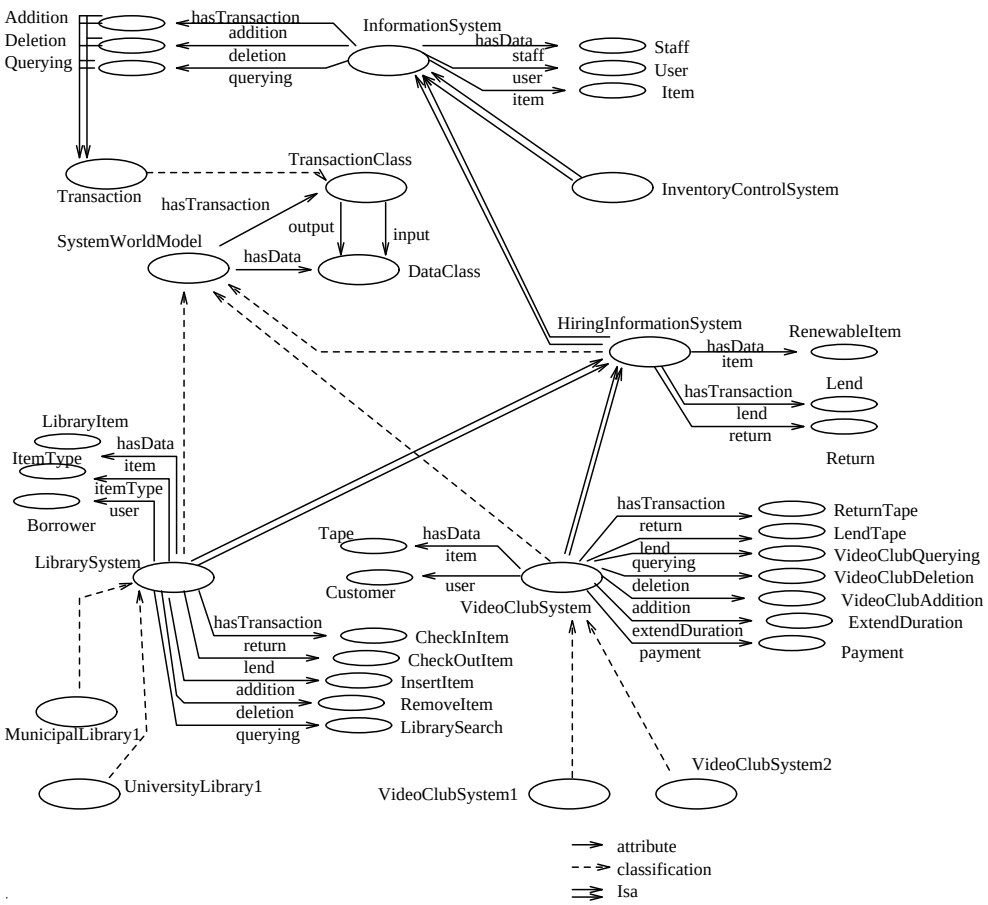

Figure 2: A model for Requirements Descriptions

2. returnACar () 2. Customer (user)

3. insertANewCar(file-in,insert,add)

4. removeACar(delete)

STEP 3:

Using these names and the associated terms as search-hints, the system starts a constrained spreading of activation within the abstract SystemWorld descriptions, to allocate those including any of these items. This process results in two SystemWorld abstract specifications, namely:

1. the LibrarySystem and,

2. the VideoClubSystem

Moreover, in order to help the engineer, to understand these abstractions[21], the system presents him the two most characteristic(prototypical) instances of each along with their degree of prototypicality:

1.1 UniversityLibrary1 0.62

1.2 MunicipalLibrary1 0.55

2.1 VideoClubSystem1 0.78

2.2 VideoClubSystem2 0.61

The prototypicality of instances with respect to some of their classes is computed as their average similarity to the rest instances of the extensions of these classes[34].

STEP 4:

The engineer focuses on the VideoClubSystem1 and the UniversityLibrary1 (see figure 3).

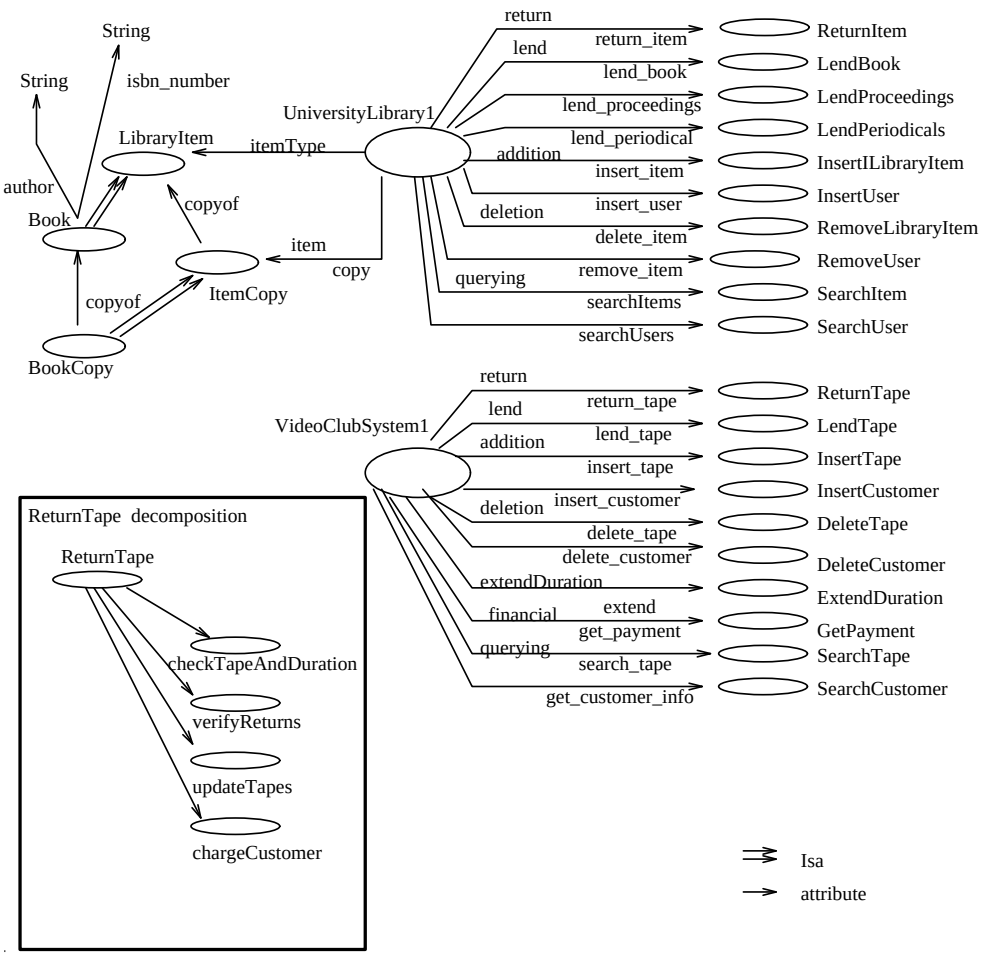

Figure 3: Specific Requirements Descriptions

After browsing through them, he proceeds in a scavenging mode of copying elements of these systems into the on-going specification of the CarRentalSystem. This results in the following transactions:

lend
 rentVehicle: RentVehicle

return
 returnVehicle: ReturnVehicle

addition
 insertVehicle: InsertVehicle
 insertCustomer: InsertNewCustomer

remove
 removeVehicle: RemoveVehicle
 removeCustomer: RemoveCustomer

querying

createReport: CreateVehicleReport

searchForVehicle: SearchForVehicle

getCustomerRecord: GetCustomerRecord

STEP 5:

Next, the engineer specifies the input and the output data classes of the transactions of the CarRentalSystem which in the case of the ReturnVehicle transaction are the following:

input	output
input1: Customer.customer_id	output1: RentedVehiclesList
input2: Extension_period	output2: VehicleBillings
input3: ReturnVerification	output3: Receipt
input4: BillingVerification	

These inputs and outputs correspond to his generic view about the functionality of this transaction. ReturnVehicle is assumed to retrieve a list of the vehicles rented by a customer, given his identifier, display them and get a response on whether the customer is going to return the vehicle or extend the renting period, bill him, verify which of the rented vehicles are to be paid immediately and which will be charged to the customer's open balance and finally print a receipt for the immediate payments.

Once the input and the output for all the top-level transactions have been specified, the engineer puts more specific similarity questions about each of the top-level transactions of the system under development. His underlying assumption is that similarity in terms of inputs and outputs, specified so far may imply functional similarity with prospects for reuse. He also, decides to restrict the search space to transactions of systems classified as only as LibrarySystems or VideoClubSystems. In response to his queries the system retrieves the two most similar transactions to each of the transactions of the CarRentalSystem:

i. InsertVehicle

1. insertLibraryItem1 (of UniversityLibrary1) 0.6
2. insertNewTape (of VideoClubSystem1) 0.52

...

ii. ReturnVehicle

1. checkInTape (of VideoClubSystem1) 0.5
2. returnLibraryItem (of MunicipalLibrary1) 0.44

The engineer browses through these transactions. In the case of the checkInTape and the returnLibraryItem, their decompositions yield the following subtransactions:

1. checkInTape	2. returnLibraryItem
1.1 checkTapeAndDuration	2.1 validateBook
1.2 verifyReturns	2.2 validateLoanDuration
1.3 updateTapes	2.3 updateReturns
1.4 chargeCustomer	

Motivated by these results, he specifies the ReturnVehicle transaction as follows:

1. CheckRentedItems
2. VerifyReturns
3. ExtendLendingDuration

4. UpdateVehicles
5. Billing
6. PaymentVerification
7. ChargeClient
8. IssueReceipt

He also introduces the input and the output data classes of these subtransactions. This more detailed specification prompts him to ask again for the similarity of the ReturnVehicle to other existing transactions, in order to check whether their exact functional specifications could be further reused. The next system response includes the following:

1. checkInTape (of VideoClubSystem1) 0.73
2. returnLibraryItem (of MunicipalLibrary1) 0.32

STEP 6:

Based on these results the engineer further describes the functionality of the ReturnVehicle according to the details of the checkInTape. However, the output of the subtransaction IssueReceipt (i.e Receipt) is not similar to any output data class, of transactions associated with any Library or VideoClub systems. Hence, he decides to enlarge his search space to include the entire extension of the InformationSystem abstraction. This results to the retrieval of a similar transaction from an InventoryControlSystem, namely the transaction IssueInvoice(similarity: 0.57, output: Invoice), which is being successfully reused in terms of its functional details.

STEP 7:

Having completed the definition of the transaction ReturnVehicle and the specification of the input and the output data classes of its subtransactions, the engineer attempts to classify it as an instance of the class Transaction(see figure 2). However, he fails due to the subtransaction Billing. This subtransaction contradicts a constraint associated with all transactions, demanding that the input data classes of all the subtransactions of a parent transaction X, must either be output data classes of some other subtransaction of X or input data classes to X.

In the case of Billing the input data class VerifiedItems defined as:

Class VerifiedItems in List with
element
item: RentedVehicle

violates the previous constraint. The system, after checking the relevant constraint, informs the engineer about the inconsistency. Consequently, he asks the system for output data classes of other subtransactions of the ReturnVehicle, that are similar to the VerifiedItems class and gets the following response:

1. RentedVehicles 0.9
2. UnverifiedItems 0.6
3. Billings 0.3

Then he notices that the RentedVehicles, which is the output data class of the subtransaction VerifyReturns, should be the input to the Billing and changes the name VerifiedItems to RentedVehicles.

...

STEP N:

Once the software engineer has completed the System World Specification of the CarRentalSystem, he wants to classify it under the appropriate SystemWorld abstraction. Thus, he asks for the system for an optimal classification suggestion. The system responds by estimating the average similarity of the new specification, with those belonging to the extensions of the available abstractions and supplying a ranked

list as follows:

1. HiringInformationSystem 0.68
2. InformationSystem 0.62
3. VideoClub 0.43
4. LibrarySystem 0.3
5. InventoryControlSystem 0.07

The engineer decides to classify it under the HiringInformationSystem.

As the previous scenario illustrates, similarity and analogical reasoning are viewed as basic techniques, that complement other forms of searching and reasoning(e.g deduction, spreading activation, integrity checking) in dealing with the operational problems of reuse. Similarity can support analogical reuse in at least the following ways:

i. as a fine search method for retrieving potentially reusable artifacts(steps 5,6)

ii. by providing a basis for prototypicality estimates and selections of instances of abstractions(step 3)

iii. by providing a criterion of replacement of sources of integrity violations, whenever their retraction is not desirable(step 7)

iv. by providing a criterion of classification to consolidate cases for future analogical reuse(step N)

5. Principles of Similarity Computation

In this section we formulate a set of principles underlying the computation of similarity. These principles reflect requirements imposed by the context of analogical reasoning and take into account the outlined representational framework. Some of them are also justified by psychological predictions about the ways that humans realize similarities between natural and nominal-kind objects.

