Analogical Reuse of Requirements Specifications: A Computational Model

George Spanoudakis
Department of Computer Science, City University
Northampton Square, London EC1V 0HB, UK
e-mail: gespan@cs.city.ac.uk

Panos Constantopoulos
Department of Computer Science, University of Crete
and
Institute of Computer Science,
Foundation for Research and Technology-Hellas
P.O Box 1385, GR 711 10 Heraklion, Crete, Greece
e-mail: panos@ics.forth.gr

Abbreviated Title: Analogical Reuse of Requirements Specifications

Abstract

Specifications of requirements for new software systems can be revised, refined or completed in reference to specifications of requirements for existing similar systems. Although realized as a form of analogical problem solving, specification by reuse is not adequately supported by available computational models for detecting analogies. This is chiefly due to the following reasons: (1) It is assumed that specifications are expressed according to the same specification model and in a uniform representation scheme. (2) Additional information is needed for the detection of analogies, which is not contained in the specifications. (3) Performance scales poorly with the complexity of specifications. This paper presents a computational model for detecting analogies, which addresses these issues to a certain extent. The application of the model in the specification of requirements by analogical reuse is demonstrated through an example, and its sensitivity to the representation of specifications is discussed. Finally, the results of a preliminary empirical evaluation of the model are reported.

1. Introduction

Specifications of requirements for new software systems can be revised, refined or completed in reference to specifications of existing, similar systems. Successful *specification of requirements by reuse* can lead to significant productivity gains in software development. This is because it can improve the completeness and correctness of specifications, thus reducing the probability of improper design and implementation decisions, with substantial delays and adverse cost effects in software projects. In particular, specification by reuse can be a useful requirements engineering practice, when co-operation with the customers of software systems is not effective or possible due to their lack of experience or domain knowledge(e.g. co-operation with new personnel(Hofmann H. 1993)). Specifications reuse may also lead to reuse of software designs and code fragments (Tracz W. 1990, Constantopoulos P. et al. 1993).

Reuse has been realized as a form of analogical problem solving in the sense that knowledge about existing systems embedded in specifications is transferred to specifications of new systems after identifying analogies between them (Miriyala K. and Harandi M. 1991, Maiden N. and Sutcliffe A. 1992, Maiden N. and Sutcliffe A. 1994). In our view this approach requires powerful mechanisms for the computational detection of analogies, which exhibit the following properties:

(1) *They support multiple specification models:* The modelling practices and tools employed in different projects for requirements specification are often based on different specification

models. According to a recent survey(Fuggetta A. 1993) only commercial upstream CASE tools support 8 different specification models(e.g. data flow diagrams, entity relationship diagrams, state-transition diagrams, object-oriented specification languages, etc.).

(2) *They support incomplete specifications expressed in non-uniform representation schemes:* Requirements are acquired and specified via an informal communication process between agents with different expertise (i.e. software engineers and customers) in a distributed fashion (Finkelstein A. et al. 1992). At this stage of software development, objectives may be unclear, partially realized, or even conflicting (Reubenstein H. and Waters R. 1991). As a result, requirements specifications tend to employ ambiguous terms, be expressed in non uniform ways and be incomplete.

(3) *They do not need information beyond that contained in the specifications:* The acquisition and incorporation of information in specifications solely for the sake of possible future reuse is an extra task for developers, the execution of which can hardly be ensured, especially in view of the more important concurrent concern of requirements capture.

(4) *They scale efficiently with specification complexity:* The need to specify requirements from multiple perspectives over software systems, including their application domain, internal system characteristics and usage (Mylopoulos J. et al. 1990) can lead to very complex specifications. Matching mechanisms have to be computationally efficient for coping with such complexity.

The available computational models of analogy are weak with regard to the above requirements. In this paper, we present a model(referred to as *similarity model* in the following) developed to address them, demonstrate its employment for supporting specification by reuse and discuss the results of a preliminary empirical evaluation.

The rest of the paper is organized as follows. After an overview of the similarity model in Section 2, we demonstrate how it detects analogies between specifications through an example, in Section 3. In Section 4, we discuss how the detection of analogies could be used in refining, revising and/or completing specifications. In Section 5, we discuss practical considerations regarding the application of the model and in Section 6 we present a preliminary empirical evaluation. In Section 7, we review the related work and finally in Section 8, we summarize our

approach and discuss some open research issues.

2. Similarity analysis of specifications

2.1 Representation of specifications: the Telos language

Telos(Mylopoulos J. et al. 1990), an object-oriented knowledge representation language, has been selected as the representation framework for similarity analysis. Telos supports three *semantic modelling abstractions*, namely *classification*, *generalization* and *attribution*(Hull R. and King R. 1987, Peckham J. and Maryanski F. 1988). It treats entities and attributes uniformly as objects with equal rights. Objects may belong to one or more classes, introducing the different kinds of attributes, used for building up their descriptions. Classes themselves are objects, which are classified under metaclasses, metaclasses are objects classified under metametaclasses and so on. Classes can be related through multiple generalization relationships (i.e. Isa relationships) which enforce the strict inheritance of attributes between them.

This representation framework can accommodate different models for specifying requirements (e.g. data-flow diagrams, ER-diagrams, state-transition diagrams). Such models are first defined as meta-models in the language and subsequently requirements specifications are described as instances of those. A concrete example of how to represent data-flow diagrams in Telos is given in Section 3. Representations of other specification models in Telos can be found elsewhere in the literature (e.g. *Object with Roles Model* in (Bellinzona R. et al. 1995), *state-transition diagrams* in (Vezerides C. 1992)). Telos is most suitable for requirements representation because it supports: (i) complex, nested and recursive structures, (ii) multiple and hierarchical organizations of objects, (iii) typed and usually attributed relations and (iv) groupings into multiple and possibly overlapping spaces providing different views on entities(Johnson L. et al. 1992).

2.2 Conceptual distances between specifications

The similarity between a pair of specifications, described as Telos objects, is defined as a function which takes values in the interval [0,1]. It is determined as a suitable inverse (e.g. negative exponential) of a conceptual distance between those specifications, taking non-negative real values.

The conceptual distance is intended to serve as a quantitative, comprehensive expression of the differences between the specifications, detected from their respective Telos representations. The conceptual distance of two objects is computed as an aggregate of partial distance metrics defined over the representation dimensions of the Telos language, namely identification, classification, generalization and attribution.

The identification distance distinguishes between identical and non-identical objects even if the latter are equal(i.e. they belong to exactly the same classes; they are generalized to exactly the same superclasses; and they have attributes with equal values).

The classification distance reflects differences expressed by the classification of two objects in different classes. The non-common classes of the objects have different contributions to the classification distance depending on the relative importance of the categorization they represent. The relative importance of each class is a decreasing function of the depth of the class in the taxonomies (generalization/specialization graphs) it participates.

The generalization distance reflects differences as captured by the assignment of non-common superclasses. Like non-common classes in the case of the classification distance, non-common superclasses have different contributions to the generalization distance which are similarly determined.

The attribution distance measures differences that occur at the level of attribute assignment. Computing this distance involves establishing a (partial) one-to-one correspondence between the attributes of the two objects, which yields corresponding and unique attributes for each object. The notion of corresponding attributes is an important and useful extension to the notion of common attributes: they are attributes that can be considered comparable regardless of whether they bear the same or different names and modelling. A necessary condition for two attributes to be comparable is that they belong to common attribute classes (*principle of semantic homogeneity*).

If more than one correspondence mappings is possible, a minimum distance criterion is applied to select one (called the *minimum distance isomorphism*). This is determined as an aggregate of pairwise distances of comparable attributes. Computing a pairwise attribute distance requires

treating the attributes as objects in their own right. Thus, the computation of the attribution distance is recursive: it generates optimal correspondence mappings at all the successive levels of decomposition along the attribution dimension. The importance of attributes in their contribution to the attribution distance is differentiated by means of a *salience* coefficient. This is determined on the basis of attribute *characterricity* (discrimination power in generalization taxonomies), *abstractness* (significance for the structure and behaviour of the owning object) and *determinance* (ability to determine the values of other objects). Approximate attribution distances can be computed by limiting the recursion to a prespecified depth.

The mathematical definitions of the distance measures are given in the Appendix. For full details see (Spanoudakis G. 1994a, Spanoudakis G. and Constantopoulos P. 1994a).

A major benefit of the above definition of conceptual distance is that it only relies on the representation dimensions of the specification model and the actual data (i.e. specifications) stored. It does not require subjective judgements of conceptual distances, which are hard to calibrate among individuals, or even to obtain, due to the inevitably low priority of such a task in a software engineer's agenda and the overhead it creates.

