Integrating Specifications: A Similarity Reasoning Approach

GEORGE SPANOUDAKIS

Department of Computer Science, City University

e-mail: gespan@cs.city.ac.uk

PANOS CONSTANTOPOULOS

Institute of Computer Science,

Foundation for Research and Technology-Hellas

e-mail: panos@ics.forth.gr

Abstract

Requirements analysis usually results in a set of different specifications for the same system, which must be integrated. Integration involves the detection and elimination of discrepancies between them. Discrepancies may be due to differences in representation models, modeling perspectives or practices. As instances of the semantic heterogeneity problem (D. Gangopadhyay and T. Barsalou 1991), discrepancies are broader than logical inconsistencies, and therefore not always detectable using theorem proving. This paper proposes an approach to their detection using meta-modeling and similarity analysis. Specification components are classified under a meta-model of domain independent semantic modeling abstractions and thereby compared according to a newly developed model of similarity. Similarity analysis results in an isomorphic mapping between them, which can be used as a basis for reconciling and merging them. The approach is extensible in the sense that it accommodates different models for representing specifications, and analysis scales up to manage large, complex specifications because the complexity of similarity analysis is polynomial.

1. Introduction

The various agents involved in requirements analysis of large information systems produce and hold different specifications, reflecting their own viewpoints on requirements about them (a process usually referred to as *distributed requirements engineering* (B. Nuseibeh et al. 1993, N. Maiden et al. 1994)). Although inevitable and acceptable in the early stages of information systems development, where innovative thinking, deferment of commitments and exploration of alternatives are all highly desirable (A. Finkelstein et al. 1994), different specifications must be integrated for guiding subsequent implementation, effectively.

Variability between specifications may occur due to a number of sources, or *dimensions*, including:

(1) representation (i.e. the use of different models for representing requirements)

(2) perspective (i.e. requirements modeling from complementary perspectives, such as usage, system, development (J. Mylopoulos et al. 1990))

(3) application domain (i.e. requirements for system components, which fall into related, yet non-identical domains)

(4) agent (i.e. requirements specified by different agents, using different modeling practices, emphasizing different aspects or expressing them in different degrees of detail)

Due to these dimensions there may appear discrepancies between specifications, in the form of semantic incongruences due to: different representation models; contradictions of terms arising in the context of the same model; components in one specification without counterparts in another; and conflicts in the values of corresponding goal attributes.

In our view, the concept of discrepancy in the context of distributed requirements engineering is broader than the concept of inconsistency in classical logic (i.e. the simultaneous truth of a fact, X, and its negation, $\neg X$). The inconsistency of a given set of formulas can be detected using resolution, if such formulas are composed from a fixed set of predicates, the identity of which enables their unification. On the other hand the detection of discrepancies can be considered as an instance of the *semantic heterogeneity problem*. This is how to decide whether given, incomplete and heterogeneous sets of representation symbols refer to the same underlying reality (D. Gangopadhyay and T. Barsalou 1991).

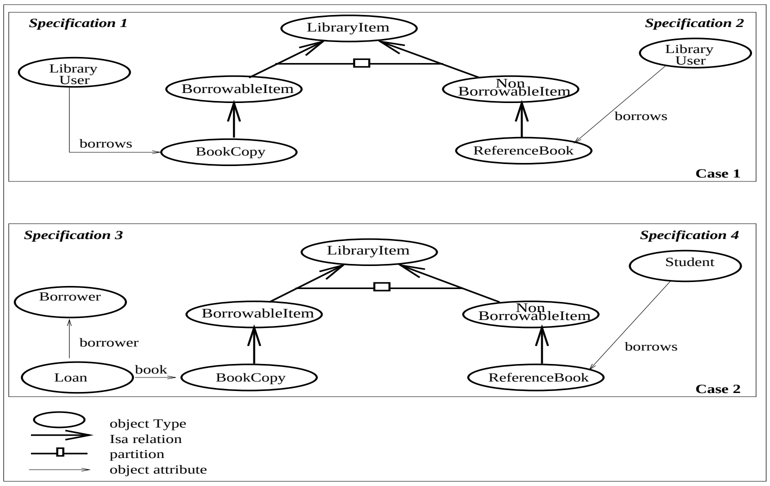

Figure 1: Inconsistencies vs. Discrepancies in Requirements Specifications

Figure 1 demonstrates the difference between inconsistencies and discrepancies, regarding two cases of different object-oriented specifications of a *borrowing* relation in a library system. In case 1, the inconsistency regards the library items that can be borrowed according to the different specifications (i.e. copies of ordinary books according to specification 1 and reference books according to specification 2, which are paiwise disjoint kinds of books according to both them). Subject to the rewriting of these specifications as a set of logical formulas, the inconsistency is detectable using theorem proving. In the second case, specifications 3 and 4 introduce the same inconsistency. However, it is not detectable since they use different names for the object types representing the library borrowers (i.e. borrower vs. student) and furthermore they represent the *borrowing* relation in different ways (i.e. an object attribute in specification 4 and an object type in specification 3). Hence, their transformation into a common set of logical formulas is impossible unless additional information about correspondences between their elements is provided.

We are especially concerned with the development of reasoning mechanisms that could assist humans in determining such discrepancies (a process referred to as *specification analysis* in the following), and with exploiting them to detect inconsistencies, as well as reconcile and merge specifications. As we argue in Section 6, currently available reasoning mechanisms are weak in supporting humans in such tasks.

Our approach to the detection of discrepancies relies on meta-modeling and similarity analysis of specifications. Specification components will be classified under a meta-model, whose elements denote general, domain-independent semantic modeling constructs and properties as well as abstract elements, allowing the structuring of specifications originally expressed in different representation models. Given their classification under this meta-model, specifications are compared using a computational measure of similarity, developed to detect analogies between conceptual objects (G. Spanoudakis 1994a). Comparisons focus on detecting isomorphisms between analogous specification components. The very properties of such isomorphisms make it likely that mapped components, even if they have different names or other modeling disparities, refer to the same underlying entity. Thus, they could be used as a basis for reconciling and merging specifications.

The rest of the paper is structured as follows. In Section 2, we introduce the meta-model for specification analysis and discuss the description of specifications according to it. In Section 3, we overview the model of similarity and subsequently, in Section 4, demonstrate how it compares specifications using an example. In Section 5, we discuss possible ways of using detected similarities and discrepancies for reconciling specifications. In Section 6, we review related work in the literature and, in Section 7 we summarize our approach and discuss open research issues. Finally, in an appendix, we introduce the representation framework underlying the similarity analysis model, namely the Telos language(J. Mylopoulos et al. 1990, M. Koubarakis et al. 1989), define the functions constituting the model and present examples of using them in measuring conceptual distances between objects.

2. The Meta-Model for Specification Analysis

The meta-model for specification analysis introduces an organized set of properties that may characterize components in different specifications discussed in the semantic modeling literature (V. Storey 1993, P. Motschnig-Pitrik 1993). In essence, this meta-model, which is domain and representation independent, enables the enrichment of the semantic content of components and allows their representation according to a common set of structuring constructs both prerequisites for the computational detection of their similarity. Before discussing it in detail, we overview Telos, a knowledge representation language (J. Mylopoulos et al. 1990) selected for defining the meta-model and describing specifications.

2.1 Telos: A Language for Representing Specifications

Telos adopts an object-oriented approach to knowledge representation. It treats entities and attributes uniformly as objects having the same rights. Objects may belong to one or more classes defining the different kinds of attributes used for their description. Like individual objects, classes can be classified under metaclasses, metaclasses under metametaclasses and so on. Furthermore, classes of the same classification level (i.e. classes, metaclasses, metametaclasses etc.) can be related through multiple generalization relations (i.e. Isa relation with a set-inclusion semantics), which enforce the strict inheritance of attributes from superclasses to subclasses.

The treatment of attributes as objects, which can be grouped into classes and have their own attributes, allows the definition of different kinds of relations, without the need of supplying specific representation primitives for each of them. Also, the ability to introduce multiple and meta classification relations enables the definition of different meta-models for describing objects, such as the meta-model we introduce in this paper. These features make Telos an expressive language, that can accommodate different representation models for specifications, in a single framework and justifies its choice for the task of specification integration.

A more detailed and formal account of the language is given in the appendix of the paper.

2.2 The Concepts of the Meta-Model

The meta-model for specification analysis is organized as a generalization taxonomy of classes. The root of this taxonomy is the class *ConceptModelingComponent*, which groups all the components in specifications regardless of the real world concept that is expressed by them. This class abstracts two basic aspects of modeling, namely the attachment of attributes and the identification of components using the attributes, *aggregates* and *identifiedBy*, as shown in Figure 2.

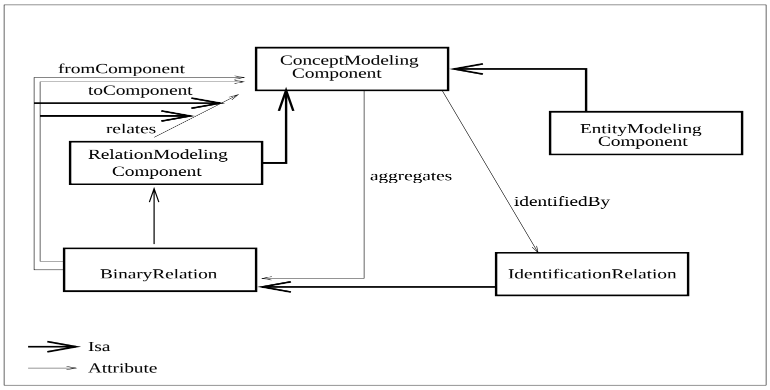

Figure 2: Components Expressing Entities and Relations

As specification components may express either entities or relations in the real world, the class ConceptModelingComponent is partitioned by its subclasses *EntityModelingComponent* and *RelationModelingComponent*.

General Classes of Components Modeling Entities. The EntityModelingComponent class is further specialized as shown in Figure 3.

The class *NaturalKindComponent* groups components representing physical entities, as opposed to tailor made or invented entities (E. Smith 1989), which are represented by components grouped under the class *NominalKindComponent*. The classes *NaturalKindComponent* and *NominalKindComponent* partition the class *EntityModelingComponent*.

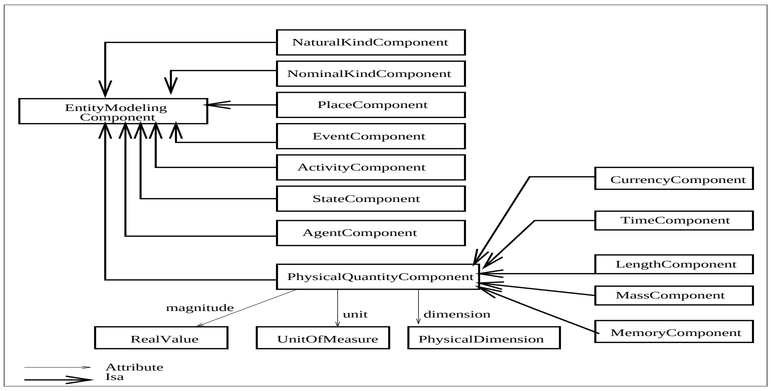

Figure 3: Partition of Entity Modeling Components

The class *PlaceComponent* groups components modeling locations. The class *EventComponent* groups components modeling events (i.e. everything in the real world occurring at a specific place and time or time period). The class *ActivityComponent* groups components representing processes in the real world. The class *AgentComponent* groups components representing agents interacting with information systems (e.g. humans, machines, organizations).

