Reconciliation: Managing Interference in Software Development

George Spanoudakis[1] and **Anthony Finkelstein**[1]

Abstract. This paper outlines a method, called reconciliation, for managing interference between partial specifications or viewpoints. The method supports the detection, verification and tracking of ontological overlaps. The paper describes the heuristics on which the method is based and illustrates the application of the method to a scenario.

1 INTRODUCTION

The construction of a complex software system involves many agents (*aka* participants or actors). These agents have different perspectives or views of the artifact or system they are trying to describe or model. This gives rise to many partial specifications (or viewpoints) reflecting those perspectives [18, 21]. These specifications "interfere" with each other to the extent they refer to, or assert properties of common aspects of the system under development and its domain. This is a particular feature of the requirements engineering setting.

Interference between specifications occurs at various levels. First, they might "ontologically overlap", that is they might incorporate components referring to common aspects of the "real world". Second, in cases where they overlap ontologically, they might also be inconsistent with each other. We believe that both ontological overlap and inconsistency are inevitable and acceptable in system development [8]. Ontological overlap because it is necessary to support multiple perspectives, and inconsistency because it is a necessary to support innovative thinking, deferment of commitments and exploration of alternatives.

The consequence of this stance is that interference between specifications needs to be "managed", involving cooperation between the "viewpoint owners" [8]. This is complicated by specifications which: use different languages; are at different degrees of abstraction, granularity and formality; deploy different terminologies; are at different stages of development or elaboration. These complexities, set alongside the normal software engineering problems of scale, suggest the need for automated reasoning and method support for interference management.

The interference management that is required (or indeed possible) varies. In certain cases it might be loose, that is simply identifying ontological overlaps and inconsistencies, in other cases it might be tight, involving the integration of the specifications and the resolution of their inconsistencies.

Reconciliation, the method discussed in this paper lies between loose and tight interference management. It supports the detection, verification and tracking of ontological overlaps. This is in some sense an easier problem than dealing with inconsistencies. It is amenable to the application of heuristic techniques, for example, inexact-matching mechanisms [26] and models of computer-supported negotiation [6], while inconsistency detection may require theorem proving and sophisticated formal frameworks [12]. In any case the detection of ontological overlaps is prerequisite for detecting inconsistencies.

In the rest of the paper[2], we describe a heuristic method for reconciling viewpoints (§2), we detail the heuristics on which it is based (§3), give a scenario of using it (§4), briefly describe tool support for specification matching (§5), review related work (§6), and conclude with a discussion of open research issues and future work (§7). An Appendix giving basic definitions is attached.

2 OVERVIEW

The method we adopt for reconciling viewpoints has two basic stages — analysis and revision. It detects ontological overlaps using a computational model of similarity and a classification of specification components with respect to a meta-model of domain-independent, semantic modelling properties — analysis. It also supports the remodelling of specification components so that the results of similarity analysis and viewpoint owners assessment of overlaps converge — revision. The goal of this process is to ensure that the modelling of specifications is consistent with the human assessment of ontological overlaps between them and establish a shared understanding among viewpoint owners of the potential for inconsistency.

2.1 Analysis of Specifications

The analysis of specifications is performed by a computational model of similarity. Specifications are treated as aggregations of "specification components", classified using a meta-model, which expresses general, domain-independent, semantic modelling properties. Both this meta-model and the specifications are described in Telos, an object-oriented knowledge representation language, supporting the semantic modelling abstractions of classification, generalisation and attribution [20].

2.1.1 Meta-model

The meta-model consists of a kernel and set of extensions. The kernel is organised as a generalisation taxonomy of classes, with specification components classified by the properties they possess. In particular, components are distinguished into those representing

[1] Department of Computer Science, City University, Northampton Square, London EC1V 0HB, UK, e-mail: {gespan | acwf @cs.city.ac.uk}

ECAI 96, Workshop on Modelling Conflict in AI

[2] a more detailed account of this paper has been submitted to the Annals of Software Engineering

entities and those representing relations. Entity representing components are further specialised into natural, nominal, place, event, activity, state, agent and physical quantity components. Components representing relations are initially distinguished by their arity (for example binary or n-ary relations). Binary relations are further specialised according to: cardinality constraints (for example 1:1, N:M, total, and onto relations); mathematical properties (for example symmetric, transitive and set-inclusion relations); existential dependencies between related items or other constraints between them, including their temporal coexistence, physical separability and substance homogeneity [27].