The Principle of Ontological Uniformity

In general objects may correspond to different levels of abstraction. This in the notation of TELOS is basically expressed through the classification level of an object. Roughly speaking, objects may refer to atomic real world entities(i.e Tokens), to abstracted classes of atomic entities(i.e S_Classes), to models enabling the generation of such abstractions(i.e M1_Classes) and so on. In this object space, it would be senseless to attempt comparisons between entities of different ontologies. Any such comparison would be ad hoc and therefore not semantically interpretable. Hence, inter-object comparisons are restricted according to the following principle of **Ontological Uniformity**:

(P1) Similarity comparisons are only valid between objects of the same classification level

This principle seems to be arguable in the context of concept formation[11] but this is only due to a confusion. The underlying reason is that the term similarity is used to describe comparisons between objects of successive levels of classification. However, these comparisons have a different objective. They are a means of deciding whether or not the lower, in terms of classification, object could be classified as an instance of the higher one. In fact, they end up in a true/false result, according to whether or not the observed attribute values of the lower object belong to the predicted domains for the relevant attributes in the higher one. But this is different from expressing the resemblance between two values or two attribute domains.

It must also be pointed out, that when humans generate concepts the predicted comparisons do not involve directly the objects of the higher level. In some cases, they are carried out implicitly through one or more prototypes of the concept in hand(figure 4, case a)[32,39]. In other cases, the higher object corresponds to an exemplar-based representation of the relevant concept (figure 4, case c) and thus it does not really belong to a higher level of classification. Finally, there exist cases where the comparison is performed in an aggregate fashion, taking into account the resemblances of the object to be classified with all the other

known instances of the concept(figure 4, case b)[27,34].

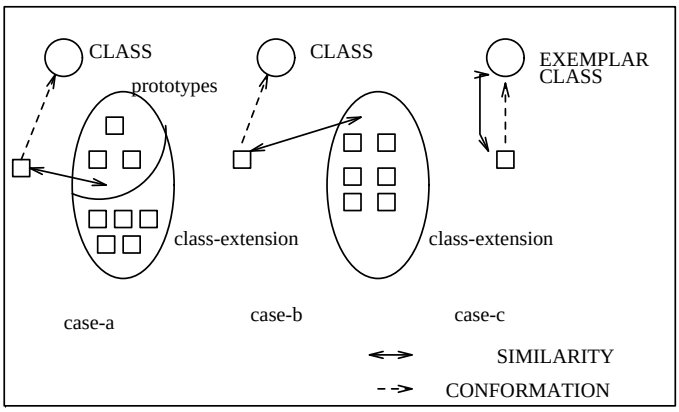

Figure 4: Comparisons between Classes and Instances

The principle of partially uniform representation

A critical issue for the evaluation of similarity is the uniformity of the underlying object representation, which, if valid, dictates that any object can be described through a fixed predefined set of features. Such a hypothesis is a good approximation of reality in narrow, well-defined and rather mature domains. But in general these characteristics are rare in analogical reasoning and software reuse. In these areas, the contexts involved include objects from different domains(e.g diverse application areas of software), described via non-standard and possibly not well-understood properties, and usually instantiating models reflecting diverse viewpoints[6,10,25,35].

As a result it is reasonable to assume only partially uniform representations of objects. An example of what partially uniform representation means is given in figure 5.

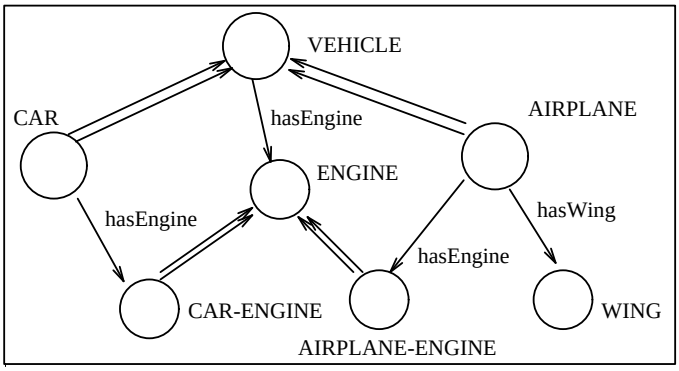

Figure 5: Isa-graphs and Inheritance

In that figure the two different vehicle types, the airplane and the car are uniform in sharing the possession of an engine(despite the different engine types) and diverse with respect to the presence of the wings.

Partial uniformity is still reasonable under the presence of domain specific methods for deducing attribute equivalences[1], since distinct but equivalent attributes will be treated as common.

Partial uniformity also suggests the restriction of comparisons only between the common attributes of two objects. This is justified by a global view of the analogical reasoning process. During the early stage of retrieval, two basic types of errors may occur. The first is the rejection of sources that could later enable sound and useful analogical transfers(*errors of type A*). The second captures the case where candidate sources are retrieved, though not enabling any acceptable transfer(*errors of type B*). Errors of type B increase the overall computational cost of the process, since they enlarge the set of the source cases to be considered for transfer during the mapping stage, yet they are recoverable during justification. On the contrary it would be impossible to recover from errors of type A. This limitation makes errors of type A more important than errors of type B.

Consequently, comparisons between the uncommon attributes of two objects should not be attempted. Such comparisons necessitate ad hoc matchings between ordinary and artificial values, that can only increase the dissimilarity of the involved objects, while not contributing in a positive way to any transfer.

The previous analysis is summarized into the principle of **Partially Uniform Representation**, stating that:

(P2) Objects have only partially uniform representations. The non-uniform parts of their representation must be excluded from similarity comparison

The normalization principle

Distance measures subsume exact matching(see the fourth axiom below) by giving finer evaluations of differences. Thus, they must be preferred as a basis for estimating similarity. Central to our approach is the idea that an overall distance metric D can be defined as an aggregate function over partial metrics, devised in accordance with the semantics of the three distinct abstractions composing the object descriptions.

D is a mapping of the form:

D: O x O ----> R

where O is the set of all objects in the context of similarity estimation, R is the set of Reals and D obeys the known metric axioms[20]:

(1) D(x,y) >= D(x,x) = 0

(2) D(x,y) = D(y,x)

(3) D(x.z) <= D(x,y) + D(y,z) and.

(4) D(x,y) = 0 ==> x = y

Similarity has been viewed as a monotonically decreasing function f of distance D [28]. A basic issue is whether f must be defined over absolute or relative distance measures. Certain characteristics of the abstractions over which the absolute distances must eventually be defined, strongly favor relative measures. Generally speaking, the elementary absolute distance metrics must be defined over Isa-graphs of classes or over domains of attributes that may be nominal, partially ordered or totally ordered(i.e linear). In all but the case of the nominal domains, which suggests an identity distance function, the relevant domains are not expected to have equal widths[28] and the Isa-graphs are expected to have diverse coarsenesses. Therefore, normalizations of absolute metrics are necessary for filtering out differences arising due to coarser and finer representations.

This requirement is summarized in the following principle of **Normalization**:

(P3) Similarity(S) is a monotonically decreasing function of a normalized Distance measure(Dr)

A trivial consequence of this principle is that:

Dr = 1 <==> S = 0 and

Dr = 0 <==> S = 1

The importance of features in Similarity Evaluation

The distinct features of objects(i.e relationships and attributes) have a different impact in both the evaluation of similarity and the success of analogical transfer. Usually referred to as salience, this impact can be attributed to three basic factors, namely the domain dominance, the pragmatic utility and the classificatory significance of a feature. Salience is critical since its overestimation or underestimation may prevent analogical transfer or enable erroneous analogies[18,26,30]. Consequently it must be carefully analyzed and quantified before taken into account by similarity estimates.

i. The domain dominance of features

The domain dominance factor of salience reflects the causality of a feature within a particular domain. Causality is determined by the dependence of values or even the presence of other features on a particular feature and has a direct effect on the mapping stage of analogical reasoning. Since dominance contributes to transfer of entire constellations of knowledge pieces, which is the ultimate purpose of analogical reasoning, this effect is obvious.

There exist two different points of view to the evaluation of the domain dominance. The first one[15,28,37] realizes it as presence in the antecedent parts of implications or in integrity constraints expressing inter-feature dependencies for a particular domain. The second approach relies on purely syntactic aspects of object representations. First, it distinguishes between relations and other representational elements(e.g identities and attributes) and then it recognizes only the former ones as being dominant[12,13].

Certainly, unless we want to hypothesize explicit and direct information about dominance, we must rely on syntactic elements for its evaluation. This approach could be justified on the basis of particular observations about the encoding process that result into an object knowledge base. In studies dealing with the surface/structure paradox of analogical reasoning[14], it has been pointed out that the encoding of a certain domain around dominant or structural features and not around superficial ones is subject to the training and the expertise of the encoder in the particular domain. In our case it is correct to assume that both the training and the expertise conditions of encoding are satisfied. Knowledge bases, used for analogical reasoning or software reuse are not normally developed by novices and furthermore they undergo constant improvements(e.g reverse engineering) during their life-cycle(see [7] for a similar viewpoint). Consequently domain dominant features will be prevalent in them.

Hence the principle of **the Domain Dominant Schema**:

(P4) The domain dominance of a feature can be determined from the schema of the relevant object knowledge base.

Here, the term schema has its usual database sense (i.e it captures the conceptual schema of the object descriptions).

ii. The pragmatic utility of features

The causal relations of features to goal attainment in the process of analogical reasoning designate what is called the pragmatic utility of features. Generally speaking, features are distinguished into those predicting the applicability of some method for achieving a goal, those predicting the success or failure of a certain solution, those that describe unusual outcomes of solutions and those which are totally irrelevant to goals and solutions. The first three of these associations result into a high pragmatic utility.

However, it is really difficult to distinguish between pragmatic utility and domain dominance in the absence of explicit information about the former(e.g Goal Dependency Graphs in [33]). Furthermore, even in the presence of relevant information, it is reasonable to assume that this information would lead to similar conclusions with those drawn only from the evaluation of domain dominance. In practice, domain dominance correlates with pragmatic utility since general causality is likely to lead to goal relevancy. Therefore, without sacrificing any important amount of information, we can rely on domain dominance in estimating pragmatic utility.

Thus, the principle of **the Pragmatic Utility Subsumption** suggests:

(P5) The pragmatic utility of a feature can be approximated by its domain dominance

In the particular case of software reuse, this principle is further justified by proposals relevant to the organization of software repositories. These proposals indicate conceptual schemas, which capture pragmatically important features(e.g correspondence links in the SIB[6], abstract domain classes in [22]).

iii. The classificatory Significance of features

The classificatory significance refers to the contribution of a feature in the production of classification schemas, being optimal with respect to certain criteria. Examples of such criteria are the inferential power of the schema in predicting missing attribute-values for instances in concept formation and the optimization of the precision measure in information retrieval.

In this framework, the sensitivity to errors of type A suggests a reasonable optimality criterion for a classification schema: the minimization of the probability of errors of type A. Consequently we can define the classificatory significance of a feature through the following principle of **Classification Optimality**:

(P6) The classificatory significance of a feature depends on its ability to produce classification schemas minimizing the probability of errors of type A in the analogical reasoning process.

6. Establishing A Basis for Comparison

The classification, the generalization and the attribution together with the two distinct types of identifiers are really expressive in describing objects but also impose the problem of deciding which pairs of elements included in object-descriptors, could be compared with each other.

The different nature of these constructs suggests that the various elements of an object representation can only be compared with elements of the same type. Otherwise we would not have a clear semantic basis for defining an appropriate distance metric. Therefore, we adopt the following top-level pairs of comparison, on the basis of our representational model:

1. Comparisons between Classifications:

   In(x) <---> In(y) {for any pair of objects (x,y)}

2. Comparisons between Generalizations:

   Isa(x) <---> Isa(y) {for any pair of classes (c1,c2)}

3. Comparisons between Attributes:

   A(x) <---> A(y) {for any pair of objects (x,y)}

4. Comparisons between System Identifiers:

   Id(x) <---> Id(y) {for any pair of objects (x,y)}

   Id(From(x)) <---> Id(From(y)) {for any pair of attributes (x,y)}

   Id(To(x)) <---> Id(To(y)) {for any pair of attributes (x,y)}

These top-level comparisons must be further elaborated in the case of the attributes, which can be utilized in representing properties and/or relations with different semantics (e.g Car.partOf, Car.mileage). A distinction is made between attribute classes and attribute tokens, due to the inheritance property employed only by the former and the different role of the attribute labels in them.

i. Comparisons between attribute classes

The inheritance mechanism associated with the Isa relations interferes with the attribution abstraction. Recall that attribute classes with the same labels along an Isa-path are perceived as being semantically identical and consequently only specializations of their specifications are permitted because of the strict inheritance model. Note also, that in the case of unordered classes(with respect to Isa relations) label

equality has no further implications.

This property of labels can be also employed in determining the valid comparisons between attribute classes. In fact, the attributes of a class can be distinguished into three main categories:

1 the attributes originally defined in this class ;

2. the inherited yet refined attributes ; and

3. the inherited but not refined attributes.

Assuming the following one to one mappings:

i. o1: I --> O

ii. o2: L --> O

iii. id1: L --> I

where O is the universal set of objects, I is the set of the system identifiers and L is the set of the logical names, we can define these attribute categories as follows:

Definition 1: The *Modifier* of a class C, M[C] is defined as the set of the labels of all the attributes included in its definition:

$$M[C]=\{\, x \mid id1(C.x) \in o2(C).A \,\}$$

where o.A is the set of the system identifiers for the attributes of the object o

The term modifier has been introduced in[37].

Definition 2: The set of the *unordered superclasses* of a class C, US[C]is defined as:

$$US[C]=$$

$$\{\, x \mid (id1(x) \in o2(C).isa^*) \text{ and } (not\ (exists\ y : (id1(y) \in o2(C).isa^*) \text{ and } (id1(x) \in o2(y).isa^*))) \,\}$$

where o.isa* is the transitive closure of the superclasses of the object o.