2.3 Tool support: the Semantic Index System

The similarity model has been implemented using the *Semantic Index System (SIS)*, a tool for representing, storing and retrieving objects described according to Telos (Constantopoulos P. and Doerr M. 1993). The SIS was developed to enable the construction of servers supplying reusable software artifacts to different stages of software development, known as *software information bases*(Constantopoulos P. et al. 1993). It provides a high-performance object-management subsystem capable of storing a large population of objects (10^6 is the latest tested order of magnitude) and querying them with performance almost independent of the size of the object base (the closure of a binary tree with 1024 nodes is retrieved in 2 secs on a SPARC station). An implementation of the similarity model has been integrated with the SIS (Spanoudakis G. 1994b), so the latter supports queries for detecting analogies between Telos objects.

3. Detecting analogies between specifications: an example

In the following, we demonstrate the detection of analogies between specifications by similarity

analysis using an example concerning the data-flow diagrams(DFDs) of Figure 1.

The first DFD, i.e. *LibrarySelfCheckOut*, describes the connection with the computer system of a library in order to search for some book and possibly reserve it for borrowing. The second, i.e. *ATMVehicleRental*, describes the connection with an automatic transaction machine(ATM) of a car renting agency for selecting and renting a car. As depicted by the dashed lines in Figure 1, these specifications are analogous in that they both comprise stages for: (1) identifying and validating the potential borrower of some resource; (2) selecting that resource; and (3) committing the borrowing activity. The analogy exists despite the fact that neither the sequence of the stages, nor their respective input and output information are exactly the same.

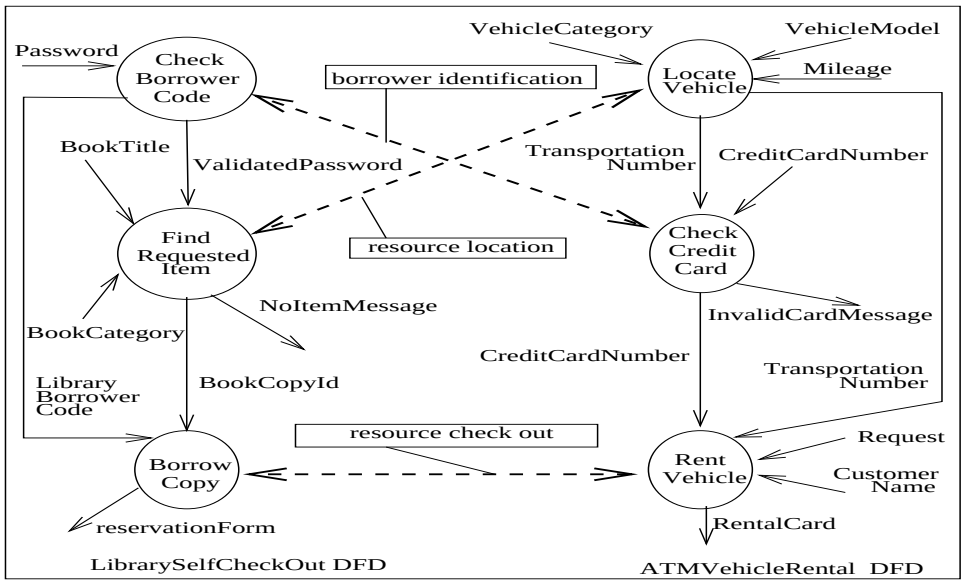


Figure 1: The analogy of distant electronic borrowing of resources

Both DFDs are represented as instances of a Telos meta-model for describing data-flow diagrams as shown in Figure 2. According to this meta-model, a DFD is described as an object classified under the class *ProcessClass*, which accepts as input and produces as output entities (cf. attribute classes *input* and *output* in Figure 2) and accesses entity stores(cf. attribute classes *readsFrom* and *writesOn*, in Figure 2). Processes can be decomposed into subprocesses(cf. attribute class *consistsOf* in Figure 2).

Computing the conceptual distance between *LibrarySelfCheckOut* and *ATMVehicleRental* boils down to searching for a minimum distance isomorphism between their attributes. By the

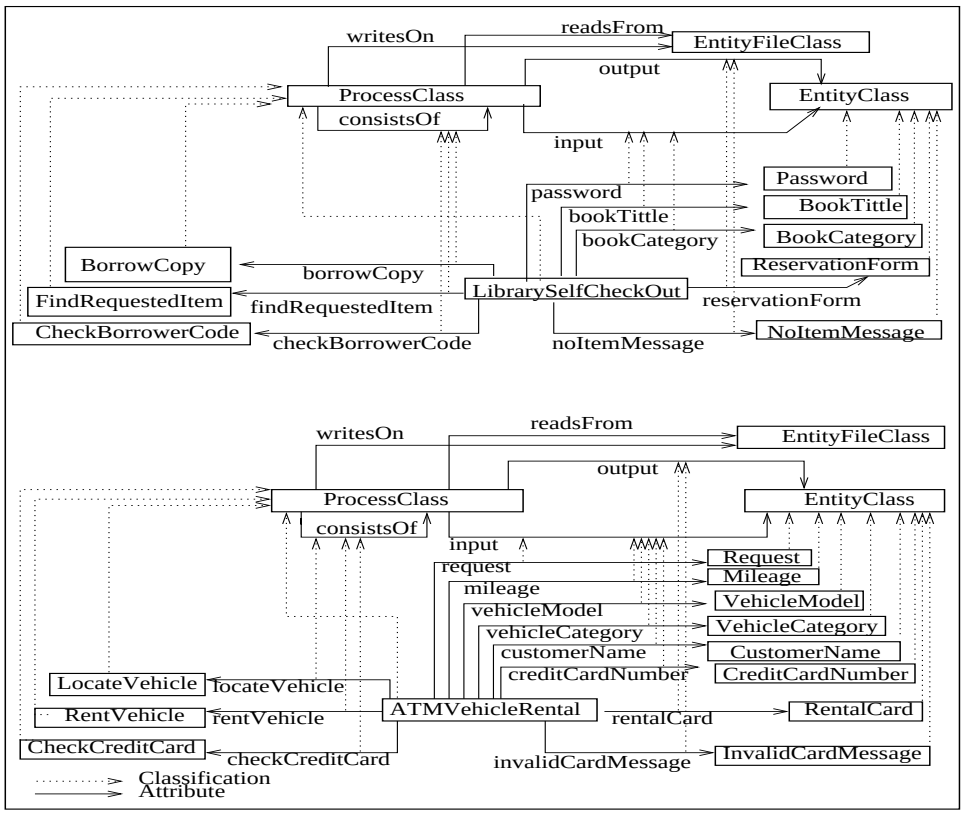


Figure 2: Data flow diagrams as instances of the DFD meta model

principle of semantic homogeneity, this search only considers mappings between attributes both classified under the attribute classes *input*, *output* or *consistsOf*.

These attributes aggregate the subprocesses of *LibrarySelfCheckOut* and *ATMVehicleRental* and can be mapped in six possible ways (cf. Figure 3). Computing their pairwise distances requires computing the overall distances between their value-objects, i.e. the subprocesses of the original DFDs (therefore computing their classification, generalization and attribution distances). These computations yield Mapping 3 in Figure 3, which maps *CheckBorrowerCode* on *CheckCreditCard*, *BorrowCopy* on *RentVehicle* and *FindRequestedItem* on *LocateVehicle*, as optimal. Along the way, the computation of the attribution distances recursively leads to selections of further optimal mappings between the attributes of these subprocesses. Let us consider the case of *FindRequestedItem* and *LocateVehicle*.