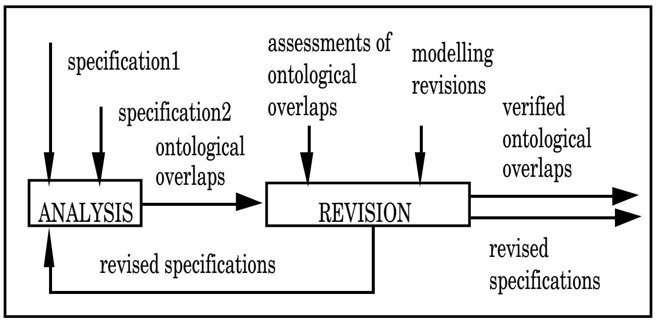

Figure 1. Reconciliation of Viewpoints

The extensions to the meta-model comprise classes of additional properties for modelling established specification languages. Current extensions include constructs to support the description of specifications developed in the relational (for example fields, relations, inclusion dependencies) and object-oriented data models (for example object types, attributes and identifiers, is-a relations).

In essence, the meta-model enables the enrichment of the semantic content of specification components by asserting domain-independent properties about them and supports their representation with respect to a common set of structuring constructs. Both are prerequisites for the computational detection of their similarity. A detailed description of the meta-model is given in [26].

2.1.2 Computational Model of Similarity

Specifications, described as Telos objects in terms of the meta-model, are compared using a computational model of similarity [25, 26]. Similarity analysis is based on three metric functions, namely the classification, generalisation and attribution metrics, which measure conceptual distances between specifications with respect to the classification, generalisation relations and the attributes constituting their descriptions.

The classification distance between specification components indicates their differences with 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 importance measures obtained into a classification distance measure (function d_c in the Appendix). The generalisation distance reveals semantic differences between specification components, indicated by their non-common superclasses and is computed in a similar manner to the classification distance (function d_g in the Appendix). The attribution distance between specifications determines an optimal isomorphism I_s between the structures of two specifications. Specification components contained in these structures are mapped only if they are classified under the same classes of the meta-model — *semantic homogeneity*. In cases where they can be mapped in many ways, the model selects the mapping with the minimum total distance (*minimum distance isomorphism* and function d_a in the Appendix). The estimation of the pairwise distances between specification components uses recursive generation of isomorphic mappings between their own substructures.

The similarity analysis of two specifications results in: (i) their classification, generalisation, attribution and overall distance measures; (ii) a graph mapping semantically homogeneous components at the successive levels of the structural closures of two specifications (the arcs of this graph are weighted by the pairwise distances of the mapped components); (iii) lists with their common and non common classes and superclasses each weighted by its importance.

2.2 Revision of Specifications

The isomorphism I_s between the components of two specifications is likely to reflect their ontological overlaps. However, flaws, incompleteness or lack of an adequate semantics in the modelling of specifications might force similarity analysis to generate mappings, which are incorrect. In such cases, viewpoint owners can propose a different isomorphism I_o between specification components, which in their opinion correctly reflects these overlaps.

Our method, based on a taxonomy of discrepancies between I_s and I_o, suggests heuristic checks on, and subsequently revisions to, specifications, which would make I_s and I_o converge. The criteria forced similarity analysis to generate incorrect mappings between components, namely the *semantic homogeneity* and the *minimum distance isomorphism*, can be used to trace those mappings back to specific elements in specifications modelling and suggest revisions. Some of these revisions resolve forms of inconsistency between specifications arising from incompatible classifications of ontologically coincident components under the meta-model. Others complete specifications with respect to each other. The method supports iterative revisions (Figure 1) up to a point where I_s and I_o coincide completely. Revision stops at this point since the results of similarity analysis are confirmed as a correct reflection of the ontological overlaps between the specifications. Along the way, the method guides viewpoint owners through a disciplined check on the correctness and completeness of their specifications as well as systematic modification in line with their indication of the ontological overlaps between them.

3 HEURISTICS

3.1 Appraisal

Viewpoint owners "appraise" the result of similarity analysis by suggesting an alternative isomorphism I_o between the ontologically coincident components of their specifications. In general, I_s and I_o will partially coincide. Components with and without counterparts in I_s will be referred to as *corresponding* and *unique*, respectively.