Definition 3: The *Intension* of a class C, INT[C] is defined as:

$$a.\ INT[C]=M[C] \text{ if } US[C] \text{ is empty}$$

$$b.\ INT[C]=\{\, x \mid (x \in M[C])\ or\ (x \in \bigcup_{j \in US[C]} INT[j]) \,\}\ otherwise$$

Definition 4: The *Horizontal Extension* of a Class C, with respect to a subset S={S1,...,Sm} of its superclasses, HE[C,S] is defined as:

$$HE[C,S]=$$

$$\{\, x \mid (x \in INT[C]) \text{ and } ((not(x \in \bigcup_{i \in S} Int[i]))) or (Forall\ i\ in\ S : (x \in INT[i]) \rightarrow (not\ (id1(i.x) \in o2(C.x) isa^*)))) \,\}$$

Thus, the horizontal extension of a class C with respect to some of its superclasses S1,..., Sm includes the labels of the attribute classes of C, which are not associated with any attribute classes of S1,...,Sm, if they are superclasses of the former(see a somehow different definition in [37]).

Definition 5: The *Vertical Replacement* of a class C, with respect to any of its superclasses S, VR[C,S] is defined as:

$$VR[C,S]=\{\, x \mid (x \in INT[C] \text{ and } (x \in INT[S]) \text{ and } (id1(S.x) \in o2(C.x).isa^*)\}$$

In words, the vertical replacement of a class with respect to one of its superclasses, includes the labels of its attributes which also specialize attributes with the same label inherited from this superclass.

Definition 6: The *Comparison Basis for Attribute Classes* of two classes C1 and C2, CBAC[C1,C2] is defined as:

$$CBAC[C1,C2]=$$

$$\{\, x \mid (id1(x) \in o2[C1].isa^*) \text{ and } (id1(x) \in o2[C2].isa^*) \; and \; (\; not \; (exists \; y : (id1(y) \in o2[C1].isa^*) \text{ and } (id1(y) \in o2[C2].isa^*) \text{ and } (id1(x) \in o2(y).isa^*)))\}$$

Since the Isa-relation is only a partial ordering and a class may have more than one superclasses, two classes C1 and C2 in general may have more than one pairwise unordered minimal common superclases. These will be the elements of their Comparison Basis for attribute classes(CBAC set). The CBAC set is guaranteed to be non-empty, since the most general TELOS built-in class at each level of classification is a superclass of all the classes at that level(see figure 1).

Given the previous definitions, we can distinguish between two categories of attributes that can be compared, with respect to two classes and their Comparison Basis:

1. The *Common Vertical Replacement*, which is a set of attribute pairs (c1.x, c2.y) defined as:

$$CVR[c1,c2,S]= \bigcup_{si \in CBAC[c1,c2]} CVR[c1,c2,Si]$$

where

$$CVR[c1,c2,Si]=\{\, (c1.x,\; c2.y) \mid (x \in VR[c1,Si]) \text{ and } (y \in VR[c2,Si]) \text{ and } (x = y)\}$$

The common vertical replacement includes the commonly inherited attributes, which have also been refined within the classes of consideration.

2. The *Unique Vertical Replacements* UVR[c1,c2,Si] and UVR[c2,c1,Si],are defined with respect to each of the superclasses Si in the comparison basis of the classes c1,c2 according to the following definition:

$$UVR[x1,x2,Si]=\{\, (x1.x,\; Si.y) \mid (x \in VR[x1,Si]) \text{ and } (not \; (x \in VR[x2,Si])) \text{ and } (x = y)\}$$

These sets pair attributes commonly inherited by superclasses, having identical labels but which are specialized in exactly one of the classes in hand.

The union of the Common and the Unique Vertical Replacements, defined as:

$$AT[C1,C2]=$$

$$CVR[C1,C2,CBAC[C1,C2]] \bigcup_{Si \in CBAC[C1,C2]} \{ UVR[C1,C2,Si] \} \bigcup_{Si \in CBAC[C1,C2]} \{ UVR[C2,C1,Si] \} \quad (1)$$

includes the pairs of attribute classes that must be taken into account in the estimation of the distance of two objects with respect to the aggregation abstraction. Note that attribute classes that belong to the horizontal extensions of the involved classes are not further compared, since they are not applicable to both of them(see the principle of the partially uniform representation).

ii. Comparisons between attribute tokens

In the case of attribute classes, comparison-pairs were formulated by using the Isa-graph for determining the Comparison Basis and the deduction of the semantic role identity from the equality of the attribute labels. However, both these features are either absent or do not have the same implications for attribute tokens. First, the Isa relation is not defined for atomic objects. Moreover, attribute tokens can not be further instantiated. Thus, their labels serve only as references within the relevant object-scopes and are meaningless outside them(e.g "mike_car" label in figure 6).

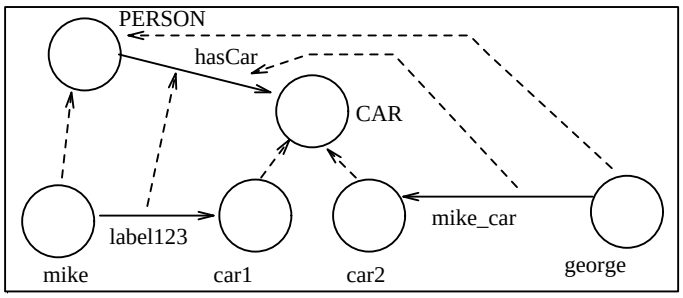

Figure 6: Entity objects and attribute tokens

Actually, the semantics of attribute tokens are expressed by their attribute classes. Therefore, their semantic identity can be detected from the labels of these classes. For instance, in the 6th figure we could compare the attribute token *mike.mike_car* with the attribute token *george.label123* due to their common classification under the attribute class *Person.hasCar* and despite their different labels.

In conclusion, attribute tokens yield comparison-pairs as follows:

Initially, the comparison basis for attribute tokens(CBAT) of two objects Oi, Oj is defined as:

$$CBAT[Oi,Oj]=Oi.in \cap Oj.in \text{ if } this\ intersection\ is\ not\ empty \quad otherwise,$$

$$CBAT[Oi,Oj]=$$

$$\left\{ x \mid (id\,1(x) \in CC(Oi,Oj)) \text{ and } (not\ (exists\ y : (id\,1(y) \in CC(Oi,Oj)) \text{ and } (id\,1(x) \in o\,2(y).isa^*))) \right\}$$

where CC(Oi,Oj) is the set of the common classes of Oi, Oj, defined as:

$$CC(Oi,Oj)= Oi.in^* \cap Oj.in^*$$

where:

$$Ok.in^*= \bigcup_{i \in Ok.in} i.isa^*$$

Then the set of the attribute classes which are applicable to both Oi and Oj(CAC) is defined as:

$$CAC[Oi,Oj]= \bigcup_{i \in CBAT[Oi,Oj]} INT[i]$$

Finally, we define the comparison-pairs consisting of sets of attribute tokens as:

$$CAT[Ok,Or]=$$

$$\left\{ \left(\left\{ X1,...,Xm \right\}, \left\{ Y1,...,Yn \right\} \right) \mid (forall\ i,j : (id\,1(Xi) \in Ok.A) \text{ and } (id\,1(Yj) \in Or.A) \text{ and } \right.$$

$$(exists\ z : (z\ \varepsilon\ CAC[Ok, Or]) \text{ and } (id\,1(z)\ \varepsilon\ o\,2(Ok.Xi).in^*) \text{ and } (id\,1(z)\ \varepsilon\ o\,2(Or.Yj).in^*)))\Big\} (2)$$

This definition reflects the possibility of objects instantiating relevant attribute classes by more than one attribute tokens.

iii. An example of attribute comparisons

According to the schema of the figure 7 we can form the following sets with respect to the classes UniversityDoctor(ud) and Professor(p):

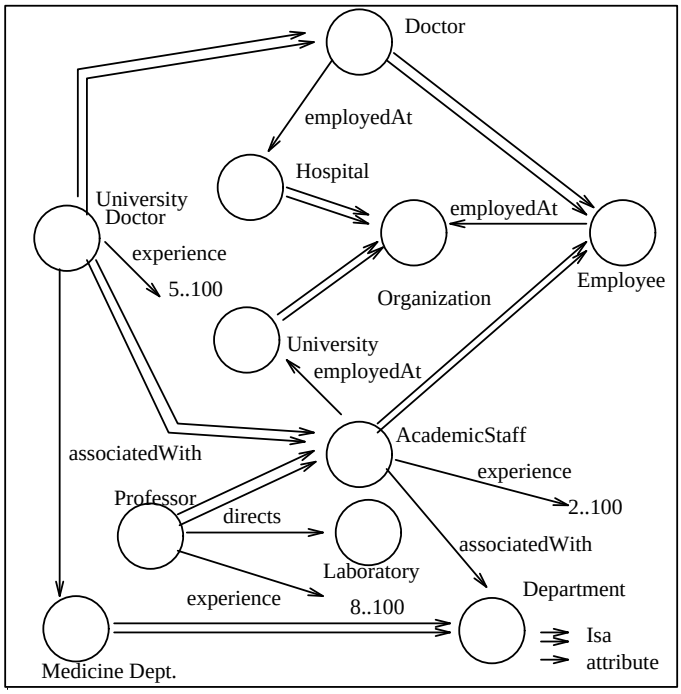

Figure 7: A schema for comparison of attribute classes

CBAC[ud,p] = { AcademicStaff }

HE[p,AcademicStaff] = { directs }

HE[ud,AcademicStaff] = { employedAt FROM doctor }

VR[ud,AcademicStaff] = { associatedWith, experience }

VR[p,AcademicStaff] = { experience }

CVR[p,ud,{AcademicStaff}] = {(p.experience, ud.experience)}

UVR[ud,p,AcademicStaff] = {(ud.associatedWith,AcademicStaff.associatedWith)}

UVR[p,ud,AcademicStaff] = {}

Note that when two attributes that have the same label are inherited from two distinct superclasses, they must be disambiguated with a FROM clause within the scope of the inheriting class, as in the case of the class Professor and the attribute employedAt.

Also, in the schema of the figure 8 the comparison pairs of the attribute tokens are:

CBAT[george, kate]={employee}

CAC[george,kate]={employee.salary, employee.experience}

CAT[george,kate]={({george.salary1},{kate.salary123,kate.salary9}),
({george.label11},{kate.experience})}

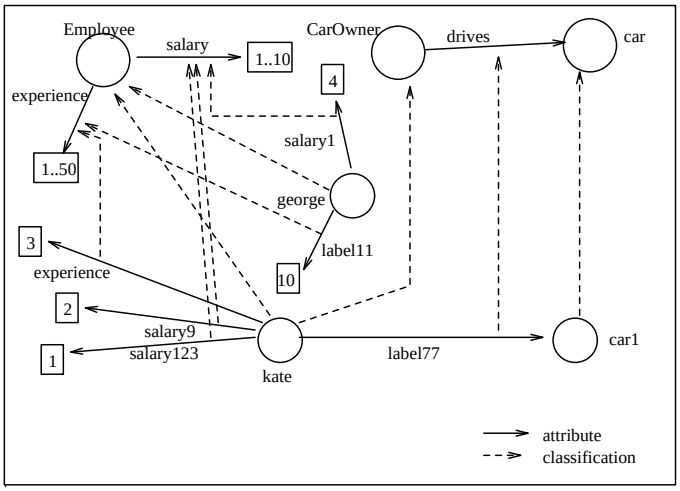

Figure 8: A schema for comparisons of attribute tokens

Thus, george and kate are not comparable in terms of car ownership since the relevant attributes are not applicable to both of them.

7. General Forms of Distance Metrics

According to the previous analysis, we can propose four general functional forms for distance metrics. These forms correspond to entity-classes, attribute classes, entity tokens and attribute tokens.

i. Entity classes,

D(c1,c2)= F(d1(Id(c1),Id(c2)), d2(In(c1),In(c2)), d3(Isa(c1),Isa(c2)),d4(AC'),d5(AT'))

ii. Attribute classes, D(ac1,ac2)=

D(ac1,ac2)= G(d1(Id(ac1),Id(ac2)),d1(Id(From(ac1)),Id(From(ac2))),d2(In(ac1),In(ac2)),
d3(Isa(ac1),Isa(ac2)),d4(AC'),d5(AT'),D(o1(Id(To(ac1))),o1(Id(To(ac2)))))

iii. Entity tokens,

D(t1,t2)= F'(d1(Id(t1),Id(t2)), d2(In(t1),In(t2)), d5(AT'))

iv. Attribute Tokens,

D(at1,at2) = G'(d1(Id(at1),Id(at2)),d1(Id(From(at1)),Id(From(at2))),
d2(In(at1),In(at2)),d5(AT'),D(o1(Id(To(at1))),o1(Id(To(at2)))))

where

* *F,F'* are aggregate functions differing only in the number of arguments measuring the overall distances of entities

* *G,G* are aggregate functions differing only in the number of arguments measuring the overall distances of attributes

* *d1* is the identity function over the system identifiers

* *d2* denotes a distance metric over the classification abstraction

* *d3* denotes a distance metric over the generalization abstraction

* *d4,d5* are aggregate functions reflecting distances over the attribution abstraction

* *AC'* is the set of the corresponding attribute classes for the objects in hand defined according to formula (1)

* *AT'* is the set of the corresponding attribute tokens for the objects in hand defined according to formula (2)

These general functional forms can be viewed as abstractions of various comparison methods proposed for analogical reasoning and can be specialized in at least the following ways or combinations of them:

i. the contribution of the partial distances to the overall measure ;

ii. the degree of recursiveness of the partial distances d2, d3, d4 and d5 ; and

iii. the salience associated with the elementary feature-distances of the aggregation dimension.