The class *PhysicalQuantityComponent* groups components expressing measures of some quantifiable aspect of a modeled reality. Physical quantities are distinguished according to their physical dimension. The physical dimension determines the type of units in which quantities are measured. Real-valued magnitudes express the amount of units in some physical quantity component. We distinguish five dimensions of physical quantities usual in requirements specifications. These include the dimensions of currency, time, mass, length and memory, which consequently distinguish five subclasses of the *PhysicalQuantityComponent*, namely the *CurrencyComponent, TimeComponent, MassComponent, LengthComponent* and *MemoryComponent*. Dimensions may be composed into other dimensions using multiplication (e.g. area = length * length) and/or be extended to cover specific domains of interest (e.g. the electrical currency dimension for the electrical engineering domain). A detailed ontology for physical quantities may be found in (T. Gruber and G. Olsen 1994).

Finally, the class *StateComponent* groups components reflecting states. We perceive states as bundles of co-occurring properties of real world concepts, which together denote a special situation for them. Therefore, we model them as separate specification components, which might aggregate other ones such as place components or physical quantities and so on.

The classes PlaceComponent, EventComponent, ActivityComponent, StateComponent, AgentComponent and PhysicalQuantityComponent constitute a partition of the class EntityModelingComponent.

General Classes of Component Modeling Relations. The class *RelationModelingComponent*, which groups components expressing relations of various arities, is also specialized. First, components expressing directed binary relations are grouped under the class *BinaryRelation*, which has two attribute classes representing the arguments of such relations (i.e. *fromComponent*, *toComponent* in Figure 2).

Binary relations are further grouped by subclasses according to cardinality constraints, general mathematical properties, existential constraints and other semantic properties (cf. Figure 4).

Cardinality constraints are reflected by classes defining three different partitions. The first

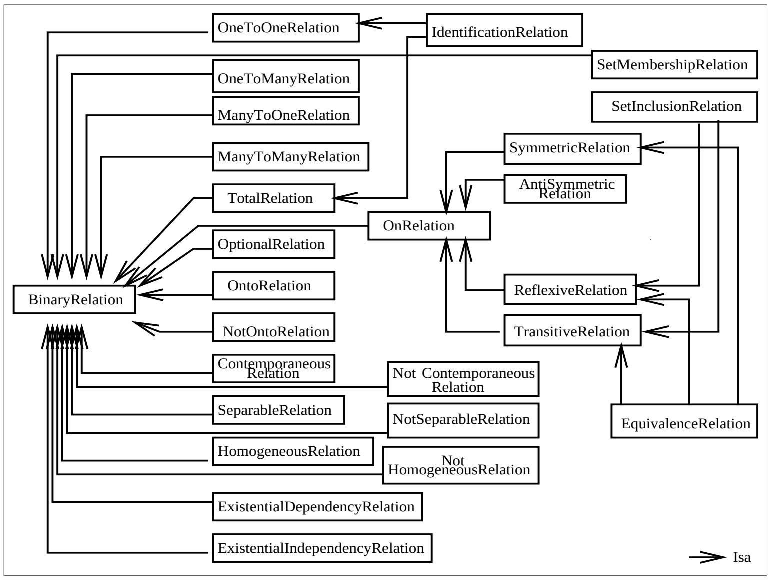

Figure 4: Special Kinds of Binary Relations

partition is defined by the classes *OneToOneRelation*, *ManyToOneRelation*, *ManyToManyRelation* and *OneToManyRelation*, which group 1:1, N:1, N:M and 1:N relations, respectively. The second partition is defined by the classes *TotalRelation* and *OptionalRelation*, which group relations that associate or not every element of their domain with some element in their range, respectively. The third partition is defined by the classes *OntoRelation* and *NotOntoRelation*, which group relations that associate or not every element in their range with some element in their domain, respectively.

A second group of subclasses reflects mathematical properties of relations defined over the same domain and range (grouped by the class *OnRelation* of Figure 4), including symmetry, transitivity and reflexivity (cf. classes *SymmetricRelation*, *AntiSymmetricRelation*, *ReflexiveRelation* and *TransitiveRelation* in Figure 4, respectively).

Binary relations can be further grouped under the classes *ExistentialDependencyRelation* or *ExistentialIndependencyRelation* according to whether or not the existence of their *toComponent* depends on the existence of their *fromComponent*. For instance, the relation *hasCopy* between a book and its copies is an existential dependency relation (i.e. there can be no copies of a non-existent book). On the other hand, the relation *authoredBy* between a book and its author is an existential independency relation.

Six more specializations of the BinaryRelation class define - in pairs - three orthogonal partitions of it, according to criteria discussed in (V. Storey 1993). The first of these criteria concerns whether the arguments of some relation must temporarily coexist or not. Binary relations whose arguments must coexist are grouped under the class *ContemporaneousRelation* (e.g. committee *hasMember* consultant), while those whose arguments may not coexist are grouped under the class *NotContemporaneousRelation* (e.g. ancientArtifact *hasBeenStudied By* archaeologist, since the entire ancient artifact may not exist at the time the archaeologist studies it based on some of its parts).

The second criterion concerns whether the arguments of a relation are at least of one common kind (i.e. classified under at least one common class) or not. If they are, the relevant relations are grouped under the class *HomogeneousRelation*. If they are not, they are grouped under the class *NotHomogeneousRelation*. Components related by homogeneous relations share a common set of properties (e.g. bread *has* slice).

The third criterion distinguishes between relations, whose arguments can be physically disconnected and relations whose arguments cannot, grouped by the classes *SeparableRelation* (e.g. bicycle *hasPart* wheel) and *NotSeparableRelation* (e.g. bicycle *isMadeOf* aluminium), respectively.

Finally, binary relations can be grouped under the classes *EquivalenceRelation* (i.e. the class of reflexive, symmetric and transitive relations), *SetMembershipRelation* (i.e. the class of relations with a set membership semantics), *SetInclusionRelation* (i.e. the class of relations with a set inclusion semantics), and *IdentificationRelation* (i.e. the class of 1:1 and total relations enabling the unique identification of their *fromComponent* by their *toComponent*).

Representation Model Dependent Extensions of the Meta-Model. The meta-model also includes classes expressing modeling constructs of specific techniques for representing specifications.

Currently, these classes concern a hypothetical object-oriented data and the relational data model.

Figure 5 presents the extension for a hypothetical object-oriented representation model, supporting object types (cf. class *ObjectType*), possibly generalized by Isa relations (cf. class *IsaRelation*) and having attributes (cf. class *ObjectAttribute*). The class IsaRelation is defined as a subclass of the *SetInclusionRelation* to express that isa relations in the particular representation model have a set inclusion semantics (i.e. every instance of a subtype has to be an

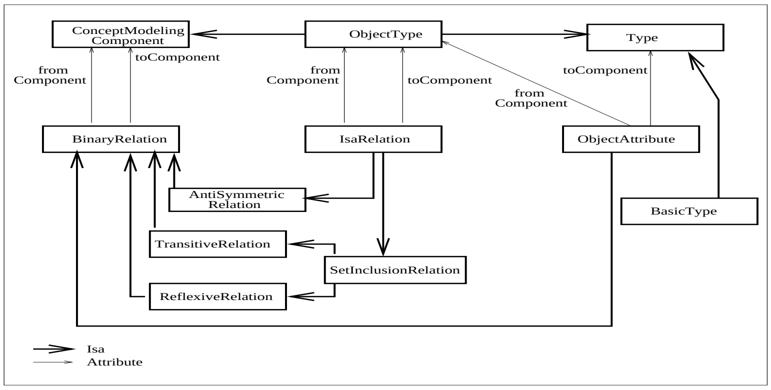

Figure 5: Extension for an Object-Oriented Representation Model

instance of its supertypes). It is also declared as a subclass of the *AntiSymmetricRelation* class to express that an object type can not be a specialization of itself. Furthermore, isa relations can only have object types as *from* and *to* components something indicated by the specialization of the attributes *fromComponent* and *toComponent*, which are inherited to *IsaRelation* from the class *BinaryRelation*. Also, attributes of object types may have as values (i.e. *to* components) either objects or basic type values (e.g. strings, reals etc.).

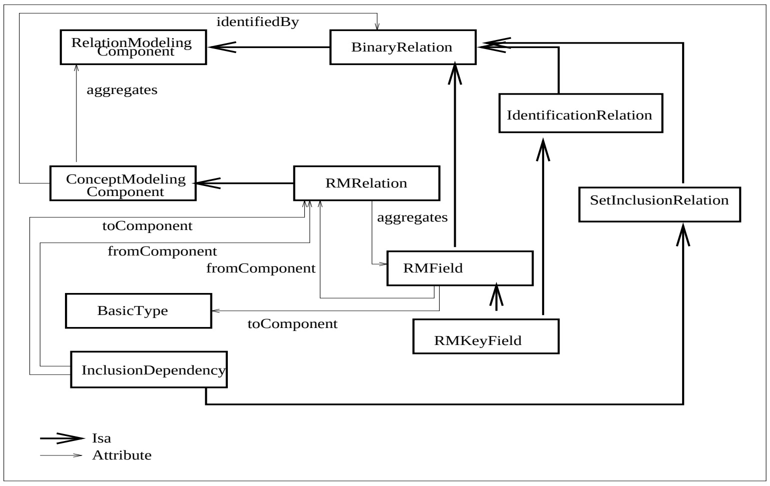

Figure 6: Extension for the Relational Representation Model

Figure 6 presents the extension of the meta-model for incorporating constructs available in the relational data model (F. Codd 1979). Those constructs include *relations, fields* and *inclusion dependencies*. The class *RMRelation* is defined as a subclass of the class *ConceptModelingComponent*, thus indicating that relations in relational specifications can express either relationships or entities. The class *RMRelation* restricts the attribute *aggregates* inherited from the class *Con-*

ceptModelingComponent to take values in the class *RMField*, which expresses the fields in the relational data model. Since fields express binary relations, the class *RMField* is declared as a subclass of the class *BinaryRelation*. Finally, inclusion dependencies are represented by the class *InclusionDependency*, which is declared as a subclass of the *SetInclusionDependecy*, due to their set inclusion semantics (F. Codd 1979).

The choice of the particular meta-classes has been influenced by their utility as general purpose semantic modeling constructs. This utility is evident from the presence of relevant concepts in many different semantic and object-oriented data models (R. Hull and R. King 1987, J. Peckham and F. Maryanski 1988, D. Monarchi and G. Puhr 1992), as well as the validity of such concepts across different application domains.

2.3 Representation of Specifications

Specifications are described as instances of the meta-model class *Specification* (cf. Figure 7). This class introduces a set of attribute classes which take values in the distinct entity and relation classes of the meta-model. For example, the attribute class *oneToManyRelation* takes as values instances of the class *OneToManyRelation* and the attribute class *naturalEntity* takes as values instances of the class *NaturalKindComponent*. These attribute classes are instantiated by attributes of concrete specifications subject to the type of the component they aggregate in them.