Components mapped identically or left without any counterparts by both I_s and I_o will be said to be correct corresponding and correct unique components, respectively. Components with non identical mappings in I_s and I_o will be said to be wrong unique or wrong corresponding components. The characterization "wrong" for corresponding components means that their mappings by I_s do not correctly indicate ontological overlaps. Similarly, the characterization "wrong" for unique components in some specification means that, despite the absence of any counterparts for them in I_s, they should be treated as ontologically coincident with components of the other specification with which it is being compared. We can further distinguish wrong components as:

(i) Wrong unique components of type 1 (WU1-components): These are unique components of one specification, which should have been mapped onto unique components of the other (according to I_o) although they have not been by I_s (e.g. components x1 and y1 in Case 1 of Figure 2).

(ii) Wrong unique components of type 2 (WU2-components): These are unique components of one specification that should have been mapped onto components of the other (according to I_o), which have been mapped onto different counterparts by I_s (e.g. component x1 in Case 2 of Figure 2).

(iii) Wrong unique components of type 3 (WU3-components): These are unique components of one specification that should have been mapped onto components of the other (according to I_o), which did not exist at the time of comparison (e.g. component x1 in Case 3 of Figure 2).

(iv) Wrong unique components of type 4 (WU4-components): These are unique components of one specification which are identified as redundant after comparison with another specification fails to map them onto counterparts (e.g. component x1 in Case 4 of Figure 2).

(v) Wrong corresponding components of type 1 (WC1-components): These are components which should have been mapped (according to I_o) on counterparts unique in I_s and different from the ones on to which I_s maps them (e.g. component y1 in Case 2 of Figure 2).

(vi) Wrong corresponding components of type 2 (WC2-components): These are components, which should not have been mapped onto any counterparts according to I_o although they have by I_s (e.g. components x1 and y1 in Case 5 of Figure 2).

(vii) Wrong corresponding components of type 3 (WC3-components): These are components, which should have been mapped (according to I_o) onto counterparts different from the ones they have in I_s. Their appropriate counterparts have themselves been mapped onto wrong components by I_s (e.g. components x1 and y2 in Case 6 of Figure 2).

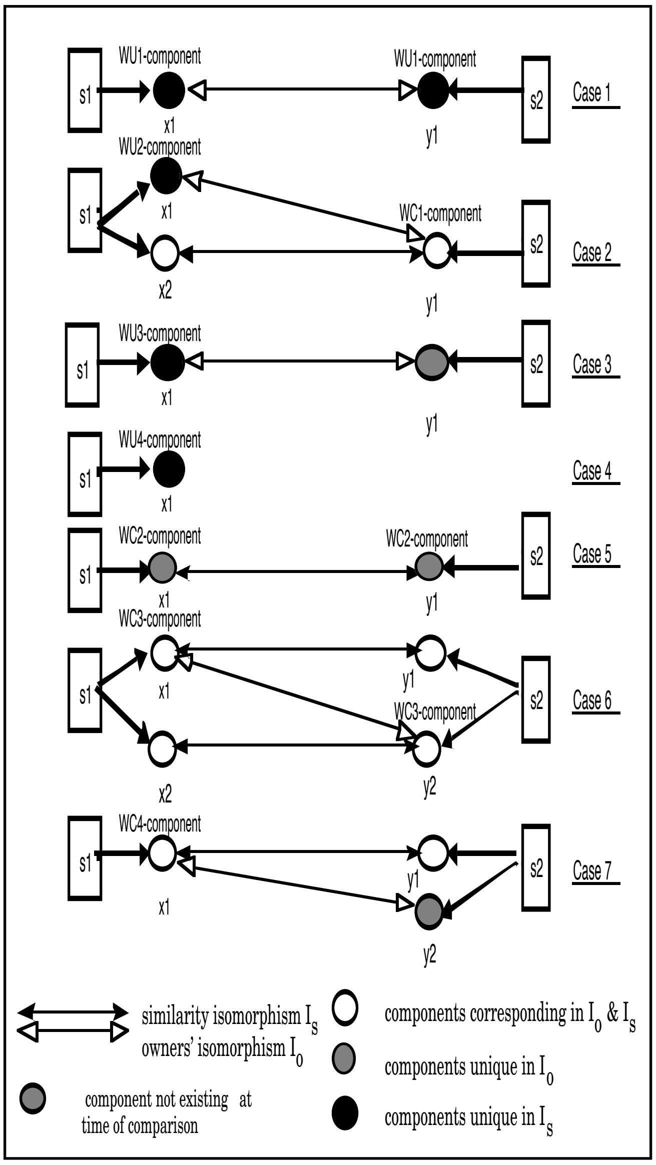

Figure 2. Wrong Unique and Corresponding Specification Components

(ix) Wrong corresponding components of type 4 (WC4-components): These are components, that should have been mapped (according to Io) onto counterparts different from the ones they have in I_s, which did not exist at the time of comparison (e.g. component x1 in Case 7 of Figure 2).