We believe that such a specialization must take into account two basic aspects depending on the application context. The first is a pragmatic consideration suggesting a trade off between the quality of the final estimate and the computational cost for obtaining it. The second is a semantic consideration, which relates to the actual usage of the offered abstractions in particular domains and the effect of this usage to their pairwise orthogonality which is not always valid.

8. Conclusions and Issues For Further Research

In this paper we presented a qualitative framework for computing similarity, within the context of analogical software reuse. We argued that similarity measures must be devised according to a set of principles, distilled from requirements of the analogical reasoning process.

We also developed a systematic comparison basis and a general distance model for similarity computation over a particular representational notation for objects. Further research aims at the precise definition of partial distance metrics according to the semantics of the relevant abstractions, the contribution of these metrics to the overall similarity estimate the quantification of the salience of the various object features and issues of computational efficiency.

A prerequisite for the integration of an adequately instantiated distance model into tools supporting analogical software reuse, will be its experimental validation in a relevant context.

9. References

[1] Bareiss R., King J., Similarity Assessment in Case-Based Reasoning, DARPRA Workshop on Case-Based Reasoning, 1988

[2] Bergadano F., et al., Learning Two-Tiered Descriptions of Flexible Concepts: The Poseidon System, Machine Learning 8, 1988

[3] Biggerstaff T., Richter C., Reusability Framework, Assessment and Directions IEEE Software, March 1987

[4] Burton B. et. al., The Reusable Software Library, IEEE Software, July 1987

[5] Cohen P., Kjeldsen R., Information Retrieval by Constrained Spreading Activation in Semantic Networks, Information Processing and Management, 23(4), 1987

[6] Constantopoulos P., et. al., The Software Information Base-Selection Tool Integrated Prototype, ITHACA.FORTH.91.e2.#3

[7] Curtis B., Cognitive Issues in Reusing Software Artifacts, Software Reusability, Addisson-Wesley, 1987

[8] Pietro-Diaz R., Freeman P., Classifying Software for Reusability, IEEE Software, January 1987

[9] Esposito F. et. al. Classification in Noisy Environments Using a Distance Measure Between Structural Symboic Descriptions, IEEE Transactions on Pattern Analysis and Machine Intelligence, 14(3), 1992

[10] Fischer G., Cognitive View of Reuse and Redesign, IEEE Software, July 1987

[11] Gennari J., et al., Models Of Incremental Concept Formation, Artificial Intelligence 40, 11-61, 1989

[12] Gentner D., Structure-Mapping: A Theoretical Framework for Analogy, Cognitive Science 7, 1983

[13] Gentner D., Analogical Inference and Analogical Access, Analogica, Armand Prieditis(ed.), Morgan Kaufmann Pub., 1988

[14] Gentner D., Finding the Needle: Accessing and Reasoning From Prior Cases, DARPRA Workshop on Case-Based Reasoning, 1988

[15] Golding A., Rosenbloom P., Combining Analytical and Similarity-Based CBR, DARPRA Workshop on Case-Based Reasoning, 1988

[16] Greiner R., Abstraction-Based Analogical Inference, Analogical Reasoning, Kluwer Academic Publishers, 1988

[17] Hall R., Computational Approaches to Analogical Reasoning: A Comparative Analysis, Artificial Intelligence 39, 1989

[18] Holyoak K., Koh K., Surface and Structural Similarity in Analogical Transfer, Memory and Cognition 15(4), 1987

[19] Kolonder J., Judging Which is the "Best" Case for a Case-Based Reasoner, DARPRA Workshop on Case-Based Reasoning, 1988

[20] Kowalski H.J., Topological Spaces, Academic Press, 1965

[21] Maiden N., Sutcliffe A., Exploiting Reusable Specifications through Analogy, Communications of the ACM, 35(4), 1992

[22] Maiden N., Sutcliffe A., Analogical Matching for Specification Reuse, Proceedings of the 6th Annual Conference on Knowledge-Based Software Engineering, IEEE Computer Society Press, 1991

[23] Michalski R., Learning from Observation: Conceptual Clustering, Machine Learning: an AI approach, Vol I, Morgan Kaufmann Pub., 1986

[24] Miriyala k., Harandi M., The role of analogy in Specification Derivation, Proceedings of the 6th Annual Conference on Knowledge-Based Software Engineering, IEEE Computer Society Press, 1991

[25] Mylopoulos J., et. al., Telos: Representing Knowledge About Information Systems, ACM Transactions on Information Systems, 8(4), 1990

[26] Novick L., Analogical Transfer: Processes and Individual Differences, Analogical Reasoning, Kluwer Academic Publishers, 1988

[27] Rosch E., et all., Basic Objects in Natural Categories, Gognitive Psychology 8, 1976

[28] Russel S., Analogy By Similarity, Analogical Reasoning, Kluwer Academic Publishers, 1988

[29] Schwanke R., An Intelligent Tool for Re-engineering Software Modularity, ICSE-13, Austin Texas, 1991

[30] Seifert C., Hammond K., Why There Is No Analogical Transfer, DARPRA Workshop on Case-Based Reasoning, 1988

[31] Sjoberg L., A Cognitive Theory of Similarity, Goteborg Psychological Reports, Number 10, Volume 2, 1972

[32] Smith E.E., Concepts and Induction, Foundations of Cognitive Science, A Bradford Book, The MIT Press, 1989

[33] Step R., Michalski R., Conceptual Clustering: Inventing Goal-Oriented Classifications of Structured Objects, Machine Learning: an AI approach, Morgan Kaufmann Pub., 1986

[34] Tversky A., Features of Similarity, Psychological Review, 44(4), July 1977

[35] Vassiliou Y. et. al., Technical Description of the Software Information Base, ITHACA.FORTH.91.E2.#2,1990

[36] Wegner P., The Object-Oriented Classification Paradigm, Research Directions in Object-Oriented Programming, ed. Shriver, Wegner, The MIT Press

[37] Wegner P., Zdonic S., Inheritance as an Incremental Modification Mechanism or What Like is and Isn't Like, Proceedings of the European Conference on Object-Oriented Programming(ECOOP'88), Lecture Notes in Computer Science, 1988

[38] Winston P., Learning and Reasoning by Analogy, Communications of the ACM, 23(12), December 1980

[39] Wrobel S., Concept Formation in Man and Machine: Fundamental Issues, Workshop on Concept Formation in Man and Machine, GMD, 1991

[40] Novel Approaches to Theories Underlying Requirements Engineering, Project Description, Esprit 6353, Technical Annex, 1992

Similarity for Analogical Software Reuse: A Conceptual Modelling Approach

George Spanoudakis and Panos Constantopoulos

Institute of Computer Science
Foundation of Research and Technology – Hellas

Abstract

We present our approach to defining similarity between software artifacts and discuss its potential exploitation in software reuse by analogy. We first establish properties of similarity which support its role in retrieving and mapping software descriptions. Then we develop a systematic basis for comparison within a fairly general conceptual modelling framework, whereby comparable elements of the descriptions of software objects and corresponding similarity criteria are identified. Finally, a general form of distance metrics for the computation of similarity measures is defined.

1. Introduction: The analogical based approach to software reuse

The reuse based approach to software development has been realized as a form of analogical problem solving, where a reasoner, the software engineer, familiar with a variety of source cases(i.e development histories and systems) attempts to transfer properties, relations and solutions from them to a target system under development[22,24].

This realization brings reuse in a ground rich in theoretical predictions and experimental evidence from a variety of scientific disciplines(AI, Psychology, Cognitive Science and Philosophy). A ground which offers a generic process schema of analogical reasoning, predicting 4 distinct stages:

1. the *retrieval* of a set of source cases given a target description ;

2. the *mapping* of knowledge from some of the retrieved sources to the target on the basis of appropriate analogies between them, established at the same stage ;

3. the *evaluation* of the transferred knowledge in terms of consistency and pragmatic utility ; and

4. the *consolidation* of the final outcome into structures of knowledge, supporting similar forms of reasoning in the future.

It is reasonable to believe that research into reuse can benefit from this conceptual framework. In fact, the solutions to the operational problems of reusability(i.e retrieval, comprehensibility, modifiability and composition[3]) offered so far, can be reviewed in this framework, enhanced with new ones motivated by it and ideally integrated into a generic and theoretically plausible methodology according to the predictions of analogical reasoning.

Adopting this approach, we focus on the problem of defining a quantitative similarity relation between descriptions of software artifacts so as to promote their analogical reuse. The potential of similarity has been pointed out in both the literature relevant to reuse[4,6,7,8] and the literature relevant to analogical reasoning[1,15,28].

As far as analogical reasoning is concerned, the role of similarity is clear during the stage of retrieval. Either as a fine evaluator of source cases initially selected by some coarser search-method(e.g constrained spreading of activation[5]) or as a basic retrieval mechanism, similarity can provide effective trial hints to the stage of mapping that follows retrieval. However, the effectiveness of similarity depends on its very definition and whether it avoids certain undesired effects that are already known(see the discussion about the importance of features below).

In this paper, we present our on-going work on the concept of similarity. The final aim is the formulation of a computational model of similarity, in a way consistent with properties inherent to relevant kinds of human processing(as they are revealed by studies in Psychology[31,34]) and dealing effectively with

reuse-oriented forms of analogical reasoning. To this end we distinguish two main stages. The first stage aims at a generic method for computing similarity from object oriented qualitative descriptions, while the second deals with specific issues of software reuse.

This work is carried out within project NATURE of the ESPRIT Basic Research Action[40], which pursues the development of requirements engineering methodologies and is connected with ongoing work on knowledge organization for software reuse(Software Information Base[35]) carried out within ESPRIT project ITHACA.

2. A general review of Similarity

In general terms, similarity is described as a relation determined by some flexible comparison between the distinct constituents of two entities (i.e situations, cases, natural or nominal-kind objects)[23,29,31,34,36]. From a quantitative viewpoint, the result of this comparison may be interpreted in two basic ways:

i. as a measure of closeness in some abstract space[23,34] and

ii. as a probability that the objects under comparison, would resemble each other even if their possibly missing constituents were considered as well[9,28].

Both the representation of objects and the nature of the comparisons are approached differently in the literature.

As far as the representaion of objects is concerned there exist two basic approaches. The first assumes the existence of a predefined set of features adequate for representing any object within a given domain of discourse. Consequently an object is viewed as possessing a subset of these features by giving appropriate values to them. The second approach assumes that objects are representable through their classification into a set of mutually disjoint and exhaustive classes.

Regarding the comparisons, we can also distinguish two basic forms:

(1) the *exact-matching form*, which results into three sets of object-features:

$$\textit{i. the common features}: S1 = F(o1) \cap F(o2)$$

$$\textit{ii.the distinctive features of the first object}: S2 = F(o1) - F(o2) \quad \textit{and,}$$

$$\textit{iii.the distinctive features the second object}: S3 = F(o2) - F(o1)$$

Given these three sets, the evaluation of similarity, in general has one of the following functional forms:

i. F(S1, S2, S3) that takes into account both the common and the distinct features of the objects. Typical instances of this form are[34]:

$$\textit{a.the ratio model}: \frac{wf(S1)}{af(S2) + bf(S3)} \quad \textit{and,}$$

$$\textit{b.the contrast model}: wf(S1) - af(S2) - bf(S3)$$

ii. F(S1) that takes into account only the common features of the two objects. In this category, we can classify the frequency model of similarity[31]:

$$f(|C|, |U|)$$

where

- |C| is the cardinality of the common class of two objects
- |U| is the cardinality of the entire universe of objects and,
- f is a function increasing in |C| and decreasing in |U|

(2) the *distance-based form*, which assumes the existence of metrics measuring the distance of features according to the types of their domains(i.e binary, nominal, ordinal or linear domains). Typical such metrics include:

i. the identity function for nominal domains and

ii. the absolute difference function for totally ordered domains[2,9]:

$$d(v1,v2) = |order(v1) - order(v2)|$$

Apart from the previous issues of object representation and comparison forms, two more aspects of similarity should be pointed out, namely the context effect and the symmetric nature of the relation.

A variety of experiments lead to the conclusion that the similarity between objects is affected by their common context. This effect has been attributed either to the dependence of the salience of the various object features on this context or to variations on the cardinality of the common class that the objects under comparison belong to, which in turn is affected by context changes(see the frequency perspective above).

A more arguable issue is the symmetric nature of similarity. In fact, reported evidence supports both the non-symmetric and the symmetric views[34]. Generally speaking the non-symmetric view regards similarity as the result of a directed comparison, which designates two basic roles for the objects involved. These are the role of the *subject* and the role of the *referent*. The features of the subject are assumed to have a salience greater than the features of the referent. Therefore, differences relevant to the features of the subject are emphasized more than differences relevant to the features of the referent. On the other hand, the symmetric view does not distinguish between these different roles and thus is does not differentiate the salience of features according to them. Hence, the direction of the comparison, does not affect the weights of the differences, resulting in equal similarity estimates.

3. TELOS: A representational framework for Similarity Evaluation

In this section we briefly review the structural part of the TELOS data model[25,35]. This model will be the basis for the representation of objects in our framework. Our choice is primarily motivated by the generic nature of this model, which in turn subsumes an entire family of object oriented data models, at least with respect to their structural elements. In addition, TELOS has been acknowledged as language for describing software artifacts[6].

TELOS provides three basic abstractions, namely classification, generalization and attribution.