As an example consider the description of a relation between a library borrower and the code assigned to him/her, which is presented in Figure 8. The object type *Borrower* is aggregated in *Specification1* as the value of its attribute *entity1*. This attribute is classified under the attribute classes *agentEntity* and *naturalEntity* to denote that it aggregates a natural kind, agent modeling component. Consequently, *Borrower* is classified under the component classes *NaturalKindComponent* and *AgentComponent*. Similarly, the object type *LibraryCode* is aggregated in *Specification1* as the value of its attribute *entity2*. This attribute is classified under the attribute class *nominalEntity* and therefore *LibraryCode* was classified under the class *NominalKindComponent*.

Finally, the attribute *HasCode* of the object type *Borrower* is aggregated in *Specification1* as the value of its attribute *relation1*. The attribute *relation1* is classified under the attribute classes *identificationRelation*, *totalRelation*, *ontoRelation*, *notHomogeneousRelation*, *separableRelation*, *contemporaneousRelation* and *existentialIndepRelation* to denote that it aggregates a component modeling a relation of the relevant types. Consequently, its value *HasCode* is classified

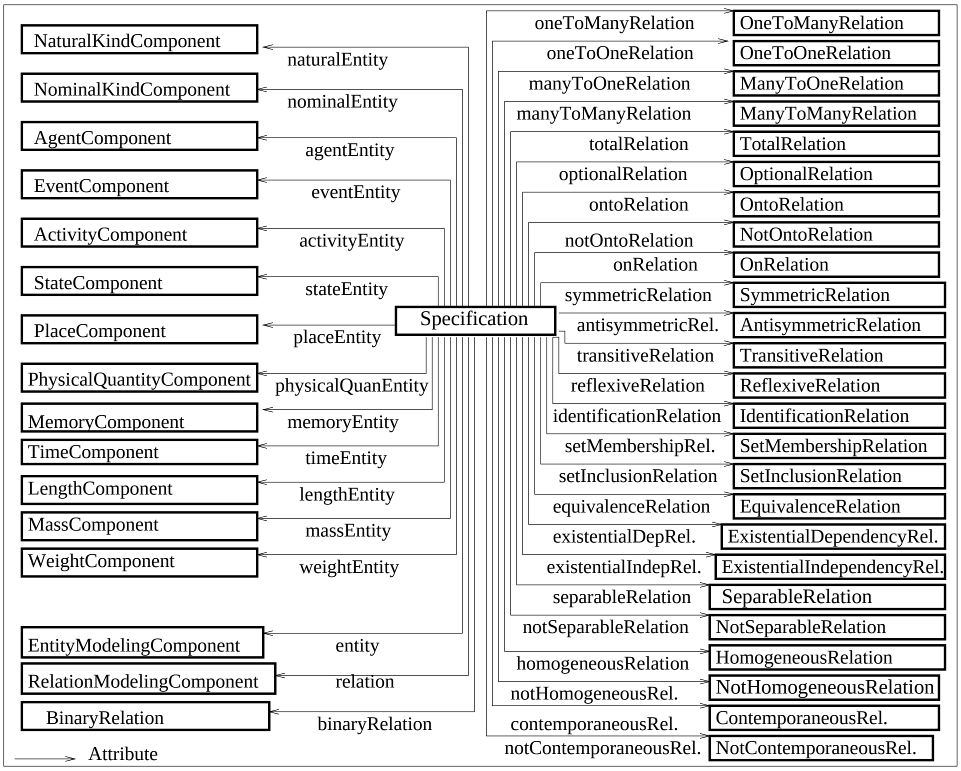

Figure 7: The Class Specification

under the classes *IdentificationRelation*, *TotalRelation*, *OntoRelation*, *NotHomogeneousRelation*, *SeparableRelation*, *ContemporaneousRelation* and *ExistentialIndepRelation*.

By classifying attributes of specifications under different attribute classes of *Specification* according to the kind of components they aggregate, we ensure that the similarity analysis of specifications will attempt mappings only between their components which are of the same kind (cf. the criterion of *semantic homogeneity* in Section 3.1 and the appendix).

3. Similarity Analysis of Specifications

3.1 The Computational Model of Similarity

Specifications are compared according to a computational model of similarity, which detects analogies between objects described by attributes as well as classification and generalization relations (G. Spanoudakis and P. Constantopoulos 1994a, G. Spanoudakis 1994a). The similarity model is based on three distance functions, namely the *classification*, *generalization* and *attribution* metrics. These metrics measure the conceptual distance between specifications with respect to their classification, generalizations and attributes.

The classification distance between specification components indicates their differences with

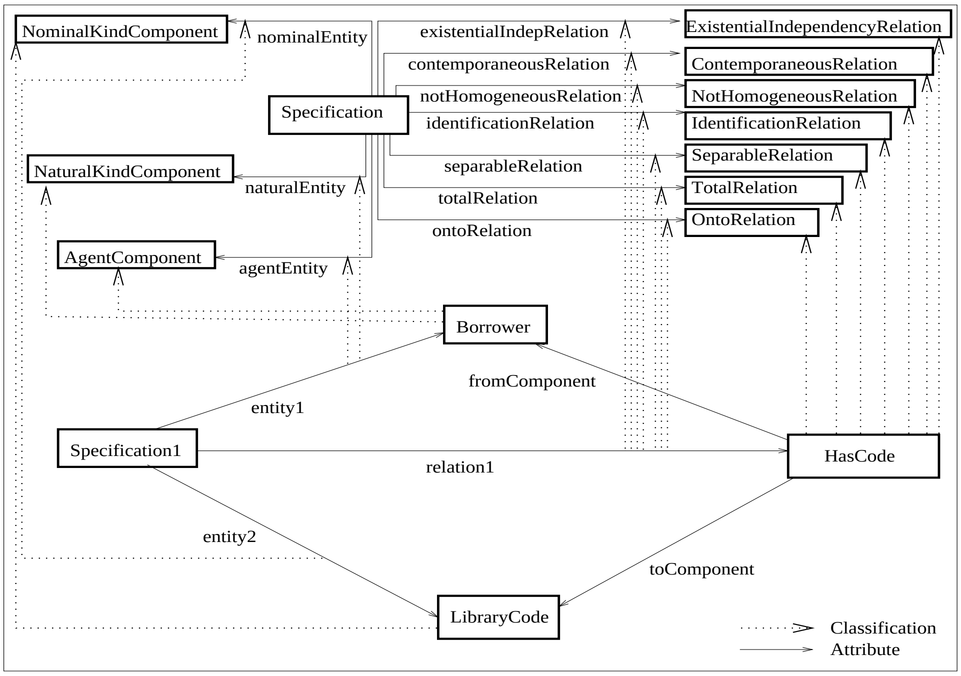

Figure 8: A Specification of Library Borrowers and their Codes

respect to the properties expressed by the relevant classes of the meta-model. It is computed by identifying the non-common classes of two components, estimating the importance of these classes, and aggregating the obtained importance measures into a classification distance measure (cf. function d_c in the appendix).

The generalization distance between specification components indicates their rough semantic differences as suggested by their non-common superclasses. It is computed like the classification distance, except that the superclasses of components are taken into account (cf. function d_g in the appendix).

The attribution distance between specifications gives a detailed account of their analogous and non-analogous components. It is computed by isomorphically mapping the attributes of specifications which are most similar to each other. Specification attributes cannot be mapped unless they are instances of the same attribute classes (i.e. the criterion of *semantic homogeneity*). Such a common classification makes likely that, despite their possible modeling disparities, mapped attributes express aspects of the same kind (M. Turner 1988). Due to the very description of specifications (see Section 2.3), the criterion of semantic homogeneity precludes mappings between specification components unless they both represent exactly the same kinds of entities or relations. In cases where the attributes of specifications can be mapped in more than one possible ways, the model selects the mapping with the minimum total distance (cf.

the criterion of the *minimum distance isomorphism* and the function D_a in the appendix). The estimation of the pairwise distances between specification attributes is based on the estimation of their own partial distances as well as the partial distances of their component-values. Therefore, it recursively generates isomorphic mappings between the attributes of these values, too.

Partial distances are aggregated in an overall distance measure (cf. function d in the appendix), which indicates the overall semantic discrepancy of specifications.

Similarity analysis provides a reasoning mechanism capable of dealing with the problem of semantic heterogeneity. Matched specification components must be of the same type, but they do not have to be identically modeled. The selection of the minimum distance isomorphism among them guarantees that the most similar components will be matched with each other, in a globally optimal way. Finally, the aggregation of the partial classification, generalization and attribution distances makes analysis relatively tolerant to the possible incompleteness of specifications. Even in the absence of some classification, generalization relations or attributes of them, their similarity can be revealed by their remaining description elements.

The similarity model has certain computational characteristics which contribute to its practical utility. First, in order to detect similarities, it doesn't need information which wouldn't be present in specifications otherwise (e.g. user-supplied distances, importance measures for distinct specification components or special relations directly indicating analogies between specification components such as annotations with terms in fixed *application vocabularies* (J. Leite and P. Freeman 1991)). The incorporation of such information in specifications, solely for the sake of detecting similarities, would be a burden for specification developers. Second, it has a polynomial-time computational complexity (G. Spanoudakis 1994a). This complexity results from terminating recursion, while estimating the attribution distance of specifications at a fixed depth of their attribution graphs. Also, since the process of mapping a particular pair of attributes does not affect the pairwise distances between other attribute pairs, the selection of the optimal isomorphism between the attributes of two specifications can be carried out by an algorithm solving the *weighted bipartite graph matching problem* (G. Spanoudakis 1994a). Such algorithms do not generate gradually all the possible isomorphisms between attributes, and have polynomial complexity (C. Papadimitriou and K. Steiglitz 1982). Thus, the model scales efficiently to problems of large size.

3.2 Tool Support: The Semantic Index System

Our approach to specification integration has been implemented using the *Semantic Index System (SIS)*, a tool for representing, storing and retrieving objects described according to Telos (P. Constantopoulos and M. Doerr 1993). The SIS was initially developed to enable the construction of servers supplying reusable software artifacts, known as *software information bases*, in the Esprit project ITHACA (P. Constantopoulos 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), importing them with reasonable speed (0.5m objects in 2.5 hours on SPARC stations) and querying them without significant performance degradations depending on the size of the stored population (the transitive closure of a binary tree with 1024 nodes is retrieved in 2 secs on SPARC stations). The system incorporates a mechanism of interactive data entry forms, which can be customized to the needs of different applications by specifying tasks of information entry in Telos.

Due to its integration with an implementation of the similarity model(G. Spanoudakis 1994b), the SIS provides queries for detecting similarities between Telos objects.

The prescribed meta-model for specification analysis has been implemented as a kernel SIS object base, which is used as a schema for describing specifications. Specifications described as SIS objects according to this schema are amenable to similarity analysis.

4. An Example of Similarity Analysis of Specifications

4.1 The Analysed Specifications

In the following, we demonstrate the similarity analysis of specifications using an example of two specifications concerning library borrowers, items and their relations. Both of them were described according to the hypothetical object-oriented representation model of Section 2, as shown in Figure 9.