Based on these distinctions, our method provides a set of heuristics that can be deployed for tracing disparities between I_o and I_s back to the modelling of specification components and revising them so that I_o and I_s converge. These heuristics are discussed below.

3.2 Dealing with WU1-Components

WU1-components may appear if similarity analysis prevents them being mapped as they are not classified under exactly the same classes of the meta-model (due to *semantic homogeneity*). Notice that, the classification of ontologically overlapping components should be identical because they are expected to share the same general semantic properties. For instance, it would not be reasonable to say that two components, classified as 1:N and M:N relations respectively, express the same relationship in the real world. Therefore, classification discrepancies between ontologically overlapping components might be reasonably attributed to incorrect and/or incomplete classification with respect to the meta-model. Consequently, they need to be checked and possibly revised thereby enabling similarity analysis to generate the desired mappings. The following heuristics (expressed for the WU1-components of Case 1 in Figure 2) may be applied in such cases:

H1: *Check if the non-common classes of x1 and y1 are correct and if not remove them.*

H2: *Check if any of the non-common classes of x1 and y1 should be classes of the other as well and add the relevant classifications.*

3.3 Dealing with WU2-Components

WU2-components may appear due to either the semantic homogeneity or the minimum distance isomorphism in similarity analysis. Because of the semantic homogeneity criterion, a WU2-component may not be mapped onto its desired counterpart as they have non identical classifications. Such cases may be explored and revised in accordance with heuristics H1 and H2, as in the case of WU1-components. In other cases, the mapping of the desired counterpart of a WU2-component onto a different, but wrong, component of the same specification (component x2 of Case 2 in Figure 2) might be the result of an accidental incorrect common classification of them. Such cases can be explored and rectified using the following heuristics (expressed for the components of Case 2 in Figure 2):

H3: *Check if the common classes of x2 and y1 are correct and if not remove them.*

H4: *Check if x2 and y1 should have been classified under any non-common classes of the meta-model, although they have not, and if so add the relevant classifications.*

Notice that, H4 supports the elicitation of new information about the components involved.

If the classification checks do not resolve the problem, the desired mapping might be achieved by exploring why $d(x2,y1)$ is less than $d(x1,y1)$ and revising the modelling of x1 and y1 in order to remedy this inequality. Viewpoint owners need to consider specific aspects in the modelling of x1 and y1, which affected the partial conceptual distances between them and consequently their overall distance. Given that H1 and H2 have been applied revealing no classification discrepancies between the components (this implies that their classification distance equals 0), the overall distance inequality might be the result of similar inequalities between the generalisation and/or the attribution distances of the involved components (i.e. $d_g(x1,y1) > d_g(x2,y1)$ and/or $d_a(x1,y1) > d_a(x2,y1)$). Reversing any of these inequalities by revising the modelling of x1 and y1 can force similarity analysis map them onto each other. Below, we present heuristics guiding such revisions, (expressed for the components of Case 2 in Figure 2).

3.3.1 Reverse the inequality between the generalisation distances

Viewpoint owners may consider revising the generalisations of x1 to decrease $d_g(x1,y1)$ and the generalisations of y1 to decrease $d_g(x1,y1)$ or increase $d_g(x2,y1)$ or both.

i) decrease $d_g(x1,y1)$. The following heuristics might be applied:

H5: *Check if x1 has been incorrectly generalised to its unique superclasses with respect to y1 and if so remove the relevant generalisations.*

H6: *Check if any of the unique superclasses of x1 with respect to y1, which are not superclasses of x2, should be superclasses of y1 and if so add the relevant generalisations.*

Notice that, adding to the superclasses of y1 the unique superclasses of x1 which are superclasses of x2, would not affect the inequality $d_g(x1,y1) > d_g(x2,y1)$ since it would decrease equally both $d_g(x1,y1)$ and $d_g(x2,y1)$.