According to their specifications(cf. Figure 4), these subprocesses: (1) accept input information which enables the location of a resource(e.g. *BookTitle* , *VehicleModel*); (2) output a

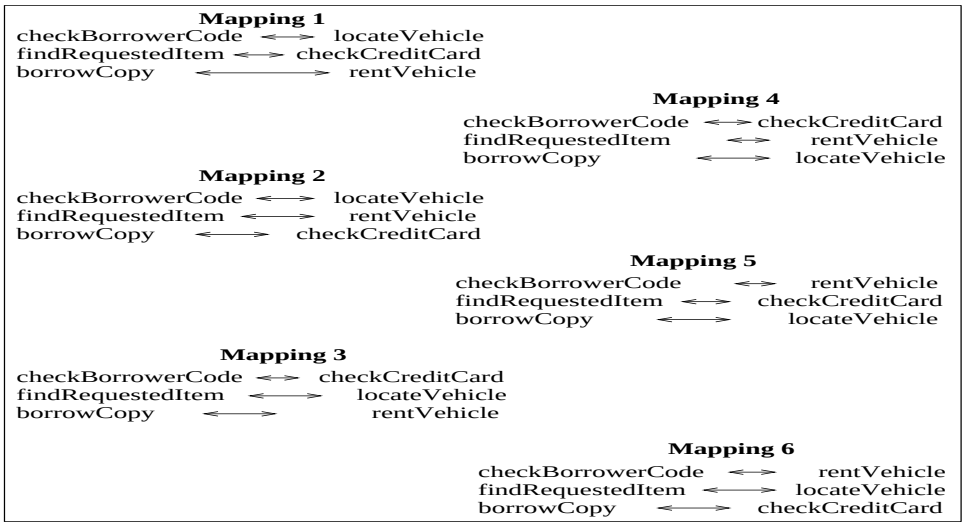


Figure 3: Mappings between the attributes of ATMVehicleRental and LibrarySelfCheckOut

unique identifier of this resource (i.e. *BookCopyId* and *TransportationNumber*); and (3) involve analogous substages of processing in a gradual mode of search(initially they determine some general category of resources and then they select a specific resource within it). Since none of their subprocesses is further decomposed, the attribution distances between them are evaluated from the distances between their inputs and outputs. The smallest such distances are obtained between the inputs and outputs of the process-pairs (*SelectBookByTitle*, *SelectVehicle*) and (*DetermineBookCategory*, *SelectVehicleCategory*), due to their relatively smaller generalization distances. Indeed all their input and output entities are classified under the class *EntityClass* and none of them has attributes of its own. Thus, they have equal pairwise classification and attribution distances, differing only with respect to their generalization distances. These are computed on their generalization graphs as shown in Figure 5.

Hence, *SelectBookByTitle* is mapped onto *SelectVehicle*. The mapping between *DetermineBookCategory* and *SelectVehicleCategory* is determined similarly.

4. Revising specifications on the basis of analogies

Analogies between reusable specifications of existing software systems and specifications of systems under development(referred to as *source* and *target* specifications respectively in the following) can be used for revising the latter in reference to the former. Three kinds of specification elements can be distinguished with respect to an analogy, including: (1) corresponding

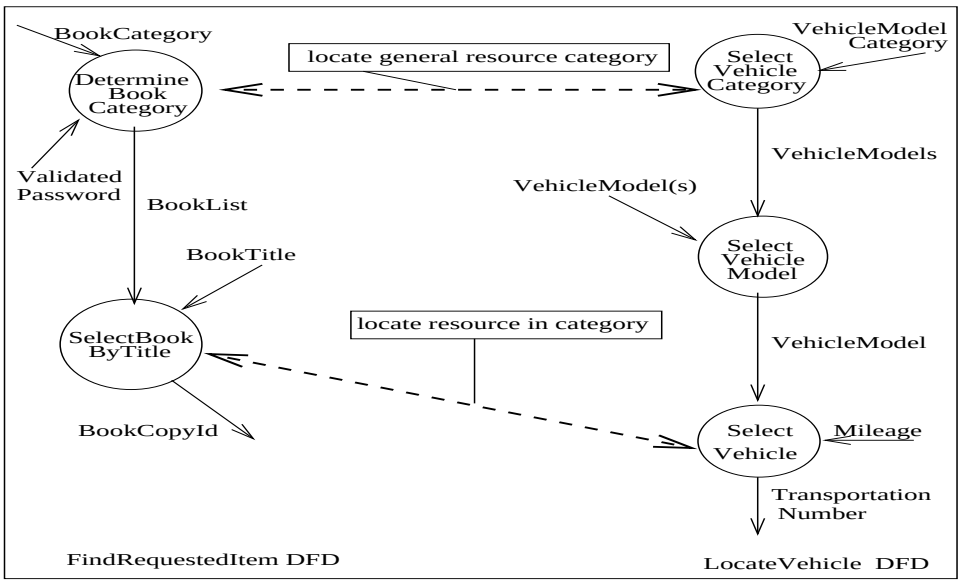
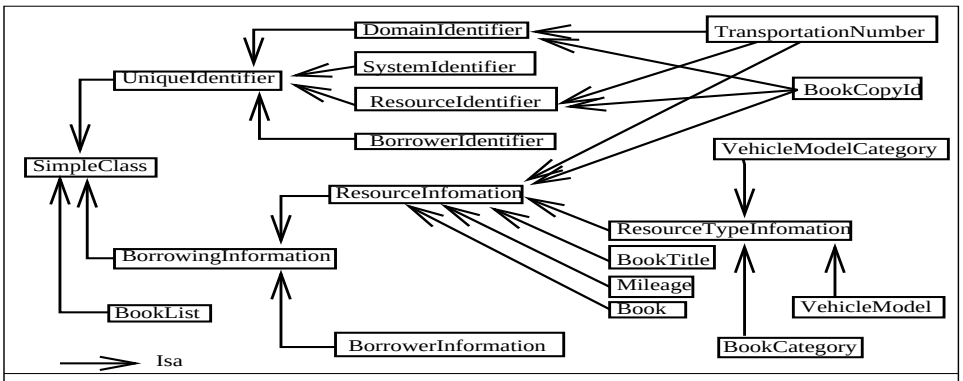


Figure 4: Analogies between the FindRequestedItem and LocateVehicle

elements; (2) unique elements of the source (i.e. elements of the source not mapped onto any of the target elements); and (3) unique elements of the target (i.e. elements of the target not mapped onto any of the source elements), as depicted in Figure 6. Corresponding elements of the target may be revised according to the modelling of their source counterparts. Unique elements of the source specification may be realized as missing from the target and consequently be imported to it, improving its completeness. Finally, unique elements of the target may be deemed redundant and consequently removed.

Let us give an example of a revision in reference to the decompositions of the *BorrowCopy* and the *RentVehicle* subprocesses in the DFDs of Figure 1. These borrowing commitment subprocesses are specified by the data flow diagrams shown in Figure 7.

The analysis of their similarity detects analogous substages concerning the update of information about resource borrowers before the commitment of borrowing(depicted by the dashed lines in Figure 7). According to *BorrowCopy*, before registering the borrowing of a book copy (cf. *CommitBorrowing* process in Figure 7), it is necessary to check the status of and retrieve information about the library borrower (cf. *CheckAndUpdateBorrowerStatus* process in Figure 7). Similarly, the reservation of a vehicle(cf. process *ReserveVehicle* in Figure 7) must follow the registration of the name and credit card number of the customer(cf. process *UpdateCus-*



Generalization Distances Dg

Pair of Entities	Distance	Pair of Entities	Distance
BookCopyId, TransportationNumber	0.4	BookCategory, Mileage	0.7
BookCategory, VehicleModelCategory	0.4	BookCopyId, VehicleModel	0.95
BookCategory, VehicleModel	0.4	BookList, Mileage	1.08
BookTitle, Mileage	0.5	BookList, VehicleModel	1.28
BookTitle, VehicleModel	0.7	BookList, TransportationNumber	1.28
BookTitle, VehicleModelCategory	0.7	BookList, VehicleModelCategory	1.28

Figure 5: Generalization graph of and distances between input and output entities

tomers in Figure 7). On the other hand, the subprocess *CheckExistenceOfRentingRequest* of *RentVehicle* has no analogous counterpart in *BorrowCopy*. This subprocess checks whether or not the customer has imposed a renting request after having found a vehicle. Such a test would be reasonable for the borrowing of book copies too, since a borrower may choose not to borrow after his/her search.

Such analogical conjectures result in proposals to the agents involved in the requirements specification process. However, they cannot be automatically adopted because there is no generally acceptable objective criterion for validating analogical inferences(Hall R. 1989, Maiden N. 1992).

Yet, similarity analysis aids analogical inference, by informing humans about (1) the analogous and unique attributes of two objects; (2) the overall distance measure indicating the aptness of their analogy; and (3) the pairwise distances between their mapped attributes and the salience of those attributes. These mappings and distances, which span the entire structural decompositions of specifications, act as catalysts to the comprehension of the source specification(especially in cases where it is a complex one), hence facilitating its reuse. In fact, the source specification is exposed to the software engineer in terms of the target specification, which directly reflects his/her mental conception of requirements(Maiden N. and Sutcliffe A. 1992, Sutcliffe A. and Maiden N. 1992). Thus, it is more easily comprehensible.