They resemble in including components representing library borrowers (i.e. object types *Borrower* and *Student*), different types of library items (i.e. the object types *CopyOfBook*, *Publication* and their subtypes) and borrowing relations between them (i.e. object type *Loans* and object attribute *Borrows*). However, they are not modeled identically (e.g. different taxonomies of library items, different attributes for students and borrowers).

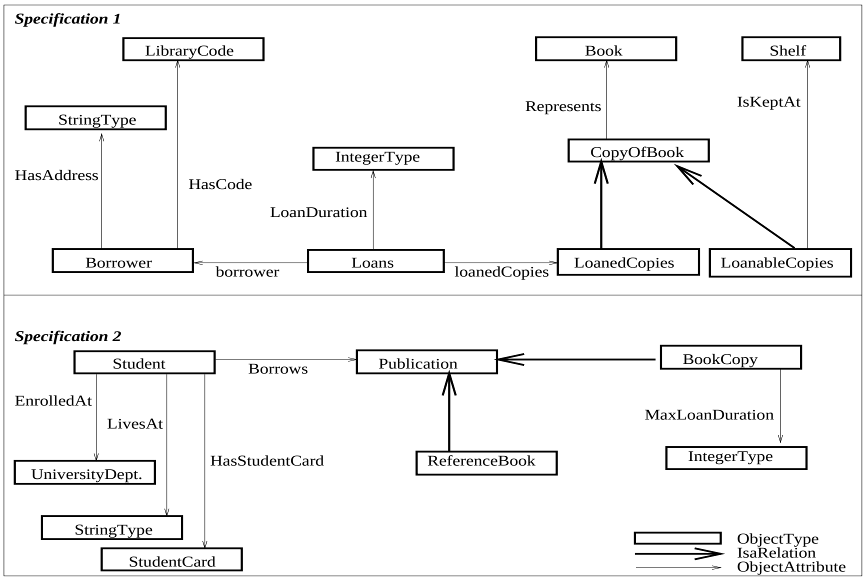

Figure 9: Specifications of Library Borrowers, Items and their Relations

4.2 Describing Specifications as Instances of the Meta-Model

The specifications of Figure 9 were described as instances of the specification analysis meta-model using the SIS mechanism of interactive data entry forms. These forms were customized by a task model prescribing various creation and update operations, decomposed into primitive classification and attribution actions concerning specific classes of the specification analysis meta-model. In summary, the available operations allow specification developers classify specification components under the various partitions of the entities and relations of the meta-model and create or modify their attributes. In the following we give an example of classifying and declaring the value of the object attribute *Borrows* in Figure 9.

As shown in Figure 10, the selection of the *Declare Cardinalities* operation against the object attribute *Borrows* in *Specification2* from the form *Create / Update Relation Modeling Components* displayed a *Selection List* with the classes of the meta-model distinguishing relations according to their cardinality. By selecting one of them (*OneToMany* in our case), *Borrows* was classified under it. Subsequently, its classification under any other subclass of *BinaryRelation* expressing cardinalities was forbidden.

Figure 10 also shows the definition of the attribute aggregating the value of *Borrows*. Components' attributes have to be created or modified as instances of one or more of the attribute classes defined in their own classes. Thus, by selecting the operation *Edit Attributes* from the

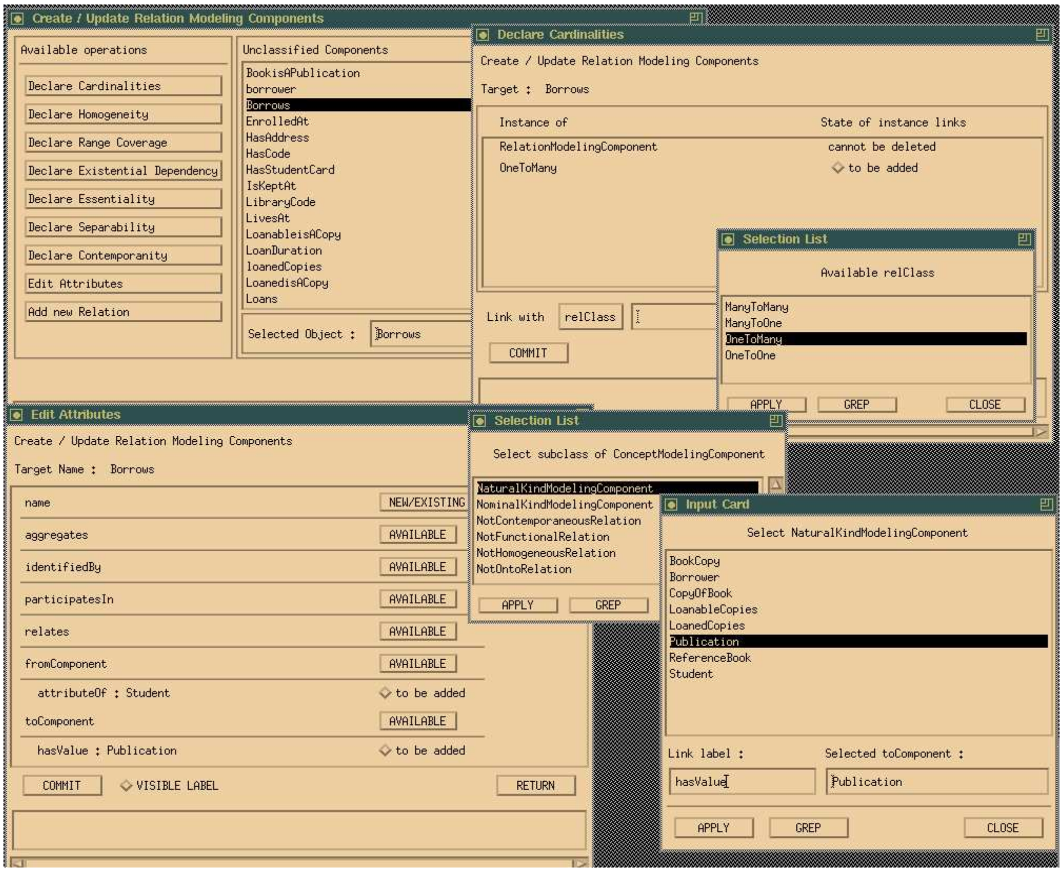

Figure 10: Description of the Borrows Relation Using Data Entry Forms

Create/Update Relation Modeling Components form, the possible attribute classes for classifying the attributes of *Borrows* are displayed (e.g. *fromComponent*, *toComponent* in *Edit Attributes* form). From those classes we selected the *toComponent* and declared an attribute called *hasValue* as an instance of it (see *InputCard* form). The value of this attribute (*Publication*) was selected from the existing instances of the *NaturalKindModelingComponent* (see *InputCard* form). Alternatively, we could have created a new component or pick up an instance of the other concept modeling component classes from the *Selection List* form. The other components of specifications 1 and 2 in Figure 9 were described using the entry forms in a similar manner.

This information entry mechanism presents specification developers with all the classification and attribution possibilities for the components in their specifications, precludes incorrect classifications of components (e.g. simultaneous classifications under classes of the same partition of the meta-model) and makes possible the construction of descriptions without having to write them in the syntax of the Telos language.

4.3 Detected Similarities and Discrepancies

The similarity analysis of specifications 1 and 2 of Figure 9 generated the isomorphism shown in Figure 11.

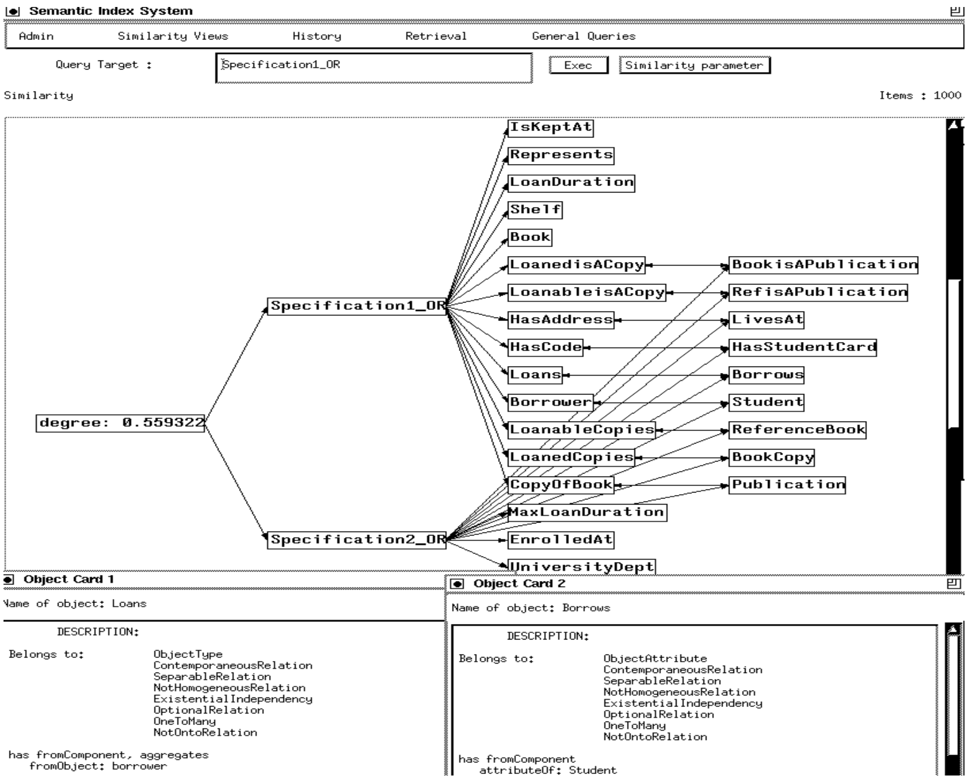

Figure 11: The Similarity Isomorphism between Specifications of Figure 9

In particular, *Borrower* was mapped onto *Student* as the only pair of object types representing natural kind entity and agent modeling components in the involved specifications (both of them had been classified under the classes *ObjectType* , *NaturalKindComponent* and *AgentComponent* of the meta-model).

Also, the attributes *HasCode* and *HasAddress* of *Borrower* were mapped onto the attributes *HasStudentCard* and *LivesAt* of *Student*, respectively. The former mapping resulted because the involved object attributes were the only relations simultaneously classified as identification (i.e. 1:1 and total), separable, not homogeneous, contemporaneous and existentially dependent ones. Unlike them, the mapped attributes *HasAddress* and *LivesAt* were not the only M:1, not onto, total, separable, not homogeneous, not contemporaneous and existentially independent binary relations of the involved specifications. In fact, the object attribute *LoanDuration* in *Specification1* had been classified as such a relation, too. Thus, *LivesAt* could have been mapped on it. However, the higher similarity between the from-components of *HasAddress* and *LivesAt* (i.e. *Borrower* and *Student*) and the identity of their to-components (i.e. *StringType*) enforced their mapping as more plausible than the mapping of *LivesAt* onto *LoanDuration* (cf. the criterion of *minimum distance isomorphism* in the appendix).

The object type *Loan* was mapped onto the object attribute *Borrows* since they had been both classified as 1:N, optional, not onto, separable, not homogeneous, contemporaneous and existentially independent binary relations (see object cards in Figure 11). Furthermore, they had similar from-components: *Borrower* and *Student*.