H7: *Check if y1 has been incorrectly generalised to its unique superclasses with respect to x1 but not x2 and remove the relevant generalisations if so.*

Notice that, removing the unique superclasses of y1 with respect to both x1 and x2 would not affect the inequality $d_g(x1,y1) > d_g(x2,y1)$ since it would decrease equally both $d_g(x1,y1)$ and $d_g(x2,y1)$.

ii) increase $d_g(x2,y1)$. The following heuristic might be applied:

H8: *Check if y1 has been incorrectly generalised into its common superclasses with x2, which are not superclasses of x1, and if so remove the relevant generalisations.*

Each non common superclass increases the generalisation distances between components by the inverse of its specialisation depth in the generalisation taxonomies of which it is a part. Therefore, it is possible to present viewpoint owners with the relevant sets of classes, which are to be added or deleted, ordered in ascending specialisation depths and identify those whose atomic modification could reverse the inequality $d_g(x1,y1) > d_g(x2,y1)$.

3.3.2 Reverse the inequality between the attribution Distances

The attribution distances between x1 and y1 and between x2 and y1 are computed from optimal isomorphisms mapping the subcomponents of y1 onto the subcomponents of x1 and x2, respectively. Revising the modelling of the subcomponents of x1 and y1 may change these isomorphisms either decreasing $d_a(x1,y1)$ or increasing $d_a(x2,y1)$. This can be done by applying the following heuristics:

i) decrease $d_a(x1,y1)$

H9: *Check if the unique subcomponents of y1, with respect to the similarity isomorphism between x1 and y1, are WU1, WU2, WU3 or WU4 components and deal with them if so.*

The subcomponents of y1, which confront to H9 can be checked in an order imposed by the following measure:

$$s(z)\ (\ d(z, I_s(z)) - d(z, I_s'(z))$$

In this formula, z refers to a unique subcomponent of y1; *s(z)* is a measure of the importance of z for y1, called *salience*, which is computed by the similarity analysis model; $I_s(z)$ and $d(z,I_s(z))$ refer to the counterpart of z in x1 (i.e. nil) and the overall distance between $I_s(z)$ and z (i.e. 1), respectively; and $I_s'(z)$ and $d(z,Is'(z))$ refer to the counterpart of z in x2 and the overall distance between Is'(z) and z (i.e. 1) respectively. Hence, the subcomponents of y1 subject to H9 should be considered in a sequence determined by their distance to their counterparts in x2, weighted by their salience for y1.

H10: *Check if the unique subcomponents of x1 with respect to the similarity isomorphism between x1 and y1 t are WU1, WU2, WU3 or WU4 components and deal with them if so.*

The subcomponents of x1, which are subject to H10 can be checked in an order imposed by their salience *s(z)*, since this salience measure determines their contribution to the overall distance between x1 and y1 (function d_a in the Appendix).

H11: *Check for WC1 or WC3 subcomponents in the similarity isomorphism between x1 and y1 and deal with them if any.*

Notice that, checking and possibly revising WC2 and WC4 components in x1 and y1 would not decrease $d_a(x1,y1)$, since re-mapping them would give rise to two or one unique components in the specifications, respectively.

ii) increase $d_a(x2,y1)$

H12: *Check for WC2 or WC4 subcomponents with respect to the similarity isomorphism between y1 and x2 and deal with them if any.*

WC2 components would increase the attribution distance between y1 and x2 if left unmapped, since according to function d_a they would contribute to the maximum extent to this distance.

3.4 Dealing with WU3-Components

H13: *Create a new component y1 with no subcomponents and deal with y1 and x1 as WU1 components.*

H13 supports the elicitation of new components in one of the participating specifications.

H14: *Check for WU1, WU2, WU3 or WU4 components in the subcomponents of x1 and deal with them if any.*

3.5 Dealing with WU4-Components

Having been identified as redundant, WC4 components should be removed from their aggregating specifications. Hence

H15: *Remove x1.*

3.6 Dealing with WC1-Components

WC1 components as counterparts of WU2 components in I_s appear for same the same reasons and therefore the heuristics introduced in section 3.3 can be applied.