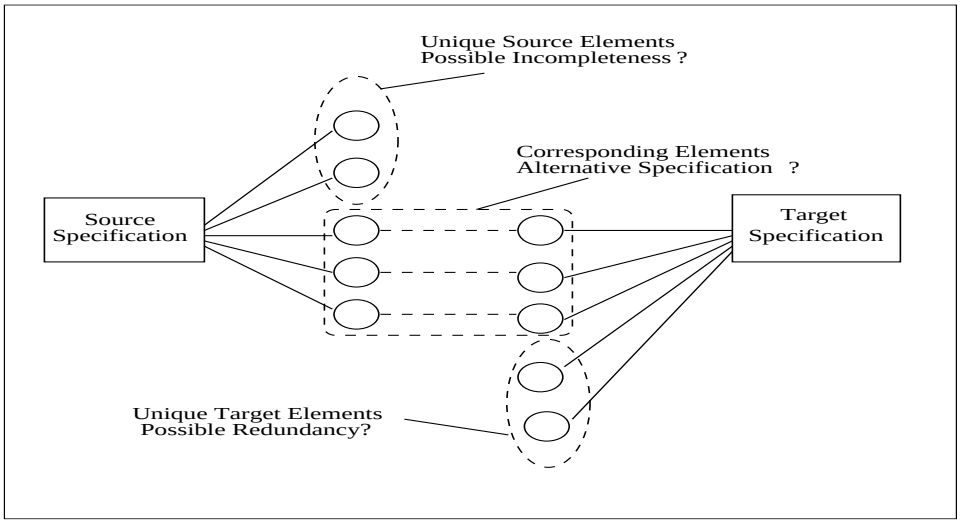


Figure 6: Pairwise analogies between source and target specifications

5. Pragmatic considerations

In this Section we discuss two aspects of similarity analysis, which are critical for applications in our view: sensitivity of similarity analysis to the completeness of specifications and computational cost.

5.1 Sensitivity to completeness of specifications

Similarity analysis is relatively tolerant to partially incomplete specifications but on the same time sensitive enough to exploit the presence of additional semantic information in producing more accurate results, faster. This behaviour has been formally analysed in(Spanoudakis G. 1994a). Here it is demonstrated through some observations in reference to the example of Section 3.

Recall the analysis of the data flow diagrams of Figure 1. The partition of their attributes into *input, output* and *consistsOf* attribute classes (cf. Figure 2) enabled a restricted search for optimal isomorphisms between them. In the absence of this partition, similarity analysis would have mapped *input, output* and *consistsOf* attributes in the same way based solely on the structural resemblances of their value-objects, but it would have been more expensive. In fact, the criterion of semantic homogeneity would not be able to limit comparisons between attributes based on their uncommon classification.

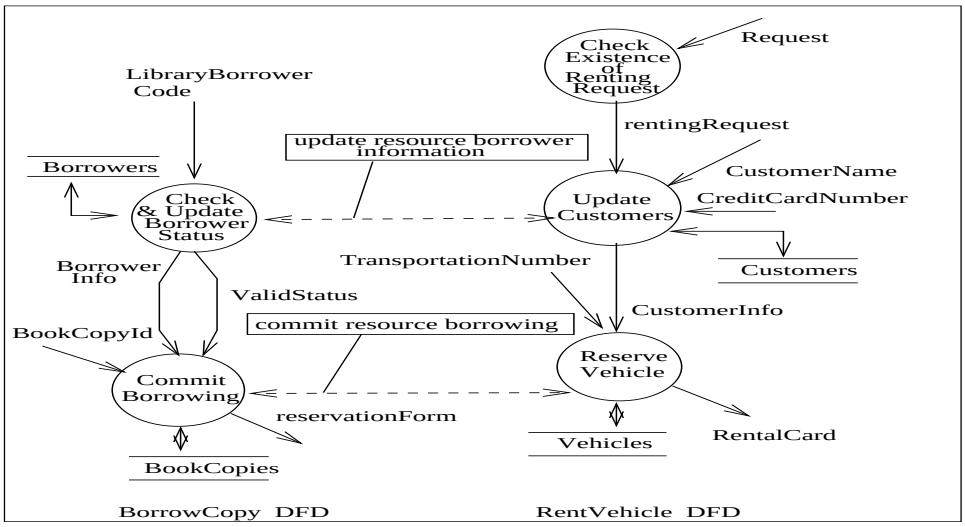


Figure 7: Analogies between resource borrowing commitment processes

Another remark concerns the effect of generalization. The detected analogies in Section 3 indicate that certain subprocesses could have been generalized into abstract superclasses prior to similarity analysis. Figure 8 presents a possible generalization of *FindRequestedItem* and *LocateVehicle* into a common superclass, *SearchResource*, which abstracts their analogous stages of processing (i.e. locating a resource category and then a resource within it).

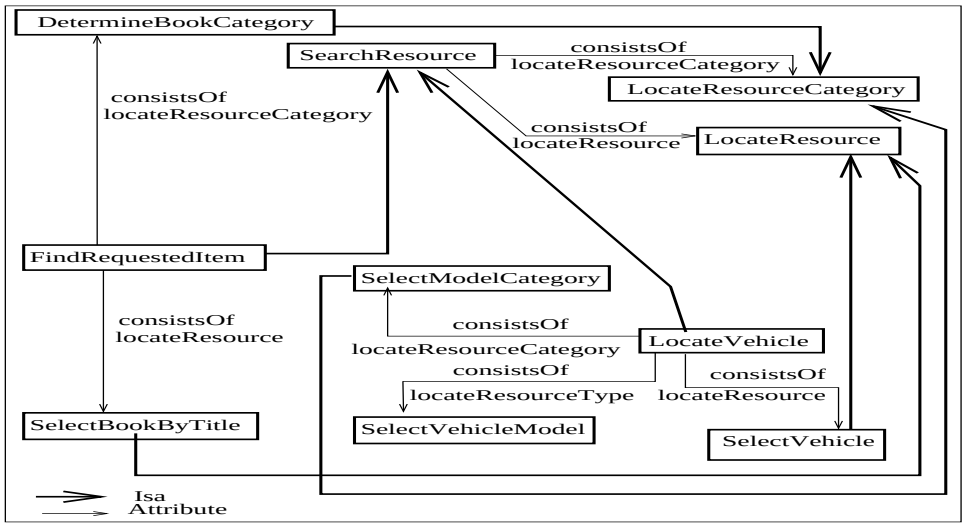


Figure 8: Generalization of the FindRequestedItem and LocateVehicle processes

If present, *SearchResource* would affect the analysis of similarity between *FindRequestedItem* and *LocateVehicle*. Instead of an evaluation of all the possible isomorphisms between their

attributes, a direct mapping between the attributes *locateResourceCategory* and *locateResource* of *FindRequestedItem* and *LocateVehicle* would take place because they would share the same original class(i.e. the attribute classes *locateResourceCategory* and *locateResource* of the process *SearchResource*, respectively), as proved in (Spanoudakis G. 1994a, cf. theorem 3.10). As a result of a smaller generalization distance between these attributes(namely, 0, according to function d_g in the Appendix), the overall distance between *FindRequestedItem* and *LocateVhicle* would also be smaller, indicating a stronger analogy between them due to the presence of their common superclass(see also Wegner P. 1987).

It can be thus seen that as the target specification evolves from partial to complete, the detection of analogies becomes more refined, accurate and efficient, because the similarity model exploits all the available information, not only to make finer evaluations, but also to limit the search space.

5.2 Complexity of analysis

As proved in (Spanoudakis G. 1994a), the worst case complexity of similarity analysis is polynomial if recursion during the computation of the overall distances between objects is terminated at a fixed level in their attribution closures. This complexity is dominated by the search for the optimal isomorphism between the attributes of objects at any of these levels, whose partial complexity is $O(n_1^3 + n_2^3)$ (n_1 , n_2 are the numbers of the attributes of the objects involved). This cost is lower than that of other computational models of analogy, which have combinatorial complexities(cf. our review in Section 7). The average complexity, on the other hand, has been shown to improve as the schema of specification descriptions evolves and gets refined (Spanoudakis G. 1994a). This is expected to occur along the lifetime of a specifications repository. In other words, the accumulation of knowledge is used positively, offsetting in part the effect of data accumulation.