The natural kind entity modeling components *CopyOfBook*, *LoanedCopies* and *LoanableCopies* were mapped onto *Publication*, *BookCopy* and *ReferenceBook* respectively, because of the resemblance of their generalization taxonomies (see the mappings of the isa relations *LoanedisACopy* and *LoanebleisACopy* onto *BookisAPublication* and *RefisAPublication* in Figure 11). However, notice that their attributes were not mapped onto each other, since they had not been classified under the same classes of the meta-model. For instance, although *LoanedCopies* was mapped onto *BookCopy*, its inherited attribute *Represents* was not mapped onto the attribute *MaxLoanDuration* of *BookCopy* because the former had been classified as a non-separable relation, unlike the later.

Finally, the object types *Shelf* (a place, nominal component), *Book* (a nominal component) and *UniversityDept* (an agent, nominal component) were left unmapped due to their different classifications.

4.4 Discussion: Factors Affecting Similarity Analysis

The most significant factor in mapping specification components is their classification under the classes of the specification analysis meta-model.

A common classification may lead to mappings between components regardless of their names or other aspects of their modeling. For instance, the object type *Loans* was mapped onto the object attribute *Borrows* in our example, in spite of their modeling disparities (i.e. modeled by an object type and an object attribute, having different names). A non-common classification precludes mappings between components. For instance, *Book* was not allowed to be mapped onto *Publication* or any of its subclasses because it had been classified as a nominal kind component, unlike those that had been classified as natural kind components.

Thus, if incorrect, the classification of specification components may lead to mappings which are not valid or preclude valid ones. For instance, if *MaxLoanDuration* had been incorrectly classified as a total (instead of an optional relation) it would have been mapped onto *LoanDuration*, although the former expresses the maximum duration of loans for a book copy in general while the second the duration of a particular loan. Also, if *Loans* had been classified as a N:M relation (instead of an 1:N) it wouldn't have been mapped onto *Borrows*.

The granularity of specification descriptions plays an important role in selecting among mappings between components, which are considered valid regarding their classification. In general, the more detailed the granularity of specifications, the more likely that similarity analysis will produce an accurate mapping between them. Consider the incorrect mapping between the generalization taxonomies of library items in our example. Intuitively, the *CopyOfBook* in *Specification1* should have been mapped on the *BookCopy* rather than the *Publication* in *Specification2*. However, the missing attributes of the components involved, the missing classes in their generalization taxonomies, and the structural resemblance of these taxonomies led to the incorrect mappings.

After expanding the descriptions and the generalization taxonomies of the components with attributes and the classes, as shown in Figure 12, similarity analysis detected the intuitively correct mapping between them.

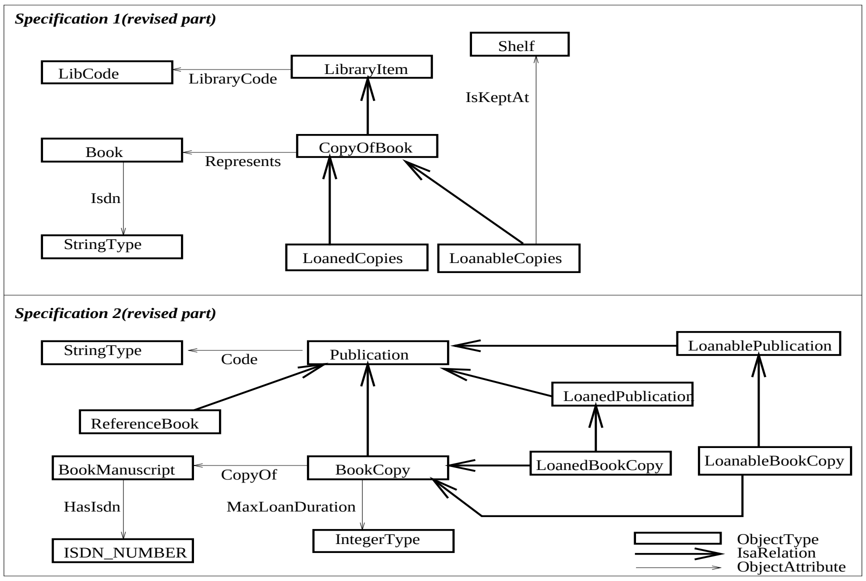

Figure 12: Revised Specifications of Library Items

In particular, *LibraryItem* was mapped to *Publication*, *CopyOfBook* to *BookCopy*, *LoanedCopies* to *LoanedBookCopy* and *LoanableCopies* to *LoanableBookCopy*. Also, *Book* was mapped to

BookManuscript. Notice that, *LoanedCopies* was found more similar to *LoanedBookCopy* than to *LoanedPublication* because *LoanedPublication* didn't have any attribute corresponding to its attribute *Represents*, unlike the *LoanedBookCopy*, which had the attribute *CopyOf* (inherited from *BookCopy*).

Also, the attribute *LibraryCode* was mapped to the attribute *Code* (as identification, total, separable, not homogeneous, not onto, contemporaneous and existentially independent relations); the attribute *Represents* was mapped to *CopyOf* (as N:1, total, not separable, not homogeneous, not onto, contemporaneous and existentially independent relations); and the attribute *Isdn* to *HasIsdn* (as identification, total separable, not homogeneous, not onto, contemporaneous and existentially dependent relations).

The names of specification components do not affect the estimation of their distances. Even the identity of components depends on the equality of the internal identifiers of the Telos objects representing them (cf. Section a1 in the appendix).

5. Dealing with Similarities and Discrepancies

Detected isomorphisms can be used as a basis for reconciling specifications. The analysis of similarity distinguishes between unique specification components (i.e. ones having no counterparts in the other) and corresponding components (i.e. ones mapped) (cf. Figure 13). However, the generated mappings may be partially incorrect, according to the opinion of specification owners, who subsequently may distinguish among:

- *false unique components*, which should have counterparts although they don't;
- *false corresponding components*, which should have either no counterpart or different mapped counterparts;
- *correct unique components*, which are correctly left unmapped;
- *correct corresponding components*, which are correctly mapped.

By tracing the result of similarity analysis back to the modeling of components, it is possible to aid owners in dealing with false components. For instance, two unique components may not have been mapped onto each other because of their non-common classification. If, in our preceding example, *Student* had not been classified as an agent, it wouldn't have been mapped onto *Borrower*. Tracing such a non-common classification of false unique components may reveal modeling errors and let specification owners fix them. In other cases, a component may not be mapped to its correct counterpart (in the opinion of the specification owner) because it was found to be closer to another one (cf. the criterion of the *minimum distance isomorphism*). By comparing its partial distances with each of these components, it might be possible to identify incorrect aspects of their modeling which made the difference and rectify them.

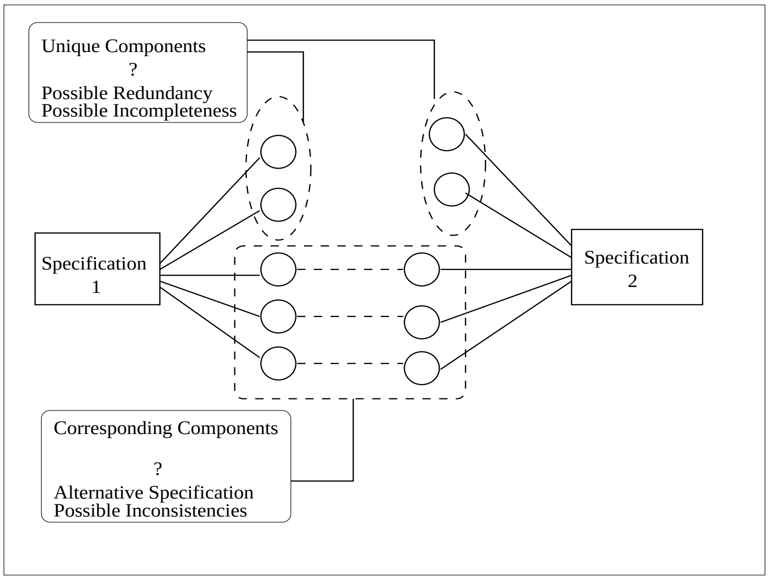

Figure 13: Corresponding and Unique Specification Components

Once false mappings have been eliminated, the correct unique and corresponding components can be used as starting points for exploring hypotheses for reconciling specifications. Unique components might indicate *missing* or *redundant* information, or even *homonyms* (if they have identical names). For instance, in our previous example *Loans* had an attribute indicating the duration of borrowing (i.e. *LoanDuration*), which was missing from the specification of *Borrows*.

Corresponding components may also indicate the need for modeling changes or lead to the detection of inconsistencies. The need for modeling changes can be revealed by investigating the similarity isomorphism between the substructures of corresponding components. False parts of this isomorphism must also be rectified first. Then various reconciling hypotheses might be explored, in addition to those mentioned for unique components. For instance, mapped components with different names might indicate a *synonymy problem*, as in the case of *Loans* and *Borrows*. Further hypotheses concern the merging of components with totally isomorphic structures or the specialization of components by their counterparts if the structures of the latter contain the structures of the former. For instance, *Student* may be reasonably considered as a specialization of *Borrower*.

In our view, the reconciling and merging of specifications has to be carried out by their owners.

Currently, we are developing a process model describing the activity of rectifying false unique and corresponding components and exploring the prescribed (and other) reconciling hypotheses and actions, in response to the isomorphisms generated by similarity analysis. The model will provide guidance to specification owners without forcing them to adopt specific, predefined types of reconciliation (A. Finkelstein et al. 1994).

6. Related Work

The problem of detecting and resolving semantic conflicts between specifications has become an important issue in requirements engineering research, following a recent interest in requirements specification from multiple viewpoints (B. Nuseibeh et al. 1993, N. Maiden et al. 1994, G. Kotonya and I. Sommerville 1992). The detection and resolution of semantic conflicts is also central to research on obtaining and maintaining the consistent exchange of information among distributed, autonomous databases composing *multiple database systems* (i.e. *semantic interoperability*) (A. Sheth and J. Larson 1990, C. Goh et al. 1994).

6.1 Integration of Multiple Perspective Specifications

Subject to the technique employed for detecting specification discrepancies, the various approaches could be distinguished into the *logic-based*, the *transformation-based* and the *inexact-matching* ones.

The *logic-based* approaches can deal only with a subset of specification discrepancies — the logical inconsistencies — but fail to cope with the problem of *semantic heterogeneity*, as discussed in Section 1. Some of them adopt an underlying first-order representational formalism, into which multi-representation specifications must be transformed before their inconsistencies can be detected and resolved (P. Zave and M. Jackson 1993). Other approaches, realizing the difficulty of transforming multi-representation specifications into a single logical formalism, and the undecidability of inconsistency given arbitrary sets of first-order formulas, focus on the detection of particular types of inconsistencies between specifications expressed in specific representation models (A. Finkelstein et al. 1994, W. Robinson and S. Fickas 1994, C. Heitmeyer et al. 1995, S. Easterbrook and B. Nuseibeh 1995).

The *transformation-based* approaches compare specifications using *canonical* representations in a common underlying language. Canonical representations are generated by predefined transformation rules (P. Johanneson 1993, S. Meyers and S. Reiss 1991). These approaches are inevitably limited by the completeness and acquisition of transformation rules. Also, the validity of their canonical forms across different representation models and application domains is not clear.