Classification defines an infinite dimension along which objects can be classified as Tokens, or Simple Classes, or Meta classes and so on. This is accomplished through the declaration of an object as an instance of the built-in disjoint classes that distinguish between the successive levels of classification(i.e Token class, S_Class class, M1_Class class, M2_Class class and so on). In addition an object can be classified as an instance of other user-defined classes thus being capable of instantiating the attributes attached to these classes.

Objects which are classes(i.e they have not been classified as Tokens) can be generalized and/or specialized into other classes, through Isa-relations. In TELOS, these relations have a set inclusion semantics and are restricted only between classes of the same classification level. A class C may have one or more direct superclasses, whose superclasses are also considered as superclasses of C (Isa transitivity). The Isa relations in general form a directed acyclic graph, the edges of which allow the inheritance of attributes from the superclasses to the subclasses(strict and multiple inheritance).

The attribution mechanism allows the attachment of attributes to objects. It must be emphasized that attributes in TELOS are also objects. Thus they can be classified into attribute classes, generalized, specialized and have attributes of their own. The term attribute captures both classes and tokens of attributes. Attribute classes may be single-valued or multi-valued (i.e an object may have one or more attributes, which are instances of the same attribute class) and necessary or not.

In addition to its modelling according to these abstractions, a TELOS object is associated with two unique identifiers. The first of them is known as *system identifier* and has the role of a surrogate. It is generated automatically by the system, and is associated with a particular object during its entire life-cycle. A system identifier can not be reused for another object and thus it supports a unique identification. The second type of identifiers is the *logical name*. A logical name is attached to an object by its creator and accomplishes logical references to it. Although it may be changed during the life-cycle of an object, a logical name ought to be system-wide unique. The logical names of attribute objects are composed of the logical names of their possessing objects and logical referents attached directly to them, called *labels*. For instance, the logical name of an attribute object with label X, possessed by an object with name O is O.X. O.X must be system-wide unique, while X must be unique only within the scope of object O.

According to the previous overview, four basic forms of TELOS objects can be distinguished:

1. Entity Tokens: ETi = [Id(i), In(i), A(i)]
2. Attribute Tokens: ATi = [Id(i), Id(From(i)), In(i), A(i), Id(To(i))]
3. Entity Classes: ECj = [Id(j), In(j), Isa(j), A(j)]
4. Attribute Classes: ACj = [Id(j), Id(From(j)), In(j), Isa(j), A(j), Id(To(j))]

where

i. Id(x) is a system identifier denoting the object Ox

ii. In(x) is a set of system identifiers denoting the classes of the object Ox

iii. Isa(x) is a set of system identifiers denoting the direct superclasses of the class-object Ox

iv. A(x) is a set of system identifiers denoting the attributes attached to the object Ox

v. From(x) denotes the object possessing the attribute Ox and,

vi. To(x) denotes the object pointed to by the attribute Ox

Two more issues regarding the topology of the Isa-graphs and the inherent assumptions of the inheritance mechanism of TELOS are of interest from the similarity perspective.

In TELOS, an Isa-graph generally consists of M disjoint subgraphs, where M is the number of classification levels used in the particular schema(see figure 1). The Isa graphs are disjoint because of the restriction of the Isa relations only between classes of the same classification level.

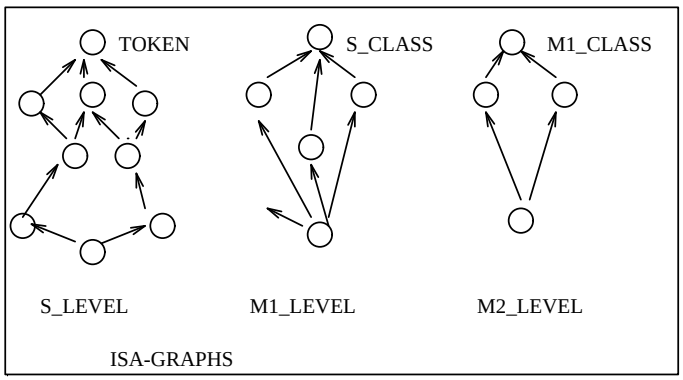

Figure 1: Distinct TELOS Isa-graphs

The set-inclusion semantics of the Isa-relation make it a partial-order relation[20]. Moreover, at each level of classification TELOS provides a most general class, whose extension contains all the objects of the lower level (see Token, S_Class and M1_Class in figure 1). Therefore the extensions of the rest of the classes at the same classification level are subsets of its extension and due to the set-inclusion semantics of the Isa-relations, these classes are regarded as subclasses of it. Thus two classes in an Isa subgraph will always have a common superclass, no matter if this is the built-in most general class for the relevant level of classification.

As far as the strict multiple inheritance mechanism is concerned, the key underlying hypothesis is that the identity of attribute-labels along an Isa-path implies the semantic identity of the relevant attribute-objects. This, due to the strict inheritance, has the consequence that attributes in subclasses, having the same labels with attributes in their superclasses, can only specialize them(see figure 5).

4. A scenario of exploiting similarity in requirements reuse

To illustrate the role of similarity and analogical reasoning in software reuse, we present a hypothetical scenario of incremental system specification in which requirements of existing information systems are reused.

These requirements are organized according to a three-level conceptual schema. The higher level of it includes models for describing the so called[25] Subject, System, Usage and Development Worlds of an Information System. Roughly speaking, these worlds correspond to the application domain, the functionality, the operational environment and the development process of an information system respectively. The next level consists of abstract specifications of system types, built according to these four perspectives(e.g LibrarySystem in figure 2). The lower level comprises descriptions of specific systems, classified into one or more of the mentioned abstract specifications(e.g MunicipalLibrary1 in figure 2).

Rather than presenting user-system dialogs, we give a rough process description, including steps where similarity and analogical reasoning are expected to enable reuse. Also, we refer to other generic reasoning and searching techniques, which could be used for carrying out particular actions of specification.

Our scenario is by no means complete and is based on an interactive view on reuse, where a software engineer uses a set of tools supporting reuse (e.g an information base, a browser, a similarity computation module, retrieval methods, integrity checking) which do not fully automate the entire process nor do they assume a unique reuse strategy(e.g templates approach).

Let us assume that an engineer wants to specify the requirements for a Car Rental Agency system and proceeds through the following steps:

STEP 1:

The software engineer asks for a requirements description, named as CarRentalSystem. The search fails and consequently he is prompted to choose one of the two complementary models of requirements descriptions, namely, the *SubjectWorldModel* and the *SystemWorldModel* to start the specification of the new system.

Assuming that the SystemWorldModel is selected, the system presents it to the engineer(i.e SystemWorldModel in the Schema of figure 2) and prompts him to specify the top-level *transactions* and *data classes* of the new system by names and other semantic terms.

STEP 2:

In response to this prompt, the engineer specifies the following transactions and data classes, accompanied by appropriate terms.

a. Transactions:

1. rentACar (hire, lend)

b. Data Classes

1. Car (Vehicle, automobile)

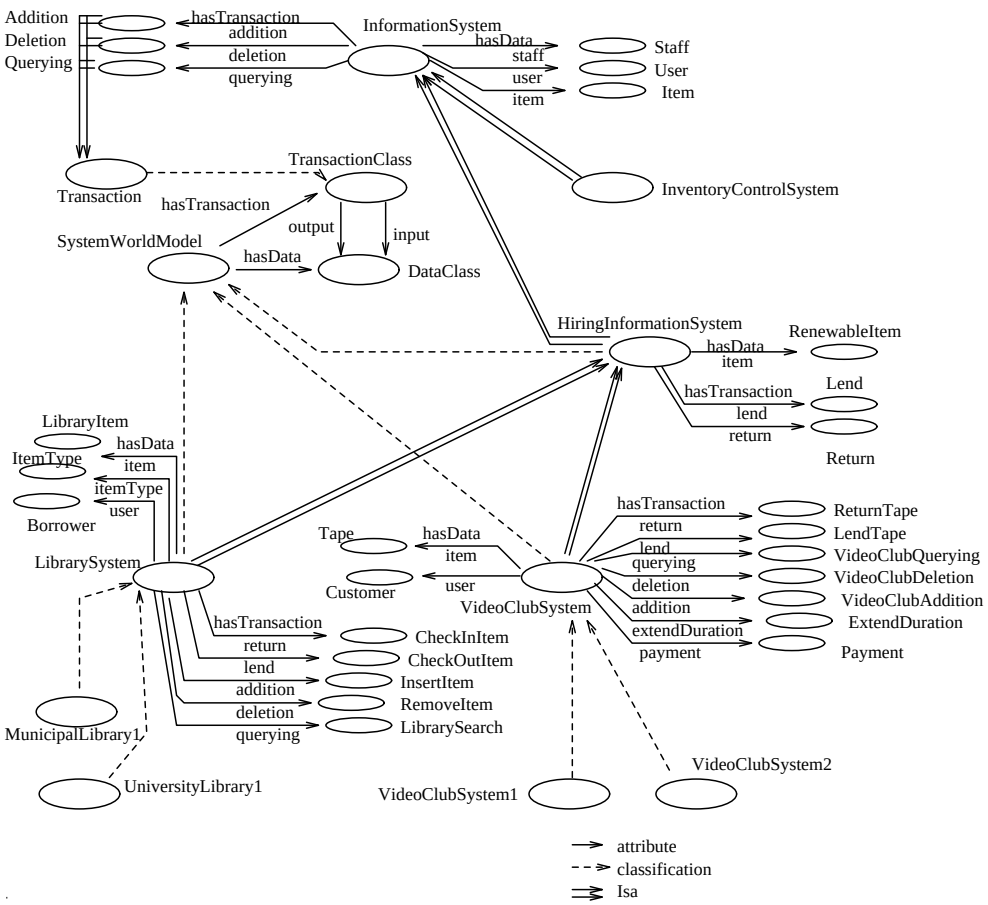

Figure 2: A model for Requirements Descriptions

2. returnACar () 2. Customer (user)

3. insertANewCar(file-in,insert,add)

4. removeACar(delete)

STEP 3:

Using these names and the associated terms as search-hints, the system starts a constrained spreading of activation within the abstract SystemWorld descriptions, to allocate those including any of these items. This process results in two SystemWorld abstract specifications, namely:

1. the LibrarySystem and,

2. the VideoClubSystem

Moreover, in order to help the engineer, to understand these abstractions[21], the system presents him the two most characteristic(prototypical) instances of each along with their degree of prototypicality:

1.1 UniversityLibrary1 0.62

1.2 MunicipalLibrary1 0.55

2.1 VideoClubSystem1 0.78

2.2 VideoClubSystem2 0.61

The prototypicality of instances with respect to some of their classes is computed as their average similarity to the rest instances of the extensions of these classes[34].

STEP 4:

The engineer focuses on the VideoClubSystem1 and the UniversityLibrary1 (see figure 3).

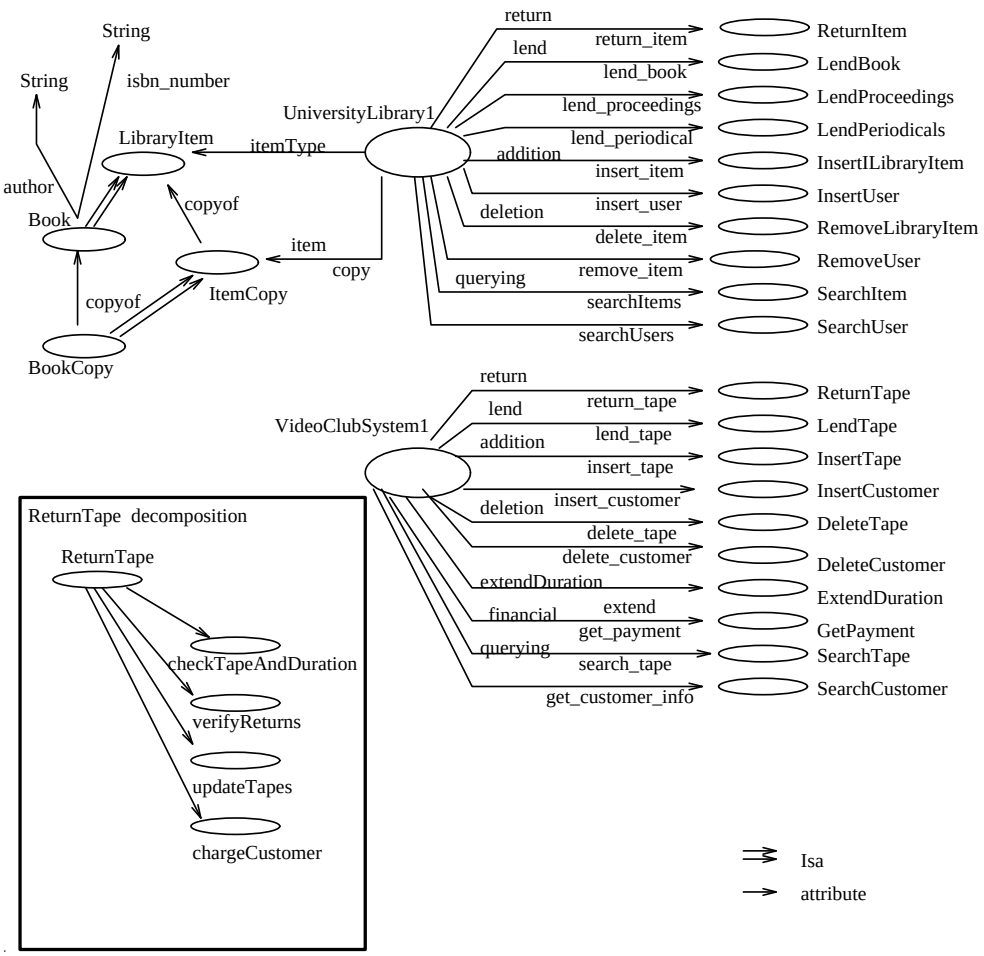

Figure 3: Specific Requirements Descriptions

After browsing through them, he proceeds in a scavenging mode of copying elements of these systems into the on-going specification of the CarRentalSystem. This results in the following transactions:

lend
 rentVehicle: RentVehicle

return
 returnVehicle: ReturnVehicle

addition
 insertVehicle: InsertVehicle
 insertCustomer: InsertNewCustomer

remove
 removeVehicle: RemoveVehicle
 removeCustomer: RemoveCustomer

querying

createReport: CreateVehicleReport

searchForVehicle: SearchForVehicle

getCustomerRecord: GetCustomerRecord

STEP 5:

Next, the engineer specifies the input and the output data classes of the transactions of the CarRentalSystem which in the case of the ReturnVehicle transaction are the following:

input	output
input1: Customer.customer_id	output1: RentedVehiclesList
input2: Extension_period	output2: VehicleBillings
input3: ReturnVerification	output3: Receipt
input4: BillingVerification	

These inputs and outputs correspond to his generic view about the functionality of this transaction. ReturnVehicle is assumed to retrieve a list of the vehicles rented by a customer, given his identifier, display them and get a response on whether the customer is going to return the vehicle or extend the renting period, bill him, verify which of the rented vehicles are to be paid immediately and which will be charged to the customer's open balance and finally print a receipt for the immediate payments.