6. Empirical Evaluation

The prototype implementation of the similarity model, integrated into the SIS, was used in preliminary experiments concerning: (1) the consistency of estimates produced by our model with similarity assessments provided by humans(consistency is a prerequisite for employing the

model in tasks of analogical reuse carried out in cooperation with humans); and (2) the recall performance of similarity analysis in retrieval.

6.1 Consistency with human assessment

In the first experiment, we used a conceptual model of the C++ programming language (Stroustrup B. 1987) developed to support the static analysis of C++ programs (Doerr M. and Klimathianakis P. 1993).

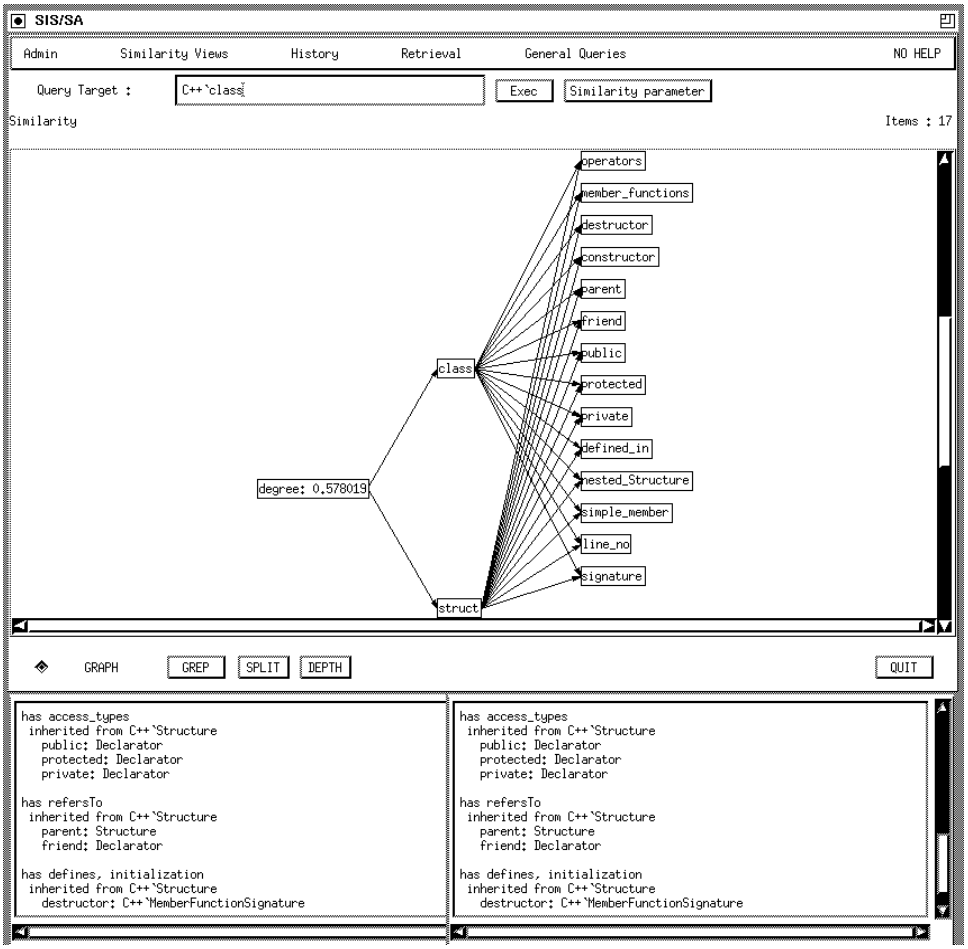


Figure 9: Similarity between C++ classes and structures

Ten cases of similarity analysis were carried out. In each case, the elements of a source set of C++ concepts were ranked in descending similarity order with respect to a target C++ concept. The similarities between the compared concepts in each of these cases(i.e. inverted overall distance measures) were computed from their descriptions in the conceptual model.

Table 1: Rank correlations between model and human generated similarity orderings

Case	Set Size	Subject 1	Subject 2	Subject 3	Subject 4	Subject 5
1	17	0.37	0.5 *	0.47 *	0.37	0.46 *
2	37	0.84 *	0.75 *	0.57 *	0.60 *	-0.06
3	18	-0.32	-	0.5 *	-0.13	0.12
4	18	-0.12	-	0.5 *	-0.18	0.10
5	4	0.55	0.55	0.55	0.55	0.55
6	17	0.55 *	0.60 *	0.55 *	0.51 *	0.64 *
7	4	0.55	0.85	0.65	0.55	0.4
8	18	0.57 *	0.48 *	0.87 *	0.65 *	0.52 *
9	17	0.42 *	0.71 *	0.39	0.50 *	0.5 *
10	19	0.52 *	0.87 *	0.73 *	0.86 *	0.64 *

Figure 9 presents the result of the similarity analysis between the conceptual descriptions of a C++ class and a C++ structure using the SIS tool.

The same cases were given to five software engineers ignorant of these results, in order to rank the source concepts with respect to their similarities to the target. The subjects had different degrees of expertise in programming with C++ and knowledge of the C++ conceptual model(decreasing along with increasing subject numbers in Table 1).

The *rank correlations* of the two orderings were measured according to the Spearman coefficient(Kevork K. 1984). As shown in Table 1, a positive rank correlation was found in all but 5 cases. Most of the positive coefficients(28 out of 44) were found statistically significant (coefficients marked by an asterisk) according to the *D or t criteria*(Kevork K. 1984) at the level $p < 0.05$, unlike the negative ones. These preliminary findings indicate that the similarity model generates measures which do not violate human intuition about similarity.

6.2 Recall evaluation

Recall essentially measures the possibility of ignoring relevant analogs in an analogical reasoning session, after analogical retrieval (high recall indicates a low such possibility and vice versa).

Using exactly the same C++ conceptual model and four of the subjects that participated in the first experiment we measured the *recall* performance of the similarity model. The subjects were given the same 10 cases as in the previous experiment and were asked to indicate whether each of the source C++ concepts could be used instead of the target one in some programming

Table 2: Average recall for different cutoffs

Cutoff	Subject 1	Subject 2	Subject 3	Subject 5	General
10 %	0.33	0.35	0.34	0.44	0.35
25 %	0.54	0.40	0.74	1.00	0.60
50 %	0.83	0.80	0.91	1.00	0.86

task they could think of. Their answers - interpreted as indicating the analogy between the relevant concepts in programming - were used as judgements of relevance(Su L. 1992). On the basis of these judgements, we measured the recall performance of the model in reference to the source sets. Recall was measured according to the formula: $r = a_c/R$ where a_c is the number of the relevant sources in the first c % positions of the sorted(in descending similarity order) source set and R is the total number of relevant items in the entire source set(Su L. 1992).

Table 2 presents measured recalls for different subjects and cutoff levels. In our view, the results are encouraging since the first two of the selected cutoffs were fairly low and the proportion of the items in any of the source sets, judged as relevant by any of the subjects, did not exceed 25%. In fact, only 10% of items in the source sets were assessed as being relevant on average.

7. Related work

Systems developed to support analogical reuse of specifications include the IRA (Maiden N. and Sutcliffe A. 1994, Maiden N. 1992) and the SPECIFIER(Miriyala K. and Harandi M. 1991).

The IRA(Intelligent Requirements Advisor) detects analogies by matching specifications against a predefined set of abstractions, which model fundamental behaviours, structures, goals and constraints of classes of requirements engineering problems. Matching requires specifications to be described in the domain modelling language of the system(i.e. a specification model in our terminology) and is bounded by the employed problem abstractions, whose completeness and granularity constitute open research issues(Maiden N. 1992).

SPECIFIER(Miriyala K. and Harandi M. 1991) detects analogies by matching structural descriptions of user specifications against pre-existing concepts on the basis of fixed primitive relations determining pairs of analogous elements in them. Successful matches are utilized for retrieving and instantiating operational schemas, indexed by these concepts in order to complete and formalize user specifications.

Other systems or methods supporting software reuse by incorporating forms of analogical reasoning, such as case-based reasoning, include: (i) KAPTUR/LEARN(Bailin S. and Moore M. 1991) and Techne(Mylopoulos J. and Rose T. 1991) for case-based reuse of software designs; and (ii) the genericity approach(Katalagarianos P. and Vassiliou Y. 1995) for case-based reuse of code. Notice that the analogical reuse of downstream software artifacts does not have all the practical problems of the analogical reuse of specifications. For instance, downstream artifacts can be described using fixed and well-articulated abstractions, such as abstract data types and algorithms, as opposed to requirements specifications(Krueger C. 1992).