The *inexact-matching* approaches, realizing the broad spectrum of possible specification discrepancies, attempt to derive mappings between specification components which satisfy properties indicating that mapped components express the same underlying entity. Such mappings are derived by matching specification components onto the same *problem abstractions* in (N. Maiden and A. Sutcliffe 1994) (i.e. models of fundamental behaviour, structure, goals and constraints of classes of requirements engineering problems) or by heuristically found analogies between components annotated by the same term(s) in domain specific vocabularies (J. Leite and P. Freeman 1991). The first of these approaches is only applicable to specifications expressed in its own domain modeling language (N. Maiden et al. 1995). The second is limited by the need to define and agree upon terms for a very large number of concepts in every single application domain. As terms often tend to be ambiguous, devising vocabularies is a difficult task even in relatively simple domains (G. Furnas et al. 1987). Our approach is also an *inexact-matching* one, introduced to overcome such problems.

6.2 Semantic Interoperability Strategies for Multiple Database Systems

The strategies developed to obtain the semantic interoperability of multiple database systems can be distinguished into the *tight-coupling*, the *loose-coupling* and the *knowledge representation* ones (C. Goh et al. 1994).

The *tight-coupling strategies* integrate local database schemas into one or more global schemas — usually referred to as *federated schemas* (A. Sheth and J. Larson 1990) — after identifying and reconciling their semantic and representation differences. Such *view* or *schema integration* methods have been reviewed in (C. Batini et al. 1986, A. Sheth and J. Larson 1990). The detection of semantic equivalencies or disparities between local schemas is facilitated by their representation in a single data model (usually the entity-relationship or some semantic data model) capable of expressing their semantics (e.g. A. Motro and P. Buneman 1981, S. Yao et al. 1982). In addition, some strategies rely on explicitly asserted relationships between local schema components, such as mappings between interattribute values (J. Larson et al. 1989), functional, inclusion or exclusion dependencies (S. Al-Fedaghi and P. Scheuermann 1981, M. Casanova and M. Vidal 1983), equivalence or containment relations (S. Navathe and S. Gadgil 1982, M. Mannino and W. Effelsberg 1984). Yet none of the proposed algorithms and heuristics fully automates local schema integration (A. Sheth and J. Larson 1990).

The *loose-coupling strategies* provide semantic interoperability by ensuring that exchange is possible only between semantically equivalent pieces of information, which are appropriately converted from the context of their supplier to the context of their consumer. Originally, loose-coupling strategies provided *multi-database manipulation languages*, such as MDSL (W. Litwin and A. Abdellatif 1987). These languages allow users formulate queries in terms of local database schemas as well as devise ways of detecting semantic conflicts and converting information between different representations (C. Goh et al. 1994). The complete delegation of responsibility for semantic conflict detection and resolution to the user makes the multi-database manipulation language approach less usable. Recently, loose-coupling strategies have been based on the explicit annotation of local schema components with terms in *shared ontologies* (M. Bright et al. 1994, E. Sciore et al. 1994). Such annotations are employed for deducing the semantic equivalence of information. The transformation of semantically equivalent information between different local representations is based on libraries of conversion functions, supplied by local systems delegates (E. Sciore et al. 1994).

Finally, the *knowledge representation strategies* (e.g. Carnot (C. Collet et al. 1991) and SIMS projects (Y. Arens and C. Knoblock 1992)) achieve semantic interoperability by transforming local schemas to a global schema, which unifies their disparate interpretations and representations. In essence, this global schema is a knowledge base describing concepts in various application domains, structured using abstraction mechanisms similar to those used in our approach (e.g. the Cyc knowledge base (D. Lenat and R. Guha 1990)). Local schemas are compared and transformed to the global schema without being explicitly interrelated with each other. Thus, the global schema is easier to construct and maintain (C. Goh et al. 1994). Also, in some cases it evolves by incorporating knowledge of new local schemas absent from it (C. Collet et al. 1991).

7. Conclusions and Further Research

In this paper, we discussed a similarity reasoning approach to the problem of integrating requirements specifications. Specifications are classified under a meta-model of domain independent abstractions denoting general semantic properties of their components. Given this classification, specifications are compared by a computational model of similarity, which detects their semantic resemblances and discrepancies, subsequently used as a basis for reconciling them. Important aspects of the approach include: (1) the extensibility of its meta-model in accommodating different representation models for specifications and (2) the polynomial complexity of similarity analysis, which makes it scalable to large complex specifications.

On-going work focuses on supporting the activity of reconciling specifications based on the results of their similarity analysis in the direction outlined in Section 5. Also, we extend the meta-model with constructs for describing state-transition diagrams, as a means of being able to analyse specifications reflecting dynamic aspects of information systems, as well. To further support the description of specifications as instances of the introduced meta-model, we investigate into the possibility of automatically translating specifications under its representation model dependent parts using grammar transformations.

Other aspects of our approach which need further investigation, include: elaboration and enhancement of the meta-model for specification analysis; a detailed study of the impact of specifications granularity on similarity analysis and large scale validation.

Acknowledgements

This research has been partially funded by the European Union Esprit III basic research action NATURE(6353). The authors wish to thank Anthony Finkelstein from the City University and anonymous referres, whose comments let them improve this paper in content and presentation.

Appendix: The Computational Model for Similarity Analysis

Similarity analysis is applicable onto objects described in the conceptual modeling framework of Telos (J. Mylopoulos et al. 1990). The interested reader can find a detailed formal description of the language in (M. Koubarakis et al. 1989). Here, we introduce in a set-theoretic manner the basic structure of Telos objects to allow the reader follow the subsequent treatment of the distance functions composing the similarity analysis model.

a1. Representation of 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. Telos objects 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). On the basis of these partitions we can distinguish four basic categories of objects having 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 a system identifier uniquely identifying the object o_i, *In* is a set of system identifiers denoting the transitive closure of the classes of object o_i (o_i is said to be an *instance* of the classes in *In*), *Isa* is a set of system identifiers denoting the transitive closure of the superclasses of object o_i , A is a set of system identifiers denoting the direct attributes (i.e. those not inherited) of object 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 of attribute o_i (or the range-class of attribute classes).

Telos objects have *logical names*, which enable external logical references to them. Logical names are unique to individual objects in an object base but they may be shared by more than one attribute objects as long as their owning classes are different. Each Telos class i has an *intension INT[i]*, that is a set including the identifiers of the attributes it introduces or inherits (and possibly refines) from its superclasses. Each Telos attribute class i has an *original class* defined as its most general superclass of i, with respect to the Isa partial ordering, which has an identical logical name with it. Assuming a 1-1 mapping $l : ID \longrightarrow L$ where ID and L are the sets of the system identifiers and the logical names of the objects respetively, the original class of an attribute class i, OC(i), is an attribute class j satisfying the following condition:

$$(j \in o_i.Isa) \; and \; ((l(i) = l(j)) \; and \; (\neg(\exists x_1 : (x_1 \in o_i.Isa) \; and \; (x_1 \in o_j.Isa) \; and \; (l(i) = l(x_1)))))$$

These modeling constructs can be illustrated with reference to the objects in Figure 14 (it presents a version of the meta-model introduced in Section 2 of the paper adapted to allow the reader to follow the subsequent computations in a comprehensible manner). In this figure, the meta-classes *EntityModelingComponent* and *aggregates* of *EntityModelingComponent* are I_c and A_c objects respectively, having the following tuple forms:

$O_{EntityModelingComponent}$ = [{ MetaMetaClass, Individual}, { MetaClass, ConceptModelingComponent }, { EntityModelingComponent.identifiedBy, EntityModelingComponent.aggregates }]

$O_{EntityModelingComponent.aggregates}$ = [EntityModelingComponent, { MetaMetaClass, Attribute }, { MetaClass, ConceptModelingComponent.aggregates}, { }, EntityModelingComponent]

In these tuple forms, object identifiers have been substituted by logical names in the case of individual classes and by concatenations of logical names in the case of attribute classes (i.e. X.Y were Y is the logical name of the attribute class and X the logical name of its owning class) to avoid ambiguities and enable the understanding of object structuring. Also, *Individual* and *Attribute* are the logical names of special Telos meta-classes of all the entity and attribute objects, respectively.

The instances of the class *EntityModelingComponent* can instantiate its attribute meta-classes *identifiedBy* and *aggregates*. For example, the class *Student* (cf. Figure 16) has its attribute class *hasStudentCard* been classified under both the attribute metaclasses *identifiedBy* and *aggregates* of *EntityModelingComponent*. Also, the original class of the attribute class *identifiedBy* of *EntityModelingComponent* is the attribute class *identifiedBy* of *ConceptModelingComponent*.

a2. Classification Distance

The classification distance gives an estimate of the analogy over the substance of two objects by measuring and aggregating the importance of their non-common classes. Class importance is measured as the inverse of the maximum depth of a class in a generalization taxonomy (i.e. specialization depth *SD(x)* in definition 1). Hence, classes placed at higher levels in generalization taxonomies are considered as more important than classes placed at lower levels. This weighting coincides with cognitive studies of classification in human mental models, which predict that the more general a class, the more significant the classification it express in a taxonomy (E. Rosch et al. 1976). Formally, the classification distance is defined as:

Definition 1: The classification distance, d_c, between two objects o_i and o_j is defined as:

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

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

$$o_i.In \ \triangle \ o_j.In \ = \ (o_i.In \ - \ o_j.In) \bigcup (o_j.In \ - \ o_i.In)$$

where

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

b_c is a normalization parameter determining the rate of asymptotic convergence of d_c to 1 as D_c goes to ∞

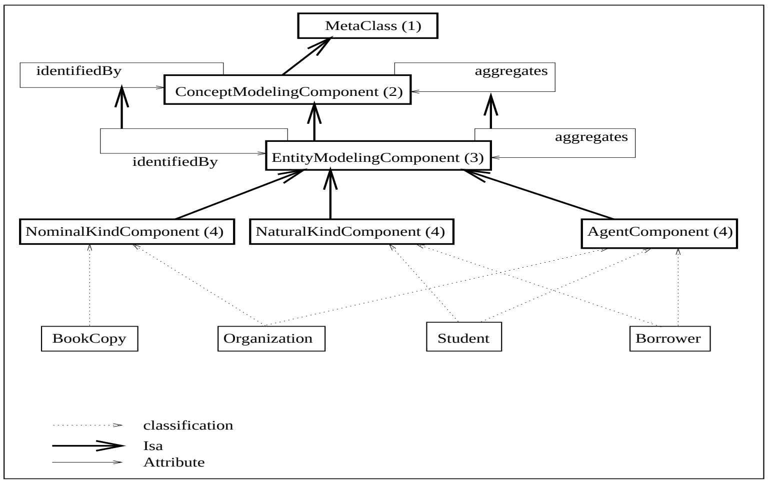

Figure 14: Classification of Entity Modeling Components

As an example, consider the estimation of the classification distances between the classes *Student*, *Borrower*, *Organization* and *BookCopy*, given their classification in Figure 14 (the numbers in parentheses are the specialization depths of the relevant classes). The absolute classification distance D_c between *Student* and *Borrower* equals 0 since both of them were classified as natural kind and agent modeling classes. The class *Organization*, although an agent modeling class itself, is a bit more distant from both *Student* and *Borrower* (their D_c distances are equal to 0.5 (0.5 = 1/4 + 1/4)) since it was classified as a nominal rather than a natural kind class. The class *BookCopy* is even more distant from *Student* and *Borrower* (their D_c distances are both equal to 0.75) since it was classified as a nominal and not as an agent class.

a3. Generalization Distance

The generalization distance provides a rough account of the semantic discrepancies of two objects as evidenced from their non-common superclasses. The generalization distance between individual classes is estimated in the same way as the classification distance except that their superclasses are taken into account. Unlike it, the generalization distance between attribute classes depends on the identity of their original classes. As discussed in (G. Spanoudakis 1994a), this distinction enables a differentiation between refinements of inherited attribute classes (i.e. specialization of their range classes) representing the same relations or properties, and generalizations between attribute classes with shared but non-identical semantics, both expressed by Isa relations in conceptual models.