Once the input and the output for all the top-level transactions have been specified, the engineer puts more specific similarity questions about each of the top-level transactions of the system under development. His underlying assumption is that similarity in terms of inputs and outputs, specified so far may imply functional similarity with prospects for reuse. He also, decides to restrict the search space to transactions of systems classified as only as LibrarySystems or VideoClubSystems. In response to his queries the system retrieves the two most similar transactions to each of the transactions of the CarRentalSystem:

i. InsertVehicle

1. insertLibraryItem1 (of UniversityLibrary1) 0.6
2. insertNewTape (of VideoClubSystem1) 0.52

...

ii. ReturnVehicle

1. checkInTape (of VideoClubSystem1) 0.5
2. returnLibraryItem (of MunicipalLibrary1) 0.44

The engineer browses through these transactions. In the case of the checkInTape and the returnLibraryItem, their decompositions yield the following subtransactions:

1. checkInTape	2. returnLibraryItem
1.1 checkTapeAndDuration	2.1 validateBook
1.2 verifyReturns	2.2 validateLoanDuration
1.3 updateTapes	2.3 updateReturns
1.4 chargeCustomer	

Motivated by these results, he specifies the ReturnVehicle transaction as follows:

1. CheckRentedItems
2. VerifyReturns
3. ExtendLendingDuration

4. UpdateVehicles
5. Billing
6. PaymentVerification
7. ChargeClient
8. IssueReceipt

He also introduces the input and the output data classes of these subtransactions. This more detailed specification prompts him to ask again for the similarity of the ReturnVehicle to other existing transactions, in order to check whether their exact functional specifications could be further reused. The next system response includes the following:

1. checkInTape (of VideoClubSystem1) 0.73
2. returnLibraryItem (of MunicipalLibrary1) 0.32

STEP 6:

Based on these results the engineer further describes the functionality of the ReturnVehicle according to the details of the checkInTape. However, the output of the subtransaction IssueReceipt (i.e Receipt) is not similar to any output data class, of transactions associated with any Library or VideoClub systems. Hence, he decides to enlarge his search space to include the entire extension of the InformationSystem abstraction. This results to the retrieval of a similar transaction from an InventoryControlSystem, namely the transaction IssueInvoice(similarity: 0.57, output: Invoice), which is being successfully reused in terms of its functional details.

STEP 7:

Having completed the definition of the transaction ReturnVehicle and the specification of the input and the output data classes of its subtransactions, the engineer attempts to classify it as an instance of the class Transaction(see figure 2). However, he fails due to the subtransaction Billing. This subtransaction contradicts a constraint associated with all transactions, demanding that the input data classes of all the subtransactions of a parent transaction X, must either be output data classes of some other subtransaction of X or input data classes to X.

In the case of Billing the input data class VerifiedItems defined as:

Class VerifiedItems in List with
element
item: RentedVehicle

violates the previous constraint. The system, after checking the relevant constraint, informs the engineer about the inconsistency. Consequently, he asks the system for output data classes of other subtransactions of the ReturnVehicle, that are similar to the VerifiedItems class and gets the following response:

1. RentedVehicles 0.9
2. UnverifiedItems 0.6
3. Billings 0.3

Then he notices that the RentedVehicles, which is the output data class of the subtransaction VerifyReturns, should be the input to the Billing and changes the name VerifiedItems to RentedVehicles.

...

STEP N:

Once the software engineer has completed the System World Specification of the CarRentalSystem, he wants to classify it under the appropriate SystemWorld abstraction. Thus, he asks for the system for an optimal classification suggestion. The system responds by estimating the average similarity of the new specification, with those belonging to the extensions of the available abstractions and supplying a ranked

list as follows:

1. HiringInformationSystem 0.68
2. InformationSystem 0.62
3. VideoClub 0.43
4. LibrarySystem 0.3
5. InventoryControlSystem 0.07

The engineer decides to classify it under the HiringInformationSystem.

As the previous scenario illustrates, similarity and analogical reasoning are viewed as basic techniques, that complement other forms of searching and reasoning(e.g deduction, spreading activation, integrity checking) in dealing with the operational problems of reuse. Similarity can support analogical reuse in at least the following ways:

i. as a fine search method for retrieving potentially reusable artifacts(steps 5,6)

ii. by providing a basis for prototypicality estimates and selections of instances of abstractions(step 3)

iii. by providing a criterion of replacement of sources of integrity violations, whenever their retraction is not desirable(step 7)

iv. by providing a criterion of classification to consolidate cases for future analogical reuse(step N)

5. Principles of Similarity Computation

In this section we formulate a set of principles underlying the computation of similarity. These principles reflect requirements imposed by the context of analogical reasoning and take into account the outlined representational framework. Some of them are also justified by psychological predictions about the ways that humans realize similarities between natural and nominal-kind objects.

The Principle of Ontological Uniformity

In general objects may correspond to different levels of abstraction. This in the notation of TELOS is basically expressed through the classification level of an object. Roughly speaking, objects may refer to atomic real world entities(i.e Tokens), to abstracted classes of atomic entities(i.e S_Classes), to models enabling the generation of such abstractions(i.e M1_Classes) and so on. In this object space, it would be senseless to attempt comparisons between entities of different ontologies. Any such comparison would be ad hoc and therefore not semantically interpretable. Hence, inter-object comparisons are restricted according to the following principle of **Ontological Uniformity**:

(P1) Similarity comparisons are only valid between objects of the same classification level

This principle seems to be arguable in the context of concept formation[11] but this is only due to a confusion. The underlying reason is that the term similarity is used to describe comparisons between objects of successive levels of classification. However, these comparisons have a different objective. They are a means of deciding whether or not the lower, in terms of classification, object could be classified as an instance of the higher one. In fact, they end up in a true/false result, according to whether or not the observed attribute values of the lower object belong to the predicted domains for the relevant attributes in the higher one. But this is different from expressing the resemblance between two values or two attribute domains.

It must also be pointed out, that when humans generate concepts the predicted comparisons do not involve directly the objects of the higher level. In some cases, they are carried out implicitly through one or more prototypes of the concept in hand(figure 4, case a)[32,39]. In other cases, the higher object corresponds to an exemplar-based representation of the relevant concept (figure 4, case c) and thus it does not really belong to a higher level of classification. Finally, there exist cases where the comparison is performed in an aggregate fashion, taking into account the resemblances of the object to be classified with all the other

known instances of the concept(figure 4, case b)[27,34].

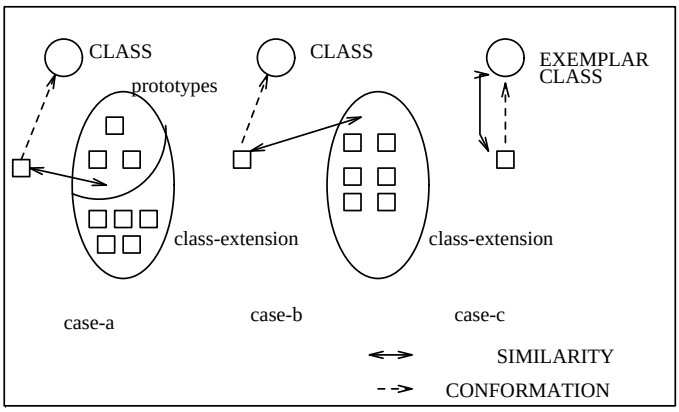

Figure 4: Comparisons between Classes and Instances

The principle of partially uniform representation

A critical issue for the evaluation of similarity is the uniformity of the underlying object representation, which, if valid, dictates that any object can be described through a fixed predefined set of features. Such a hypothesis is a good approximation of reality in narrow, well-defined and rather mature domains. But in general these characteristics are rare in analogical reasoning and software reuse. In these areas, the contexts involved include objects from different domains(e.g diverse application areas of software), described via non-standard and possibly not well-understood properties, and usually instantiating models reflecting diverse viewpoints[6,10,25,35].

As a result it is reasonable to assume only partially uniform representations of objects. An example of what partially uniform representation means is given in figure 5.

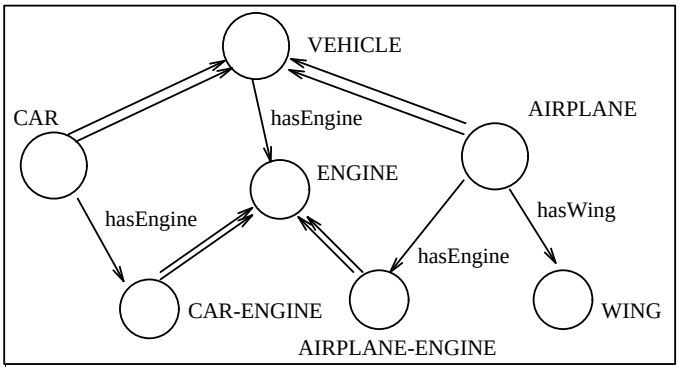

Figure 5: Isa-graphs and Inheritance

In that figure the two different vehicle types, the airplane and the car are uniform in sharing the possession of an engine(despite the different engine types) and diverse with respect to the presence of the wings.

Partial uniformity is still reasonable under the presence of domain specific methods for deducing attribute equivalences[1], since distinct but equivalent attributes will be treated as common.

Partial uniformity also suggests the restriction of comparisons only between the common attributes of two objects. This is justified by a global view of the analogical reasoning process. During the early stage of retrieval, two basic types of errors may occur. The first is the rejection of sources that could later enable sound and useful analogical transfers(*errors of type A*). The second captures the case where candidate sources are retrieved, though not enabling any acceptable transfer(*errors of type B*). Errors of type B increase the overall computational cost of the process, since they enlarge the set of the source cases to be considered for transfer during the mapping stage, yet they are recoverable during justification. On the contrary it would be impossible to recover from errors of type A. This limitation makes errors of type A more important than errors of type B.

Consequently, comparisons between the uncommon attributes of two objects should not be attempted. Such comparisons necessitate ad hoc matchings between ordinary and artificial values, that can only increase the dissimilarity of the involved objects, while not contributing in a positive way to any transfer.

The previous analysis is summarized into the principle of **Partially Uniform Representation**, stating that:

(P2) Objects have only partially uniform representations. The non-uniform parts of their representation must be excluded from similarity comparison

The normalization principle

Distance measures subsume exact matching(see the fourth axiom below) by giving finer evaluations of differences. Thus, they must be preferred as a basis for estimating similarity. Central to our approach is the idea that an overall distance metric D can be defined as an aggregate function over partial metrics, devised in accordance with the semantics of the three distinct abstractions composing the object descriptions.

D is a mapping of the form:

D: O x O ----> R

where O is the set of all objects in the context of similarity estimation, R is the set of Reals and D obeys the known metric axioms[20]:

(1) D(x,y) >= D(x,x) = 0

(2) D(x,y) = D(y,x)

(3) D(x.z) <= D(x,y) + D(y,z) and.

(4) D(x,y) = 0 ==> x = y

Similarity has been viewed as a monotonically decreasing function f of distance D [28]. A basic issue is whether f must be defined over absolute or relative distance measures. Certain characteristics of the abstractions over which the absolute distances must eventually be defined, strongly favor relative measures. Generally speaking, the elementary absolute distance metrics must be defined over Isa-graphs of classes or over domains of attributes that may be nominal, partially ordered or totally ordered(i.e linear). In all but the case of the nominal domains, which suggests an identity distance function, the relevant domains are not expected to have equal widths[28] and the Isa-graphs are expected to have diverse coarsenesses. Therefore, normalizations of absolute metrics are necessary for filtering out differences arising due to coarser and finer representations.