Finally, general computational models for analogy are usually inadequate for specifications reuse. This is because in order to detect them, they need to be informed about features of objects, which are important for analogies sought for a particular purpose (e.g. derivational analogy (Carbonell J. 1986), NLANG(Greiner R. 1988)) or because they have high computational complexity(e.g. combinatorial complexity of SME(Falkenhainer B. et al. 1990)).

In contrast with the reviewed approaches, the similarity model discussed in this paper: (1) is applicable on specifications built according to different specification models, (2) detects analogies without requiring extra information pertaining solely to this task, and (3) scales well to complex problems due to its polynomial complexity.

8. Conclusions and further research

In this paper, we presented a general model for computing similarities between requirements specifications so as to promote their analogical reuse.

The model assumes that specifications are described using semantic modelling abstractions, including classification, generalization and attribution. The semantics of these abstractions enable the employment of general criteria for detecting analogies between specifications, without relying on other special knowledge. Different specification models are simultaneously supported. The similarity model is relatively tolerant to incompleteness of specifications, improves as their semantic content is enriched and copes well with large scale problems.

The model supports specification by reuse by suggesting revisions in specifications on the basis of detected analogies. The whole activity, demonstrated through an example in Section 4, is

currently being elaborated into a process model, predicting actions of revising or completing requirements specifications on the basis of specific kinds of detected analogies.

Appendix: The Computational Model for Similarity Analysis

The similarity model is composed of *distance* and *salience* measuring functions defined in reference to Telos objects. In this appendix, we formally introduce these objects and functions.

a.1 The Structure of Telos Objects

Telos objects are partitioned according to their *classification level* into Tokens and Classes. Classes are further partitioned into Simple Classes, Meta Classes, Meta Meta Classes and so on. They are also partitioned according to their *role* into Individuals (i.e. objects modeling entities) and Attributes (i.e. objects modeling properties and/or relations between entities). These four basic categories of objects have the following tuple forms:

$$Individual\ Tokens\ (I_t)\ :\ o_i = [In, A]$$

$$Individual\ Classes\ (I_c)\ :\ o_i = [In, Isa, A]$$

$$Attribute\ Tokens\ (A_t)\ :\ o_i = [From, In, A, To]$$

$$Attribute\ Classes\ (A_c)\ :\ o_i = [From, In, Isa, A, To]$$

In these forms, i is an object identifier for o_i, *In* is a set of object identifiers denoting the classes of o_i (o_i is said to be an *instance* of the classes in *In*), *Isa* is a set of object identifiers denoting the superclasses of o_i , A is a set of system identifiers denoting the direct attributes (i.e. those not inherited) of o_i, *From* is the identifier of the object owning the attribute o_i and *To* is the identifier of the object being the value/range of attribute o_i.

Telos objects have *logical names*(unique to individual objects but shared by more than one attribute objects owned by distinct classes). Telos classes have *intensions (INT[i])* including the identifiers of the attributes they introduce or inherit from their superclasses. Each Telos attribute class i has an *original class OC(i)*(i.e. the most general attribute superclass of i, which has an identical logical name with it) and:

(i) a *scope S[i]* (i.e. a set with the classes which it applies to) defined as:

$$S[i] = \{c | \exists x, j\ :\ (x = o_i.From)\ and\ (c \in o_x.Isa)\ and\ (j \in INT[c])\ and\ (l(i) = l(j))\ or\ (o_i.From \in o_c.Isa))\}$$

(ii) a set of *refining classes R[i]* (i.e. a set with the classes which inherit and refine i) defined as:

$$R[i] = \{c|(c \in S[i])\ and\ (\exists x : (x \in o_c.A)\ and\ (l(i) = l(x)))\}$$

(iii) a set of *ranges AR[i]* (i.e. a set with the classes serving as its ranges) defined as:

$$AR[i] = \{x|(\exists y, z : (y \in R[i])\ and\ (z \in o_y.A)\ and\ (l(z) = l(i))\ and\ (o_z.TO = x))\}$$

In these definitions l is assumed to be an M:1 mapping from the set of object identifiers to the set of object logical names.

a2. Distance Functions

(i) The Identification Distance

The identification distance indicates whether two objects are identical or not. Object identity depends on the equality of internal unique identifiers assigned to objects by the database system. Formally, the identification distance is defined as follows:

Definition 1: The identification distance, d_{id}, between two objects o_i and o_j is defined as:

$$d_{id}(o_i, o_j) = \begin{cases} 0 & \text{if } i = j \\ \\ 1 & \text{if } i \neq j \end{cases}$$

(ii) The Classification Distance

The classification distance between two objects is measured by identifying and measuring the importance of their non-common classes. This importance is measured by the specialization depth, *SD(x)*, of each class, formally defined as:

Definition 2: $SD(x)$ is the maximum length (number of links) of the paths connecting class x with the most general class of its generalization taxonomy, called *specialization depth* of x.

Given SD(x), the classification distance is defined as follows:

Definition 3: The classification distance, d_c, between two objects o_i and o_j is defined as:

$$d_c(o_i, o_j) = \frac{b_c D_c(o_i, o_j)}{b_c D_c(o_i, o_j) + 1} \quad (\ b_c \in R^+\ )$$

$$D_c(o_i, o_j) = \sum_{x \in NCC_{ij}} SD(x)^{-1}\ , \quad NCC_{ij} = (o_i.In - o_j.In) \bigcup (o_j.In - o_i.In)$$

Thus, in measuring the classification distance between two objects, their non-common classes, which are placed at higher levels in generalization taxonomies are considered as more important than those placed at lower levels. b_c is a normalization parameter evaluated so that d_c equals 0.5 when D_c takes its average value given a specific set of objects(i.e. a context-sensitive estimation of the classification distance).

(iii) Generalization Distance

Definition 4: The generalization distance, d_g, between objects o_i and o_j is defined as:

$$d_g(o_i,o_j) = \begin{cases} \frac{b_g D_g(o_i,o_j)}{b_g D_g(o_i,o_j)+1} \ (b_g \in R^+) & \text{if } o_i, o_j \in I_c \\ d_o(o_i,o_j) & \text{if } o_i, o_j \in A_c \end{cases}$$

$$D_g(o_i,o_j) = \sum_{x \in NCS_{ij}} SD(x)^{-1}, \quad NCS_{ij} = (o_i.Isa - o_j.Isa) \bigcup (o_j.Isa - o_i.Isa) \bigcup \{i,j\}$$

$$d_o(o_i,o_j) = \begin{cases} 0 & \text{if } OC(i) = OC(j) \\ 1 & \text{if } OC(i) \neq OC(j) \end{cases}$$

The generalization distance between individual classes is measured like their classification distance, except that their superclasses are taken into account. The generalization distance between attribute classes depends on the identity of their original classes and distinguishes between refined specializations of the same attribute and specializations between attributes with shared but non-identical semantics(Spanoudakis G. 1994a). b_g is similar to and evaluated like b_c in definition 3.

(iii) Attribution Distance

Definition 5: The attribution distance, d_a, between objects o_i and o_j is defined as:

$$d_a(o_i,o_j) = \frac{b_a D_a(o_i,o_j)}{b_a D_a(o_i,o_j)+1}, \quad (b_a \in R^+)$$

$$D_a(o_i,o_j) = \begin{cases} \infty & \text{if } o_i.A = \emptyset \text{ or } o_j.A = \emptyset \\ min_{m \in I(o_i,o_j)} (\sum_{(x_1,x_2) \in m} s(x_1)s(x_2)d(x_1,x_2) & \\ + \sum_{x_3 \in o_i[m]} s(x_3)^2 + \sum_{x_4 \in o_j[m]} s(x_4)^2) & \text{otherwise} \end{cases}$$

$$s(i) = max_{x \in OCL[i]}\{SL(x)\}$$

where

$I(o_i, o_j)$: is the set of all possible isomorphisms between the semantically homogeneous attributes of o_i and o_j (two attribute objects k and l are semantically homogeneous if and only if $OCL[k] \equiv OCL[l]$ where $OCL[x] = \{y|(y = OC(z))\ and\ (z \in o_x.In)\}$)

$o_i[m](o_j[m])$: is the set with the attributes of $o_i(o_j)$ that map onto no attribute of $o_j(o_i)$ given the isomorphism m

$SL(x)$: is the salience of attribute class x computed as described in Section a.3

The attribution distance is estimated by searching for a minimum distance isomorphism between the attributes of two objects. Mappings are only considered between attributes, which are instances of the same original attribute classes. The existence of more than one minimum distance isomorphisms does not lead to any ambiguity regarding the attribution distance measure. The model generates all of them, as alternative interpretations of the analogy between the objects under comparison, if so. The attribution distance is recursively defined through the overall distance between the values of attributes (cf. definition 6 below). Therefore, it generates optimal isomorphisms between attributes at all the successive levels of the decomposition-closures of the compared objects. b_a parameter is similar to and evaluated as b_c in definition 3.