Formally, the generalization distance is defined as:

Definition 2: The generalization distance, d_g, between objects o_i and o_j is defined as:

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

$$D_g(o_i, o_j) = \sum_{x \in NCS_{ij}} SD(x)^{-1}$$

$$NCS_{ij} = (o_i.Isa - o_j.Isa) \bigcup (o_j.Isa - o_i.Isa) \bigcup \{i, j\}$$

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

where

b_g is a parameter determining the rate of asymptotic convergence of d_g to 1 as D_g goes to ∞

As an example, consider the estimation of the absolute generalization distances D_g between the classes *Student*, *Borrower*, *Organization* and *BookCopy*, given their generalization taxonomy in Figure 15 (numbers in parentheses are the specialization depths of the relevant classes). This taxonomy distinguishes between resources and resource holder agents in specifications of resource borrowing systems. According to it, the classes *Student*, *Borrower* and *Organization* have pairwise absolute generalization distances equal to 0.5 (i.e. 1/4 + 1/4). The class *Librarian*, although a specialization of *ResourceHolderBorrower* itself, has a larger absolute generalization distance to all of them (i.e. 0.83 = 1/4 + 1/4 + 1/3) since it is also a subclass of *ResourceHolderEmployee*. Finally, the class *BookCopy* has an even larger absolute generalization distance to *Student*, *Borrower* and *Organization* (i.e. 1.91 = 1/4 + 1/3 + 1/2 + 1/2 + 1/3), since it expresses a special kind of resource rather than a resource holder agent.

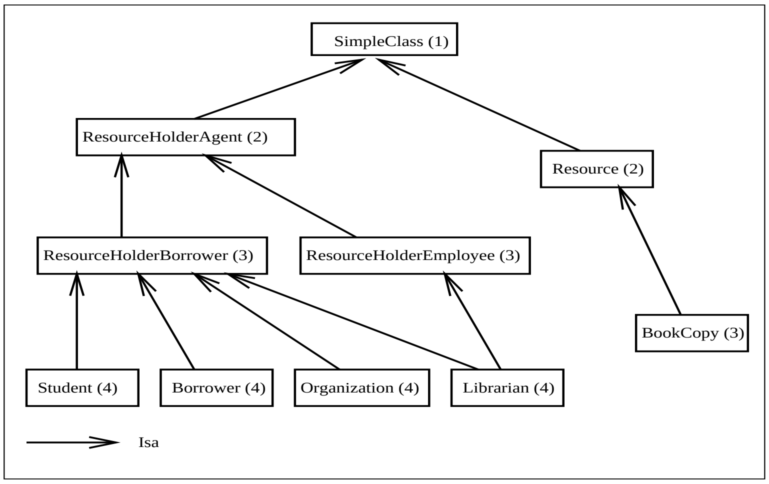

Figure 15: Generalization of Entity Modeling Components

a4. Attribution Distance

The attribution distance gives a detailed account of the analogous and unique attributes of two objects. It generates an isomorphism between their analogous attributes employing two criteria, namely the criteria of the *semantic homogeneity* and the *minimum distance isomorphism* (G. Spanoudakis 1994a).

The criterion of semantic homogeneity is employed for hypothesizing valid interattribute mappings in the first place. It precludes mappings between attributes, unless they are instances of the same original attribute classes. Such a common classification strongly indicates an analogy over the substance of objects (M. Turner 1988) and thus makes it likely that mapped attributes - despite of possible semantic discrepancies - express aspects of the same kind. Formally, two attribute objects o_i and o_j are semantically homogeneous if:

$$OCL[i] \equiv OCL[j] \;\; where \;\; OCL[x] = \{y | (y = OC(z)) \; and \; (z \in o_x.In)\}$$

The criterion of minimum distance isomorphism is employed for selecting a single among all the valid isomorphisms between the attributes of two objects, in particular the isomorphism, which has the minimum total distance. The total distance of an isomorphism is defined as the sum of the pairwise distances of the attributes mapped by it and the importance of the ones not mapped (cf. function D_a in definition 3). This criterion introduces a reasonable, objective notion of optimality in the selection process. The total distance of the optimal isomorphism between the attributes of two objects is defined as their attribution distance:

Definition 3: The attribution Distance, d_a, between objects o_i and o_j is defined as:

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

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

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

where

$I(o_i, o_j)$: is the set of all possible isomorphisms between the semantically homogeneous attributes of o_i and o_j

$o_i[m](o_j[m])$: 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 (G. Spanoudakis and P. Constantopoulos 1994b)

b_a is a normalization parameter determining the rate of asymptotic convergence of d_a to 1 as D_a goes to ∞

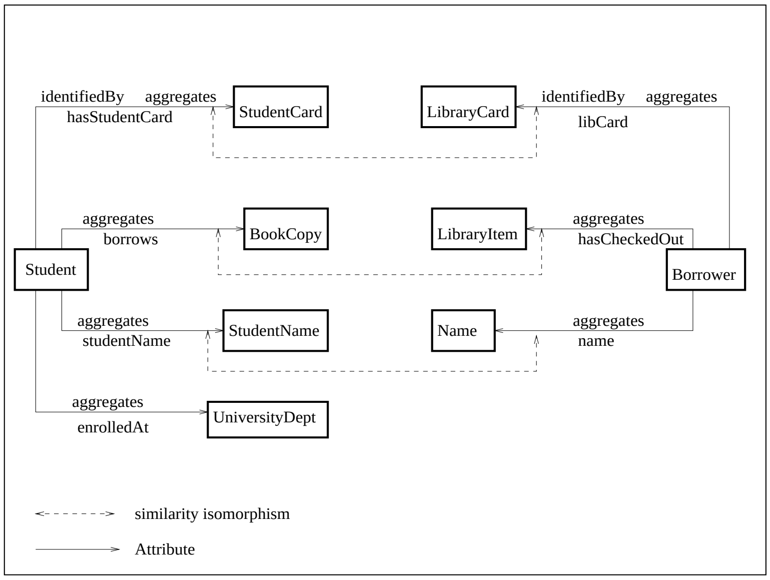

Figure 16: Optimal Isomorphism Between Student and Borrower Attributes

Notice that, since the attribution distance is recursively defined through the overall distance between the values of attributes (cf. definition 4 below), it generates optimal isomorphic mappings between the attributes in all the successive levels of the decompositions of two objects.

As an example, consider the estimation of the attribution distance between the classes *Student* and *Borrower* in Figure 16 (by convention the names above and below the lines denoting attributes are the logical names of their classes and themselves, respectively).

The attribution as well as the other partial and overall distances between the individual and attribute objects in Figures 16 and 17 are shown in table 1. The distances in this table were computed according to the conceptual descriptions of the relevant objects in the Figures 14,

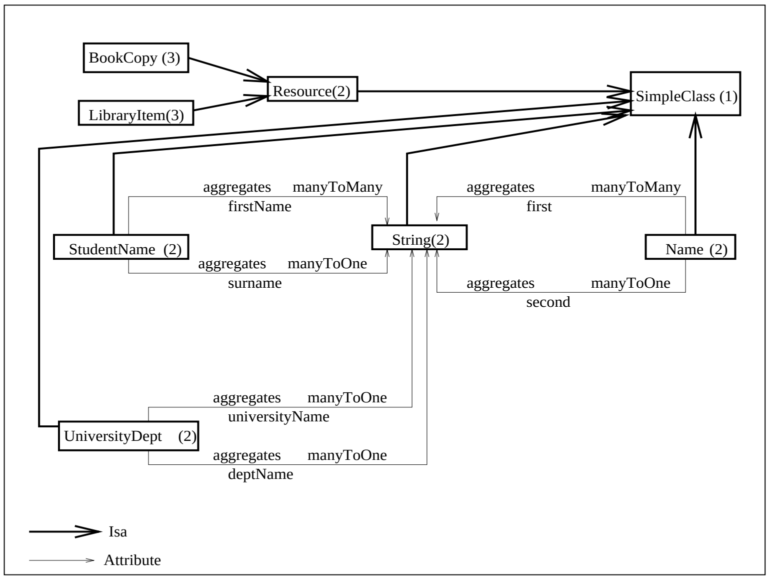

Figure 17: Modeling of Attribute Values in Figure 16

15, 16 and 17. Also, the classes *StudentCard*, *LibraryCard*, *LibraryItem*, *StudentName*, *Name*, *UniversityDept* and *String* had been all classified under the classes *EntityModelingComponent*, *ConceptModelingComponent* and *MetaClass* (cf. Figure 14). The salience measures for all the attribute objects (cf. s(i) measures in definition 3) were set equal to 1 in estimating the attribution distances of table 1 (normally the similarity model computes salience measures from the representation of attribute classes, as described in (G. Spanoudakis and P. Constantopoulos 1994b)).

According to the criterion of the semantic homogeneity, the attribute *hasStudentCard* of *Student* could be mapped only onto the attribute *libCard* of *Borrower*, since those attributes were both instances of the same original attribute meta-classes *identifiedBy* and *aggregates* of *ConceptModelingComponent* (cf. Figure 14). The same criterion allowed the mapping of any of the attributes *borrows*, *studentName* and *enrolledAt* of *Student* onto the attributes *hasCheckedOut* and *name* of *Borrower* as instances of the attribute meta-class *aggregates* of *ConceptModelingComponent*. From all the possible isomorphisms between these attributes, the criterion of the *minimum distance isomorphism* selected the one having mapped *borrows* onto *hasCheckedOut* and *studentName* onto *name* (cf. similarity isomorphism in Figure 16). As shown in table 1, these attribute pairs have lower overall pairwise distances (cf. column d in table 1) than the other possible pairs of attributes of *Student* and *Borrower* due to the lower overall distances of their object-values (cf. column $d(to\ values)$ in table 1). Notice that the classification distances between all these pairs were equal to 0 (cf. column d_c) since the relevant attributes were in-

Table 1: Partial and Overall Distances of Objects in Figures 16, 17

Objects	d_{id}	$d_c(b_c = 1.5)$	$d_g(b_g = 1.5)$	$d_a(b_a = 1.5)$	$d(to\ values)$	$d(b = 1)$
StudentCard, LibraryCard	1	0	0.6	1	-	0.632
BookCopy, LibraryItem	1	0	0.49	1	-	0.622
BookCopy, Name	1	0	0.55	1	-	0.626
StudentName, LibraryItem	1	0	0.66	1	-	0.636
StudentName, Name	1	0	0.6	0.636	-	0.594
UniversityDept, LibraryItem	1	0	0.66	1	-	0.636
UniversityDept, Name	1	0	0.6	0.652	-	0.595
String, String	0	0	0	0	-	0
firstName, first	1	0	1	1	0	0.876
surname, second	1	0	1	1	0	0.876
universityName, second	1	0	1	1	0	0.876
deptName, second	1	0	1	1	0	0.876
hasStudentCard, libCard	1	0	1	1	0.632	0.87652
borrows, hasCheckedOut	1	0	1	1	0.622	0.87651
borrows, name	1	0	1	1	0.626	0.87652
studentName, hasCheckedOut	1	0	1	1	0.636	0.87653
studentName, name	1	0	1	1	0.594	0.87647
enrolledAt, hasCheckedOut	1	0	1	1	0.636	0.87653
enrolledAt, name	1	0	1	1	0.595	0.87648
Student, Borrower	1	0	0.42	0.7839	-	0.592

stances of the same class (i.e. *aggregates*). Also, their generalization distances were all equal to 1 (cf. column d_g) since they did not share the same original attribute classes (the original class of each of them was itself, according to their modeling in Figure 16).