This requirement is summarized in the following principle of **Normalization**:

(P3) Similarity(S) is a monotonically decreasing function of a normalized Distance measure(Dr)

A trivial consequence of this principle is that:

Dr = 1 <==> S = 0 and

Dr = 0 <==> S = 1

The importance of features in Similarity Evaluation

The distinct features of objects(i.e relationships and attributes) have a different impact in both the evaluation of similarity and the success of analogical transfer. Usually referred to as salience, this impact can be attributed to three basic factors, namely the domain dominance, the pragmatic utility and the classificatory significance of a feature. Salience is critical since its overestimation or underestimation may prevent analogical transfer or enable erroneous analogies[18,26,30]. Consequently it must be carefully analyzed and quantified before taken into account by similarity estimates.

i. The domain dominance of features

The domain dominance factor of salience reflects the causality of a feature within a particular domain. Causality is determined by the dependence of values or even the presence of other features on a particular feature and has a direct effect on the mapping stage of analogical reasoning. Since dominance contributes to transfer of entire constellations of knowledge pieces, which is the ultimate purpose of analogical reasoning, this effect is obvious.

There exist two different points of view to the evaluation of the domain dominance. The first one[15,28,37] realizes it as presence in the antecedent parts of implications or in integrity constraints expressing inter-feature dependencies for a particular domain. The second approach relies on purely syntactic aspects of object representations. First, it distinguishes between relations and other representational elements(e.g identities and attributes) and then it recognizes only the former ones as being dominant[12,13].

Certainly, unless we want to hypothesize explicit and direct information about dominance, we must rely on syntactic elements for its evaluation. This approach could be justified on the basis of particular observations about the encoding process that result into an object knowledge base. In studies dealing with the surface/structure paradox of analogical reasoning[14], it has been pointed out that the encoding of a certain domain around dominant or structural features and not around superficial ones is subject to the training and the expertise of the encoder in the particular domain. In our case it is correct to assume that both the training and the expertise conditions of encoding are satisfied. Knowledge bases, used for analogical reasoning or software reuse are not normally developed by novices and furthermore they undergo constant improvements(e.g reverse engineering) during their life-cycle(see [7] for a similar viewpoint). Consequently domain dominant features will be prevalent in them.

Hence the principle of **the Domain Dominant Schema**:

(P4) The domain dominance of a feature can be determined from the schema of the relevant object knowledge base.

Here, the term schema has its usual database sense (i.e it captures the conceptual schema of the object descriptions).

ii. The pragmatic utility of features

The causal relations of features to goal attainment in the process of analogical reasoning designate what is called the pragmatic utility of features. Generally speaking, features are distinguished into those predicting the applicability of some method for achieving a goal, those predicting the success or failure of a certain solution, those that describe unusual outcomes of solutions and those which are totally irrelevant to goals and solutions. The first three of these associations result into a high pragmatic utility.

However, it is really difficult to distinguish between pragmatic utility and domain dominance in the absence of explicit information about the former(e.g Goal Dependency Graphs in [33]). Furthermore, even in the presence of relevant information, it is reasonable to assume that this information would lead to similar conclusions with those drawn only from the evaluation of domain dominance. In practice, domain dominance correlates with pragmatic utility since general causality is likely to lead to goal relevancy. Therefore, without sacrificing any important amount of information, we can rely on domain dominance in estimating pragmatic utility.

Thus, the principle of **the Pragmatic Utility Subsumption** suggests:

(P5) The pragmatic utility of a feature can be approximated by its domain dominance

In the particular case of software reuse, this principle is further justified by proposals relevant to the organization of software repositories. These proposals indicate conceptual schemas, which capture pragmatically important features(e.g correspondence links in the SIB[6], abstract domain classes in [22]).

iii. The classificatory Significance of features

The classificatory significance refers to the contribution of a feature in the production of classification schemas, being optimal with respect to certain criteria. Examples of such criteria are the inferential power of the schema in predicting missing attribute-values for instances in concept formation and the optimization of the precision measure in information retrieval.

In this framework, the sensitivity to errors of type A suggests a reasonable optimality criterion for a classification schema: the minimization of the probability of errors of type A. Consequently we can define the classificatory significance of a feature through the following principle of **Classification Optimality**:

(P6) The classificatory significance of a feature depends on its ability to produce classification schemas minimizing the probability of errors of type A in the analogical reasoning process.

6. Establishing A Basis for Comparison

The classification, the generalization and the attribution together with the two distinct types of identifiers are really expressive in describing objects but also impose the problem of deciding which pairs of elements included in object-descriptors, could be compared with each other.

The different nature of these constructs suggests that the various elements of an object representation can only be compared with elements of the same type. Otherwise we would not have a clear semantic basis for defining an appropriate distance metric. Therefore, we adopt the following top-level pairs of comparison, on the basis of our representational model:

1. Comparisons between Classifications:

   In(x) <---> In(y) {for any pair of objects (x,y)}

2. Comparisons between Generalizations:

   Isa(x) <---> Isa(y) {for any pair of classes (c1,c2)}

3. Comparisons between Attributes:

   A(x) <---> A(y) {for any pair of objects (x,y)}

4. Comparisons between System Identifiers:

   Id(x) <---> Id(y) {for any pair of objects (x,y)}

   Id(From(x)) <---> Id(From(y)) {for any pair of attributes (x,y)}

   Id(To(x)) <---> Id(To(y)) {for any pair of attributes (x,y)}

These top-level comparisons must be further elaborated in the case of the attributes, which can be utilized in representing properties and/or relations with different semantics (e.g Car.partOf, Car.mileage). A distinction is made between attribute classes and attribute tokens, due to the inheritance property employed only by the former and the different role of the attribute labels in them.

i. Comparisons between attribute classes

The inheritance mechanism associated with the Isa relations interferes with the attribution abstraction. Recall that attribute classes with the same labels along an Isa-path are perceived as being semantically identical and consequently only specializations of their specifications are permitted because of the strict inheritance model. Note also, that in the case of unordered classes(with respect to Isa relations) label

equality has no further implications.

This property of labels can be also employed in determining the valid comparisons between attribute classes. In fact, the attributes of a class can be distinguished into three main categories:

1 the attributes originally defined in this class ;

2. the inherited yet refined attributes ; and

3. the inherited but not refined attributes.

Assuming the following one to one mappings:

i. o1: I --> O

ii. o2: L --> O

iii. id1: L --> I

where O is the universal set of objects, I is the set of the system identifiers and L is the set of the logical names, we can define these attribute categories as follows:

Definition 1: The *Modifier* of a class C, M[C] is defined as the set of the labels of all the attributes included in its definition:

$$M[C]=\{\, x \mid id1(C.x) \in o2(C).A \,\}$$

where o.A is the set of the system identifiers for the attributes of the object o

The term modifier has been introduced in[37].

Definition 2: The set of the *unordered superclasses* of a class C, US[C]is defined as:

$$US[C]=$$

$$\{\, x \mid (id1(x) \in o2(C).isa^*) \text{ and } (not\ (exists\ y : (id1(y) \in o2(C).isa^*) \text{ and } (id1(x) \in o2(y).isa^*))) \,\}$$

where o.isa* is the transitive closure of the superclasses of the object o.

Definition 3: The *Intension* of a class C, INT[C] is defined as:

$$a.\ INT[C]=M[C] \text{ if } US[C] \text{ is empty}$$

$$b.\ INT[C]=\{\, x \mid (x \in M[C])\ or\ (x \in \bigcup_{j \in US[C]} INT[j]) \,\}\ otherwise$$

Definition 4: The *Horizontal Extension* of a Class C, with respect to a subset S={S1,...,Sm} of its superclasses, HE[C,S] is defined as:

$$HE[C,S]=$$

$$\{\, x \mid (x \in INT[C]) \text{ and } ((not(x \in \bigcup_{i \in S} Int[i]))) or (Forall\ i\ in\ S : (x \in INT[i]) \rightarrow (not\ (id1(i.x) \in o2(C.x) isa^*)))) \,\}$$

Thus, the horizontal extension of a class C with respect to some of its superclasses S1,..., Sm includes the labels of the attribute classes of C, which are not associated with any attribute classes of S1,...,Sm, if they are superclasses of the former(see a somehow different definition in [37]).

Definition 5: The *Vertical Replacement* of a class C, with respect to any of its superclasses S, VR[C,S] is defined as:

$$VR[C,S]=\{\, x \mid (x \in INT[C] \text{ and } (x \in INT[S]) \text{ and } (id1(S.x) \in o2(C.x).isa^*)\}$$

In words, the vertical replacement of a class with respect to one of its superclasses, includes the labels of its attributes which also specialize attributes with the same label inherited from this superclass.

Definition 6: The *Comparison Basis for Attribute Classes* of two classes C1 and C2, CBAC[C1,C2] is defined as:

$$CBAC[C1,C2]=$$

$$\{\, x \mid (id1(x) \in o2[C1].isa^*) \text{ and } (id1(x) \in o2[C2].isa^*) \; and \; (\; not \; (exists \; y : (id1(y) \in o2[C1].isa^*) \text{ and } (id1(y) \in o2[C2].isa^*) \text{ and } (id1(x) \in o2(y).isa^*)))\}$$

Since the Isa-relation is only a partial ordering and a class may have more than one superclasses, two classes C1 and C2 in general may have more than one pairwise unordered minimal common superclases. These will be the elements of their Comparison Basis for attribute classes(CBAC set). The CBAC set is guaranteed to be non-empty, since the most general TELOS built-in class at each level of classification is a superclass of all the classes at that level(see figure 1).

Given the previous definitions, we can distinguish between two categories of attributes that can be compared, with respect to two classes and their Comparison Basis:

1. The *Common Vertical Replacement*, which is a set of attribute pairs (c1.x, c2.y) defined as:

$$CVR[c1,c2,S]= \bigcup_{si \in CBAC[c1,c2]} CVR[c1,c2,Si]$$

where

$$CVR[c1,c2,Si]=\{\, (c1.x,\; c2.y) \mid (x \in VR[c1,Si]) \text{ and } (y \in VR[c2,Si]) \text{ and } (x = y)\}$$

The common vertical replacement includes the commonly inherited attributes, which have also been refined within the classes of consideration.

2. The *Unique Vertical Replacements* UVR[c1,c2,Si] and UVR[c2,c1,Si],are defined with respect to each of the superclasses Si in the comparison basis of the classes c1,c2 according to the following definition:

$$UVR[x1,x2,Si]=\{\, (x1.x,\; Si.y) \mid (x \in VR[x1,Si]) \text{ and } (not \; (x \in VR[x2,Si])) \text{ and } (x = y)\}$$

These sets pair attributes commonly inherited by superclasses, having identical labels but which are specialized in exactly one of the classes in hand.

The union of the Common and the Unique Vertical Replacements, defined as:

$$AT[C1,C2]=$$

$$CVR[C1,C2,CBAC[C1,C2]] \bigcup_{Si \in CBAC[C1,C2]} \{ UVR[C1,C2,Si] \} \bigcup_{Si \in CBAC[C1,C2]} \{ UVR[C2,C1,Si] \} \quad (1)$$

includes the pairs of attribute classes that must be taken into account in the estimation of the distance of two objects with respect to the aggregation abstraction. Note that attribute classes that belong to the horizontal extensions of the involved classes are not further compared, since they are not applicable to both of them(see the principle of the partially uniform representation).

ii. Comparisons between attribute tokens

In the case of attribute classes, comparison-pairs were formulated by using the Isa-graph for determining the Comparison Basis and the deduction of the semantic role identity from the equality of the attribute labels. However, both these features are either absent or do not have the same implications for attribute tokens. First, the Isa relation is not defined for atomic objects. Moreover, attribute tokens can not be further instantiated. Thus, their labels serve only as references within the relevant object-scopes and are meaningless outside them(e.g "mike_car" label in figure 6).

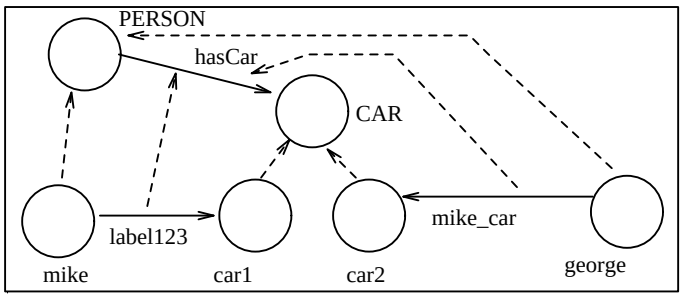

Figure 6: Entity objects and attribute tokens

Actually, the semantics of attribute tokens are expressed by their attribute classes. Therefore, their semantic identity can be detected from the labels of these classes. For instance, in the 6th figure we could compare the attribute token *mike.mike_car* with the attribute token *george.label123* due to their common classification under the attribute class *Person.hasCar* and despite their different labels.