(iv) Overall Object Distance

Definition 6: The overall distance, d, between objects o_i and o_j is defined as:

$$d(o_i, o_j) = \frac{bD(o_i, o_j)}{bD(o_i, o_j) + 1} \quad (\ b\ \in\ R^+\ )$$

$$D(o_i, o_j) = \begin{cases} (d_{id}(o_i, o_j)^2\ +\ d_c(o_i, o_j)^2\ + d_g(o_i, o_j)^2\ +\ d_a(o_i, o_j)^2\ + \\ d_c(o_i, o_j)d_g(o_i, o_j)\ + d_c(o_i, o_j)d_a(o_i, o_j)\ + \\ d_g(o_i, o_j)d_a(o_i, o_j))^{1/2} & \text{if } o_i, o_j \in (I_t \bigcup I_c) \\ \\ (d_{id}(o_i, o_j)^2\ +\ d_c(o_i, o_j)^2\ + 36d_g(o_i, o_j)^2\ +\ d_a(o_i, o_j)^2\ + \\ d(o_i.To, o_j.To)^2\ +\ 12d_c(o_i, o_j)d_g(o_i, o_j)\ + \\ d_c(o_i, o_j)d_a(o_i, o_j)\ + \\ 12d_g(o_i, o_j)d_a(o_i, o_j))^{1/2} & \text{if } o_i, o_j \in (A_t \bigcup A_c) \end{cases}$$

D aggregates the identification, classification, generalization and attribution distances between objects. It is defined as a quadric functional form due to experiments indicating statistically significant correlations between d_c, d_g and d_a (Spanoudakis G. 1994a). The relatively higher coefficients of products having d_g as a factor(i.e. 36, 12) ensure that attributes with the same original class (and therefore 0 generalization distance according to definition 4) will be necessarily mapped to each other, when comparing their owning objects, as proved in (Spanoudakis G. 1994a). b parameter is similar to and evaluated as b_c in definition 3.

a.3 The Salience Measuring Functions

Salience reflects the varying importance of different attributes to their owning objects and is measured as belief about their *dominance*. An attribute i is defined to be dominant (DOM_i) if it is characteristic (CH_i) and abstract (ABS_i) or determinative (DET_i) (i.e. $DOM_i \equiv (ABS_i \; and \; CH_i) \; or \; DET_i$).

Characticity of Attributes. Characticity is a property of attributes which discriminate the classes in their scopes by being refined(i.e. the specialization of their ranges) by those classes. Formally:

Definition 7: The evidence to the characticity of an attribute i, m_i^{ch} is defined as $m_i^{ch} = c_i = \frac{|AR[i]|}{|S[i]|}$

According to m_i^{ch}, the attribute *method* in Figure 10 (it presents a conceptual model of object types in different object-oriented programming languages (Weber M. 1992)) is very characteristic ($c_{method} = 0.8$) as opposed to the attribute *author* ($c_{author} = 0.2$). This difference is due to the refinement of *method* in most of the classes of its scope(i.e. classes *CoolObjectType*, *C++Class* and *EiffelClass*) unlike *author*.

Abstractness of Attributes. Abstractness is a property of attributes, which are significant to the structure and behaviour of their owning objects. It depends on the classes introducing them in conceptual models(Spanoudakis G. and Constantopoulos P. 1994b), which in turn may be distinguished into *concrete* and *abstract* according to whether or not they have instances of their own (Meyer B. 1989, Wegner P. 1987). Only abstract classes introduce attributes, which are essential for the structure and functional behaviour of the instances of their concrete

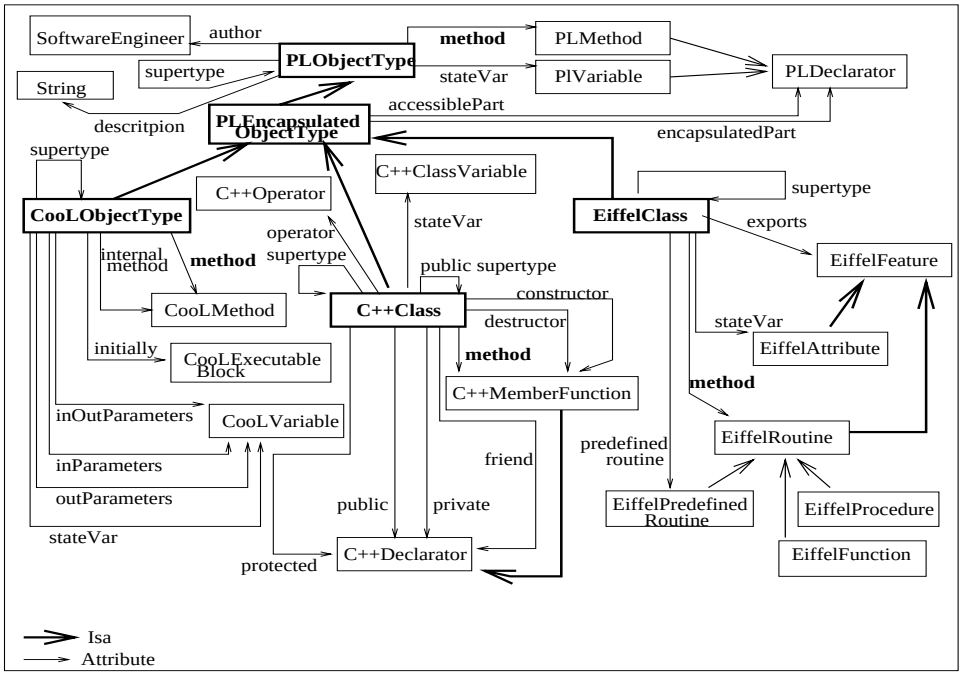


Figure 10: A Conceptual Model of Object Types in Object Oriented Languages

subclasses (Rosch E. et al. 1976). Formally:

Definition 8 : The evidence to the abstractness of an attribute i introduced by class j, m_i^a is defined as

$$m_i^a \;=\; e_j \;=\; \frac{|EXT_s[j]|}{|EXT[j]|} \quad j = o_{OC(i)}.From$$

where

$EXT[j]$ is a set with the identifiers of all the instances of j, called *extension* of j

$EXT_s[j]$ is a set with the identifiers of all the instances of j, which are also instances of another class(not a superclass of it) called *shared extension* of j

According to m_i^a the evidence to the abstractness of the attributes *method, stateVar, supertype* and *author* introduced by *PLObjectType* in Figure 10 is 1(*PLObjectType* has no instances of its own since an object type must be implemented in some specific programming language). Unlike them, the abstractness of the attribute *initially* of class *CooLObjectType* is 0, assuming that this class has no shared instances(COOL object types cannot be types of some other programming language).

Determinance of Attributes. Determinance is a property of attributes, whose values determine the values of other attributes having the same domain with them. This might be expressed as a mapping M between the value-ranges of the relevant attributes. If M is a total and onto isomorphism, the respective attributes will be said to be totally equivalent. An attribute x will be said to be determinant if it determines at least one of the other attributes in its scope.

The violation of the domain equality condition for a dependency mapping M is evidenced by the existence of non common classes in the scopes of two attributes since such classes indicate disjoint subsets in their domains (Spanoudakis G. and Constantopoulos P. 1994b). Formally:

Definition 9: The evidence to the undefinability of a dependency mapping between an attribute i and an attribute j, due to their non common scope, m_{ij}^{ncs} , is defined as

$$m_{ij}^{ncs} = d_{ij} = \frac{|(S[i] - S[j])| + |(S[j] - S[i])|}{|(S[i] \bigcup S[j])|}$$

According to definition 9, the more the non common classes in the scopes of two attributes, the less likely the equality of their domains and consequently, the less likely the definability of a dependency mapping between them.