As far as the object-values of these attribute classes are concerned, the pairs (*BookCopy*, *LibraryItem*) and (*StudentName*, *Name*) had the lower overall distances. This was because of the common superclass *Resource* of the objects in the former pair which resulted into a lower generalization distance between them (i.e. 0.49). Also, the mapping between all the attributes of the objects in the latter pair resulted into a relatively lower attribution distance between them (i.e. 0.636). Notice, that the attribution distance between *UniversityDept* and *Name* (i.e. the object-values of the pair of attributes *enrolledAt* and *name*) was larger than the relevant distance between *StudentName* and *Name*(0.652 vs. 0.636) because only one of the attributes *universityName*, *deptName* of *UniversityDept* could be mapped onto the attribute *second* of *Name* due to the criterion of the semantic homogeneity (the attribute *first* could not be mapped on any of them since it had been classified as a *manyToMany* attribute, as shown in Figure 17).

a5. Overall Object Distance

Overall distances are estimated by aggregating classification, generalization and attribution distances between objects, using a quadratic functional form prompted by empirical findings of statistically significant correlations between d_c, d_g and d_a (G. Spanoudakis 1994a) and formally defined as:

Definition 4: The overall distance, d, between objects o_i and o_j is defined as:

$$d(o_i, o_j) = \frac{bD(o_i, o_j)}{bD(o_i, o_j) + 1}$$

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

where

d_{id} is the identification distance of objects o_i and o_j, defined as:

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

b is a normalization parameter determining the rate of asymptotic convergence of d to 1 as D goes to ∞

Table 1 presents the overall distances between some of the entity and attribute objects of Figures 16 and 17 as estimated from their partial distances according to definition 4. For example, the absolute overall distance (D) between the entity objects *Student* and *Borrower* was estimated as $(1^2 + 0^2 + 0.42^2 + 0.7839^2 + 0 * 0.42 + 0 * 0.7839 + 0.42 * 0.7839)^{1/2} \ = \ 1.45606$. The same distance between the attribute objects *hasStudentCard* and *libCard* was estimated as $(1^2 + 0^2 + 36 * 1^2 + 1^2 + 0.632^2 + 12 * 0 * 1^2 + 0 * 1^2 + 12 * 1 * 1)^{1/2} \ = \ 7.0988$. Notice that since these attribute classes had no attributes of their own their attribution distance was 1, according to definition 3.

References

S. Al-Fedaghi and P. Scheuermann. 1981. Mapping Considerations in the Design of Schemas for the Relational Model, IEEE Transactions on Software Engineering, 7(1), 1981

Y. Arens and C. Knoblock. 1992. Planning and Reformulating Queries for Semantically-Modeled Multidatabase Systems, In Proceedings of the 1st International Conference on Information and Knowledge Management, 1992

C. Batini and M. Lenzerini and S. Navathe. 1986. A Comparative Analysis of Methodologies for Database Schema Integration, ACM Computing Surveys, 18(4), 1986

M. Bright and A. Hurson and S. Pakzad. 1994. Automated Resolution of Semantic Heterogeneity in Multidatabases, ACM Transactions on Database Systems, 19(2), June 1994

M. Casanova and M. Vidal. 1983. Towards a Sound View Integration Methodology, In Proceedings of the 2nd ACM SIGART/SIGMOD Conference on Principles of Database Systems, ACM, New York

P. Constantopoulos et al. 1993. Repositories for Software Reuse: The Software Information Base, In Proceedings of the IFIP Conference on the Software Development Process, Como, Italy, 1993

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

F. Codd. 1979. Extending the Database Relational Model to Capture More Meaning, ACM Transactions on Database Systems, 4(4), December 1979

C. Collet et al. 1991. Resource Integration Using a Large Knowledge Base in Carnot, IEEE Computer, 24(12), December 1992

S. Easterbrook and B. Nuseibeh. 1995. Managing Inconsistencies in an Evolving Specification, In Proceedings of the IEEE International Conference on Requirements Engineering, York, England, March 1995

B. Falkenhainer et al. 1990. The Structure Mapping Engine: Algorithm and Examples, Artificial Intelligence, 41, 1990

A. Finkelstein et al. 1994. Inconsistency Handling in Multi-Perspective Specifications, IEEE Transactions on Software Engineering, 20(8), August 1994

G. Furnas et al. 1987. The Vocabulary Problem in Human-System Communication, Communications of the ACM, 30(11), 1987

D. Gangopadhyay and T. Barsalou. 1991. On the Semantic Equivalence of Heterogeneous Populations in Multimodel, Multidatabase Systems, SIGMOD Record 20(4), 1991

C. Goh and S. Madnick and M. Siegel. 1994. Context Interchange: Overcoming the Challenges of Large-Scale Interoperable Database Systems in a Dynamic Environment, In Proceedings of the 3rd International Conference on Information and Knowledge Management, Gaithersurg, Maryland, 1994

T. Gruber and G. Olsen. 1994. An Ontology for Engineering Mathematics, In Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning, Bonn, Germany, 1994

C. Heitmeyer et al. 1995. Consistency Checking of SCR-Style Requirements Specifications, In Proceedings of the IEEE International Conference on Requirements Engineering, York, England, March 1995

R. Hull and R. King. 1987. Semantic Database Modeling: Survey, Applications and Research Issues, ACM Computing Surveys, 19(3), 1987

P. Johanneson. 1993. Schema Transformations as an Aid in View Integration, In Proceedings of CAiSE '93, LNCS 685, June 1993

G. Kotonya and I. Sommerville. 1992. Viewpoints for Requirements Definition, Software Engineering Journal, 7(6), 1992

J. Larson and S. Navathe and R. Elmasri. 1989. A Theory of Attribute Equivalence in Databases with Application to Schema Integration, IEEE Transactions on Software Engineering, 15(4), April 1989

J. Leite and P. Freeman. 1991. Requirements Validation Through Viewpoint Resolution, IEEE Transactions on Software Engineering 17(12), 1991

D. Lenat and R. Guha. 1990. Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project, Addison-Wesley, Reading, Mass., 1990

W. Litwin and A. Abdellatif. 1987. An Overview of the Multi-Database Manipulation Language MDSL, In Proceedings of the IEEE, 75(4), 1987

N. Maiden et al. 1994. Distributed Requirements Engineering within NATURE, ESPRIT BRA

NATURE, Nature Report Series, 1994

N. Maiden et al. 1995. Computational Mechanisms for Distributed Requirements Engineering, In Proceedings of the 7th International Conference on Software Engineering & Knowledge Engineering (SEKE '95), USA, June 1995

M. Mannino and W. Effelsberg. 1984. Matching Techniques in Global Schema Design, In Proceedings of the IEEE COMPDEC Conference, Los Angeles, California, 1984

S. Meyers and S. Reiss. 1991. A System for Multiparadigm Development of Software Systems, In Proceedings of the 6th International Workshop on Software Specification and Design(IWSSD-6), Como, Italy, October 1991

N. Maiden and A. Sutcliffe. 1994. Requirements Critiquing Using Domain Abstractions, In Proceedings of the IEEE Conference on Requirements Engineering, 1994

D. Monarchi and G. Puhr. 1992. A Research Typology for Object Oriented Analysis and Design, Communications of the ACM, 35(9), 1992

A. Motro and P. Buneman. 1981. Constructing Superviews, In Proceedings of the International Conference on Management of Data, ACM, New York, 1981

P. Motschnig-Pitrik. 1993. The Semantics of Parts vs. Aggregates in Data Knowledge Modeling, In Proceedings of CAiSE '93, LNCS 685, Paris, France, June 1993

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

S. Navathe and S. Gadgil. 1982. A Methodology for View Integration in Logical Data Base Design, In Proceedings of the 8th International Conferences on Very Large Data Bases, VLDB Endowment, Saratoga, California, 1982

B. Nuseibeh et al. 1993. Expressing the Relationship between Multiple Views in Requirements Specification, Proceedings of 15th International Conference on Software Engineering, 1993

J. Peckham and F. Maryanski. 1988. Semantic Data Models, ACM Computing Surveys, 20(3), 1988

C. Papadimitriou and K. Steiglitz. 1982. Combinatorial Optimization: Algorithms and Complexity, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982

W. Robinson and S. Fickas. 1994. Supporting Multi-Perspective Requirements Engineering, In

Proceedings of the IEEE Conference on Requirements Engineering, 1994

E. Sciore and M. Siegel and A. Rosenthal. 1994. Using Semantic Values to Facilitate the Interoperability Among Heterogeneous Information Systems, ACM Transactions on Database Systems, 19(2), June 1994

A. Sheth and J. Larson. 1990. Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases, ACM Computing Surveys, 22(3), September 1990

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

G. Spanoudakis and P. Constantopoulos. 1994a. Similarity for Analogical Software Reuse: A Computational Model, In Proceedings of the 11th European Conference on Artificial Intelligence(ECAI '94), Amsterdam, The Netherlands, August 1994

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

G. Spanoudakis. 1994a. Analogical Similarity of Objects: A Conceptual Modeling Approach, Ph.D Dissertation, Department of Computer Science, University of Crete, Heraklion, September 1994
(available from *http://web.cs.city.ac.uk/homes/gespan/info.html*)

G. Spanoudakis. 1994b. Similarity Analyser: An Implementation Overview, Working Paper 12, Institute of Computer Science, Foundation for Research and Technology-Hellas, Heraklion, Crete, Greece, September 1994
(available from *http: //web.cs.city.ac.uk/homes/gespan/info.html*)

V. Storey. 1993. Understanding Semantic Relations, VLDB Journal 3, 1993

M. Turner. 1988. Categories and Analogies, Analogical Reasoning: Perspectives of Artificial Intelligence, Cognitive Science and Philosophy, (ed) Helman D.H., Kluwer Academic Pub., Dordrecht, The Netherlands, 1988

S. Yao and V. Waddle and B. Bousel. 1982. View Modeling and Integration Using the Functional Data Model. IEEE Transactions on Software Engineering, 8(6), 1982

P. Zave and M. Jackson. 1993. Conjunction as Composition, ACM Transactions on Software Engineering and Methodology, 2(4), October 1993