In conclusion, attribute tokens yield comparison-pairs as follows:

Initially, the comparison basis for attribute tokens(CBAT) of two objects Oi, Oj is defined as:

$$CBAT[Oi,Oj]=Oi.in \cap Oj.in \text{ if } this\ intersection\ is\ not\ empty \quad otherwise,$$

$$CBAT[Oi,Oj]=$$

$$\left\{ x \mid (id\,1(x) \in CC(Oi,Oj)) \text{ and } (not\ (exists\ y : (id\,1(y) \in CC(Oi,Oj)) \text{ and } (id\,1(x) \in o\,2(y).isa^*))) \right\}$$

where CC(Oi,Oj) is the set of the common classes of Oi, Oj, defined as:

$$CC(Oi,Oj)= Oi.in^* \cap Oj.in^*$$

where:

$$Ok.in^*= \bigcup_{i \in Ok.in} i.isa^*$$

Then the set of the attribute classes which are applicable to both Oi and Oj(CAC) is defined as:

$$CAC[Oi,Oj]= \bigcup_{i \in CBAT[Oi,Oj]} INT[i]$$

Finally, we define the comparison-pairs consisting of sets of attribute tokens as:

$$CAT[Ok,Or]=$$

$$\left\{ \left(\left\{ X1,...,Xm \right\}, \left\{ Y1,...,Yn \right\} \right) \mid (forall\ i,j : (id\,1(Xi) \in Ok.A) \text{ and } (id\,1(Yj) \in Or.A) \text{ and } \right.$$

$$(exists\ z : (z\ \varepsilon\ CAC[Ok, Or]) \text{ and } (id\,1(z)\ \varepsilon\ o\,2(Ok.Xi).in^*) \text{ and } (id\,1(z)\ \varepsilon\ o\,2(Or.Yj).in^*)))\Big\} (2)$$

This definition reflects the possibility of objects instantiating relevant attribute classes by more than one attribute tokens.

iii. An example of attribute comparisons

According to the schema of the figure 7 we can form the following sets with respect to the classes UniversityDoctor(ud) and Professor(p):

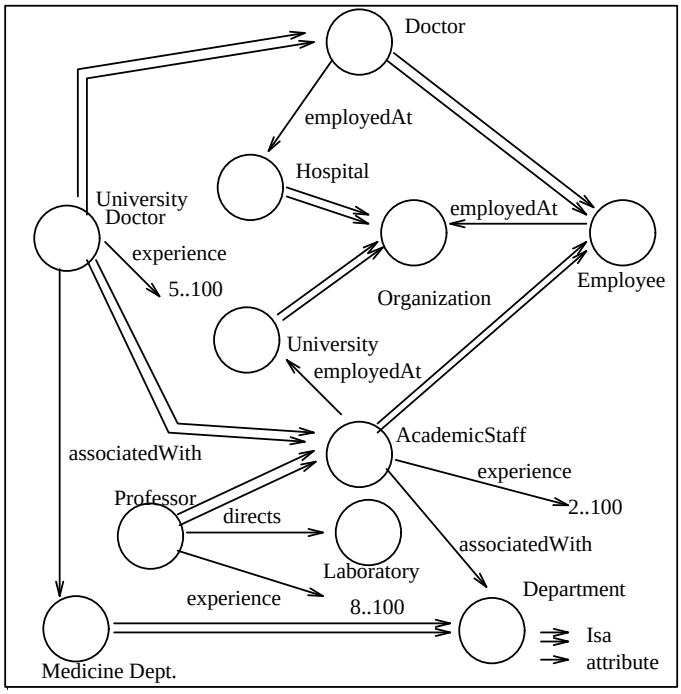

Figure 7: A schema for comparison of attribute classes

CBAC[ud,p] = { AcademicStaff }

HE[p,AcademicStaff] = { directs }

HE[ud,AcademicStaff] = { employedAt FROM doctor }

VR[ud,AcademicStaff] = { associatedWith, experience }

VR[p,AcademicStaff] = { experience }

CVR[p,ud,{AcademicStaff}] = {(p.experience, ud.experience)}

UVR[ud,p,AcademicStaff] = {(ud.associatedWith,AcademicStaff.associatedWith)}

UVR[p,ud,AcademicStaff] = {}

Note that when two attributes that have the same label are inherited from two distinct superclasses, they must be disambiguated with a FROM clause within the scope of the inheriting class, as in the case of the class Professor and the attribute employedAt.

Also, in the schema of the figure 8 the comparison pairs of the attribute tokens are:

CBAT[george, kate]={employee}

CAC[george,kate]={employee.salary, employee.experience}

CAT[george,kate]={({george.salary1},{kate.salary123,kate.salary9}),
({george.label11},{kate.experience})}

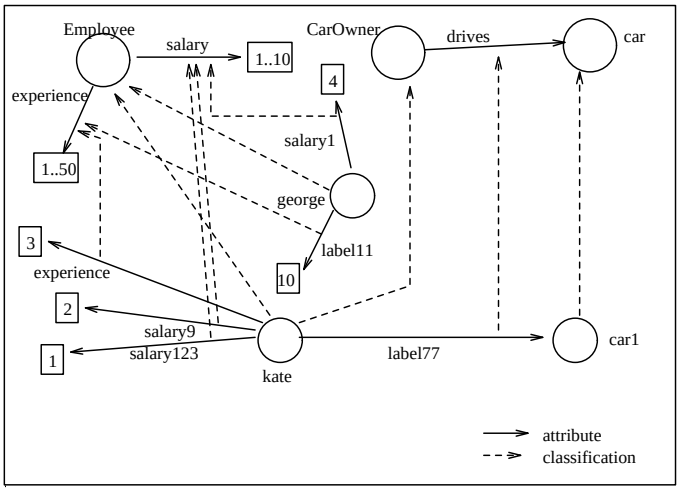

Figure 8: A schema for comparisons of attribute tokens

Thus, george and kate are not comparable in terms of car ownership since the relevant attributes are not applicable to both of them.

7. General Forms of Distance Metrics

According to the previous analysis, we can propose four general functional forms for distance metrics. These forms correspond to entity-classes, attribute classes, entity tokens and attribute tokens.

i. Entity classes,

D(c1,c2)= F(d1(Id(c1),Id(c2)), d2(In(c1),In(c2)), d3(Isa(c1),Isa(c2)),d4(AC'),d5(AT'))

ii. Attribute classes, D(ac1,ac2)=

D(ac1,ac2)= G(d1(Id(ac1),Id(ac2)),d1(Id(From(ac1)),Id(From(ac2))),d2(In(ac1),In(ac2)),
d3(Isa(ac1),Isa(ac2)),d4(AC'),d5(AT'),D(o1(Id(To(ac1))),o1(Id(To(ac2)))))

iii. Entity tokens,

D(t1,t2)= F'(d1(Id(t1),Id(t2)), d2(In(t1),In(t2)), d5(AT'))

iv. Attribute Tokens,

D(at1,at2) = G'(d1(Id(at1),Id(at2)),d1(Id(From(at1)),Id(From(at2))),
d2(In(at1),In(at2)),d5(AT'),D(o1(Id(To(at1))),o1(Id(To(at2)))))

where

* *F,F'* are aggregate functions differing only in the number of arguments measuring the overall distances of entities

* *G,G* are aggregate functions differing only in the number of arguments measuring the overall distances of attributes

* *d1* is the identity function over the system identifiers

* *d2* denotes a distance metric over the classification abstraction

* *d3* denotes a distance metric over the generalization abstraction

* *d4,d5* are aggregate functions reflecting distances over the attribution abstraction

* *AC'* is the set of the corresponding attribute classes for the objects in hand defined according to formula (1)

* *AT'* is the set of the corresponding attribute tokens for the objects in hand defined according to formula (2)

These general functional forms can be viewed as abstractions of various comparison methods proposed for analogical reasoning and can be specialized in at least the following ways or combinations of them:

i. the contribution of the partial distances to the overall measure ;

ii. the degree of recursiveness of the partial distances d2, d3, d4 and d5 ; and

iii. the salience associated with the elementary feature-distances of the aggregation dimension.

We believe that such a specialization must take into account two basic aspects depending on the application context. The first is a pragmatic consideration suggesting a trade off between the quality of the final estimate and the computational cost for obtaining it. The second is a semantic consideration, which relates to the actual usage of the offered abstractions in particular domains and the effect of this usage to their pairwise orthogonality which is not always valid.

8. Conclusions and Issues For Further Research

In this paper we presented a qualitative framework for computing similarity, within the context of analogical software reuse. We argued that similarity measures must be devised according to a set of principles, distilled from requirements of the analogical reasoning process.

We also developed a systematic comparison basis and a general distance model for similarity computation over a particular representational notation for objects. Further research aims at the precise definition of partial distance metrics according to the semantics of the relevant abstractions, the contribution of these metrics to the overall similarity estimate the quantification of the salience of the various object features and issues of computational efficiency.

A prerequisite for the integration of an adequately instantiated distance model into tools supporting analogical software reuse, will be its experimental validation in a relevant context.

9. References

[1] Bareiss R., King J., Similarity Assessment in Case-Based Reasoning, DARPRA Workshop on Case-Based Reasoning, 1988

[2] Bergadano F., et al., Learning Two-Tiered Descriptions of Flexible Concepts: The Poseidon System, Machine Learning 8, 1988

[3] Biggerstaff T., Richter C., Reusability Framework, Assessment and Directions IEEE Software, March 1987

[4] Burton B. et. al., The Reusable Software Library, IEEE Software, July 1987

[5] Cohen P., Kjeldsen R., Information Retrieval by Constrained Spreading Activation in Semantic Networks, Information Processing and Management, 23(4), 1987

[6] Constantopoulos P., et. al., The Software Information Base-Selection Tool Integrated Prototype, ITHACA.FORTH.91.e2.#3

[7] Curtis B., Cognitive Issues in Reusing Software Artifacts, Software Reusability, Addisson-Wesley, 1987

[8] Pietro-Diaz R., Freeman P., Classifying Software for Reusability, IEEE Software, January 1987

[9] Esposito F. et. al. Classification in Noisy Environments Using a Distance Measure Between Structural Symboic Descriptions, IEEE Transactions on Pattern Analysis and Machine Intelligence, 14(3), 1992

[10] Fischer G., Cognitive View of Reuse and Redesign, IEEE Software, July 1987

[11] Gennari J., et al., Models Of Incremental Concept Formation, Artificial Intelligence 40, 11-61, 1989

[12] Gentner D., Structure-Mapping: A Theoretical Framework for Analogy, Cognitive Science 7, 1983

[13] Gentner D., Analogical Inference and Analogical Access, Analogica, Armand Prieditis(ed.), Morgan Kaufmann Pub., 1988

[14] Gentner D., Finding the Needle: Accessing and Reasoning From Prior Cases, DARPRA Workshop on Case-Based Reasoning, 1988

[15] Golding A., Rosenbloom P., Combining Analytical and Similarity-Based CBR, DARPRA Workshop on Case-Based Reasoning, 1988

[16] Greiner R., Abstraction-Based Analogical Inference, Analogical Reasoning, Kluwer Academic Publishers, 1988

[17] Hall R., Computational Approaches to Analogical Reasoning: A Comparative Analysis, Artificial Intelligence 39, 1989

[18] Holyoak K., Koh K., Surface and Structural Similarity in Analogical Transfer, Memory and Cognition 15(4), 1987

[19] Kolonder J., Judging Which is the "Best" Case for a Case-Based Reasoner, DARPRA Workshop on Case-Based Reasoning, 1988

[20] Kowalski H.J., Topological Spaces, Academic Press, 1965

[21] Maiden N., Sutcliffe A., Exploiting Reusable Specifications through Analogy, Communications of the ACM, 35(4), 1992

[22] Maiden N., Sutcliffe A., Analogical Matching for Specification Reuse, Proceedings of the 6th Annual Conference on Knowledge-Based Software Engineering, IEEE Computer Society Press, 1991

[23] Michalski R., Learning from Observation: Conceptual Clustering, Machine Learning: an AI approach, Vol I, Morgan Kaufmann Pub., 1986

[24] Miriyala k., Harandi M., The role of analogy in Specification Derivation, Proceedings of the 6th Annual Conference on Knowledge-Based Software Engineering, IEEE Computer Society Press, 1991

[25] Mylopoulos J., et. al., Telos: Representing Knowledge About Information Systems, ACM Transactions on Information Systems, 8(4), 1990

[26] Novick L., Analogical Transfer: Processes and Individual Differences, Analogical Reasoning, Kluwer Academic Publishers, 1988

[27] Rosch E., et all., Basic Objects in Natural Categories, Gognitive Psychology 8, 1976

[28] Russel S., Analogy By Similarity, Analogical Reasoning, Kluwer Academic Publishers, 1988

[29] Schwanke R., An Intelligent Tool for Re-engineering Software Modularity, ICSE-13, Austin Texas, 1991

[30] Seifert C., Hammond K., Why There Is No Analogical Transfer, DARPRA Workshop on Case-Based Reasoning, 1988

[31] Sjoberg L., A Cognitive Theory of Similarity, Goteborg Psychological Reports, Number 10, Volume 2, 1972

[32] Smith E.E., Concepts and Induction, Foundations of Cognitive Science, A Bradford Book, The MIT Press, 1989

[33] Step R., Michalski R., Conceptual Clustering: Inventing Goal-Oriented Classifications of Structured Objects, Machine Learning: an AI approach, Morgan Kaufmann Pub., 1986

[34] Tversky A., Features of Similarity, Psychological Review, 44(4), July 1977

[35] Vassiliou Y. et. al., Technical Description of the Software Information Base, ITHACA.FORTH.91.E2.#2,1990

[36] Wegner P., The Object-Oriented Classification Paradigm, Research Directions in Object-Oriented Programming, ed. Shriver, Wegner, The MIT Press

[37] Wegner P., Zdonic S., Inheritance as an Incremental Modification Mechanism or What Like is and Isn't Like, Proceedings of the European Conference on Object-Oriented Programming(ECOOP'88), Lecture Notes in Computer Science, 1988

[38] Winston P., Learning and Reasoning by Analogy, Communications of the ACM, 23(12), December 1980

[39] Wrobel S., Concept Formation in Man and Machine: Fundamental Issues, Workshop on Concept Formation in Man and Machine, GMD, 1991

[40] Novel Approaches to Theories Underlying Requirements Engineering, Project Description, Esprit 6353, Technical Annex, 1992