The definability of a total equivalence mapping between two attributes is unlikely in cases where they are refined by different classes in their scopes (Spanoudakis G. and Constantopoulos P. 1994b). Formally:

Definition 10: The evidence to the undefinability of a total equivalence mapping between an attribute i and an attribute j, due to their non common refinements over a conceptual schema, m_{ij}^{ncr} , is defined as

$$m_{ij}^{ncr} = r_{ij} = \frac{|(S[i] - AR[i]) \bigcap AR[j]| + |(S[j] - AR[j]) \bigcap AR[i]|}{|AR[i] \bigcup AR[j]|}$$

m_{ij}^{ncr} measures the cases where an attribute i is refined over a conceptual schema but another attribute j is not and vice versa. The absence of non-common refinements is not considered as evidence for the existence of a total equivalence mapping between i and j.

Estimating the Belief and Plausibility of Dominance. The evidence for the dominance of an attribute is measured from the evidence assigned to the properties of characticity, abstract-

ness and determinance. As proved in (Spanoudakis G. 1994a), the evidence functions m_i^{ch}, m_i^a, m_{ij}^{ncs} and m_{ij}^{ncr} satisfy the axioms of the *basic probability assignments* in the *Dempster-Shafer theory of evidence*(Shafer G. 1975). Therefore, they can be interpreted as such assignments and consequently be combined using the *rule of the orthogonal sum*(Shafer G. 1975) resulting into the following belief and plausibility functions to attribute dominance:

$$Bel(DOM_i) = e_u c_i$$

$$P*(DOM_i) = (1-(1-c_i e_u)\prod_{j=1}^{n} d_{ij}) \ (if\ any\ kind\ of\ interattribute\ dependency\ is\ taken\ into\ account)$$

$$P*(DOM_i) = 1-(1-c_i e_u)(\prod_{j=1}^{n}(1-(1-d_{ij})(1-r_{ij}))) \ (if\ only\ total\ equivalences\ are\ taken\ into\ account)$$

In these formulae, u is the introducing class of attribute i(i.e. $u = o_{OC(i)}.From$) and n is the number of the other attributes of the classes of its scope.

Then the salience of an attribute class i is defined as follows:

Definition 11: The salience of an attribute class o_i, SL(i), is defined as:

$$SL(i) = \frac{Bel(DOM_i) + P*(DOM_i)}{2} \ \ \forall\ i \in A_c$$

According to definition 11, the salience of the attribute classes *method*, *supertype* and *stateVar* of class *PLObjectType* in Figure 10 is 0.9, while that of *author* and *description* is 0.502(when total equivalencies of attributes are taken into account).

References

Bailin S., Moore M. 1991. A Knowledge-Oriented Approach to Reuse, In *Proc. of the First International Workshop on Software Reusability*, Dortmund, Germany, July

Bellinzona R. et al. 1995. Reusing Specifications in OO Applications, *IEEE Software*, March, pp. 65-75

Carbonell J. 1986. Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition, *Machine Learning: An Artificial Intelligence Approach*, Vol. II, Morgan Kaufmann Publishers Inc., Los Altos, California

Constantopoulos P. et al. 1993. Repositories for Software Reuse: The Software Information Base, In *Proc. of the IFIP Conference on the Software Development Process*, Como,Italy, pp. 285-307

Constantopoulos P. , Doerr M. 1993. The Semantic Index System: A Brief Presentation, Institute of Computer Science, Foundation for Research and Technology-Hellas, Heraklion, Crete, Greece

Doerr M., Klimathianakis P. 1993. A TELOS Model for C++ Static Analysis Data, Technical Report, Institute of Computer Science, Foundation for Research and Technology-Hellas, Heraklion, Crete, Greece, September

Falkenhainer B. et al. 1990. The Structure Mapping Engine: Algorithm and Examples, *Artificial Intelligence*, 41, pp. 1-64

Finkelstein A. et al. 1992. Viewpoints: A Framework for Integrating Multiple Perspectives in Systems Development, *International Journal of Software Engineering and Knowledge Engineering*, 2(1), pp. 31-58

Fuggetta A. 1993. A Classification of CASE Technology, *IEEE Computer*, December, pp. 25-38

Greiner R. 1988. Learning and Understanding by Analogies, *Artificial Intelligence*, 35, pp. 81-126

Hall R., 1989. Computational Approaches to Analogical Reasoning: A Comparative Analysis, *Artificial Intelligence*, 39, pp. 39-120

Hull R., King R. 1987. Semantic Database Modelling: Survey, Applications and Research Issues, *ACM Computing Surveys*, 19(3), pp. 201-260

Hofmann H. 1993. Requirements Engineering: A Survey of Methods and Tools, Institute for Informatics, University of Zurich, March

Johnson L. et al. 1992. Representation and Presentation of Requirements Knowledge, *IEEE Transactions on Software Engineering*, 18(10), pp. 853-869

Krueger C. 1992. Software Reuse, *ACM Computing Surveys*, 24(2), pp. 131-184

Katalagarianos P., Vassiliou Y. 1995. On the Reuse of Software: A Case-Based Approach Employing a Repository, *Automated Software Engineering*, 2(1), pp. 55-86

Kevork K. 1984. Analysis of Regression and Correlation, Statistical Methods, Vol. III, Athens University of Economics, Athens, 1984

Maiden N. 1992. Analogical Specification Reuse During Requirements Analysis, Ph.D Thesis, Dept. of Business Computing, City University, London , July

Meyer B. 1989. Object Oriented Software Construction, C.A.R Hoare, Series Editor

Mylopoulos J., Rose T. 1991. Case-Based Reuse for Information System Development: The Techne Project, In *Proc. of the First International Workshop on Software Reusability*, Dortmund, Germany, July

Maiden N., Sutcliffe A. 1992. Exploiting Reusable Specifications through Analogy, *Communications of the ACM*, 35(4), pp. 55-64

Maiden N., Sutcliffe A. 1994. Requirements Critiquing Using Domain Abstractions, In *Proc. of the IEEE Conference on Requirements Engineering*, pp. 184-193

Miriyala K., Harandi M. 1991. Automatic Derivation of Formal Software Specifications From Informal Descriptions, *IEEE Transactions on Software Engineering*, 17(10), pp. 1126-1141

Mylopoulos J. et. al. 1990. Telos: Representing Knowledge About Information Systems, *ACM Transactions on Information Systems*, 8(4), pp. 325-362

Peckham J., Maryanski F. 1988. Semantic Data Models, *ACM Computing Surveys*, 20(3), pp. 153-190

Rosch E. et al. 1976. Basic Objects in Natural Categories, Academic Press

Reubenstein H., Waters R. 1991. The Requirements Apprentice: Automated Assistant for Requirements Acquisition, *IEEE Transactions on Software Engineering*, 17(3), pp. 226-240

Shafer G. 1975. A Mathematical Theory of Evidence, Princeton University Press

Spanoudakis G. and Constantopoulos P. 1994a. Similarity for Analogical Software Reuse: A Computational Model, In *Proc. of the 11th European Conference on Artificial Intelligence*

(ECAI '94), Amsterdam, The Netherlands, August, pp. 18-22

Spanoudakis G. and Constantopoulos P. 1994b. On Evidential Feature Salience, In *Proc. of the 5th International Conference on Database & Expert Systems Applications (DEXA '94)*, Athens, Greece, September

Sutcliffe A., Maiden N. 1992. Analyzing the Novice Analyst: Cognitive Models in Software Engineering, *International Journal on Man-Machine Studies*, 36, pp. 719-740

Spanoudakis G. 1994a. Analogical Similarity of Objects: A Conceptual modelling Approach, Ph.D Dissertation, Department of Computer Science, University of Crete, Heraklion, September

Spanoudakis G. 1994b. Similarity Analyzer: An Implementation Overview, Working Paper #11, Institute of Computer Science, Foundation for Research and Technology - Hellas, Heraklion, Crete, Greece, September

Stroustrup B. 1987. The C++ Programming Language, Addison-Wesley, Reading, Mass.

Su L. 1992. Evaluation Measures for Interactive Information Retrieval, *Information Processing and Management*, 28(4), pp. 503-516

Tracz W. 1990. Where Does Reuse Start ?, *ACM SIGSOFT Software Engineering Notes*, 15(2), pp. 42-46

Vezerides C. 1992. The Organization of a Software Information Base for Software Reuse by a Community of Programmers, MSc Thesis, Department of Computer Science, University of Crete, Heraklion, Crete, Greece, May

Wegner P. 1987. Dimensions of Object Based Language Design, In *Proc. of the ACM Conference on Object-Oriented Programming, Systems, Languages and Applications, (OOPSLA '87)*, pp. 168-182

Weber M. 1992. Implementation Models for the SIB, ITHACA.SNI.92.E2.#2, Siemens-Nixdorf