3.7 Dealing with WC2-Components

Since the contribution of unique components to the overall distance between specifications is maximal, similarity analysis prefers to map them onto dissimilar counterparts rather than leaving them unmapped, provided that the criterion of semantic homogeneity allows it. By doing this, it minimizes the overall distance between the specifications being compared. In such circumstances it is possible that an incorrect common classification of two components might have caused an undesired mapping, which could be avoided by modifying the classification of the components. This case might be subject to the heuristics H3 and H4.

3.8 Dealing with WC3-Components

The WC3-components x1 and y2 of Case 6 in Figure 2 might not have been mapped onto each other as required either because they were not identically classified or because the mapping of x1 onto y1 and x2 onto y2 had a relatively lower aggregate distance, i.e. $d(x1,y1) + d(x2,y2) < d(x1,y2) + d(x2,y1)$. In this circumstance we can apply the following heuristic:

H16: *Regard x1 as a WU2-component that should be mapped onto y2, and y2 as a WU2-component that should be mapped onto x1, and deal with them*

H16 leads to the application of the heuristics concerning the classification of components and the inequalities between their generalisation and attribution distances, discussed above.

3.9 Dealing with WC4-Components

H17: *Create a new component y2 with no subcomponents and consider it as WU2-component.*

4 SCENARIO

In this scenario we demonstrate the application of our method using as an example two object-oriented specifications of library borrowers, items and their relations. These specifications overlap by including components representing library borrowers (the object types *Borrower* and *Student*), different types of library items (the object types *CopyOfBook*, *Publication* and their subtypes) and borrowing relations between them (the object type *Loans* and the object attribute *Borrows*). However, they are not modelled identically (different taxonomies of library items, different attributes for students and borrowers). Similarity analysis generates the isomorphism I_s shown in Figure 3, driven by the identical classification and the structural similarities of the components. In particular, I_s maps:

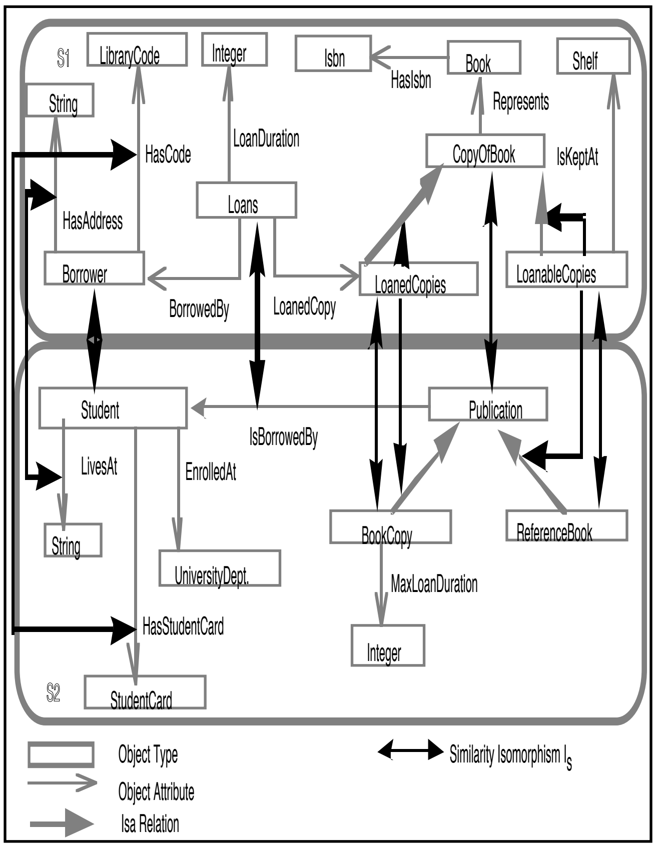

Figure 3. Scenario Step 1

(1) *Borrower* onto *Student*, as the most similar pair of natural kind, agent modelling entities

(2) *CopyOfBook* onto *Publication*, as the most similar pair of natural kind, entities

(3) *LoanedCopies* onto *BookCopy*, as the most similar pair of natural kind, entities

(4) *LoanableCopies* onto *ReferenceBook*, as the most similar pair of natural kind, entities

(5) *HasCode* onto *HasStudentCard*, as the most similar pair of 1:1, total, onto, contemporaneous, non homogeneous, separable and existentially independent binary relations

(6) *HasAddress* onto *LivesAt*, as the most similar pair of N:M, optional, onto, contemporaneous, non homogeneous, separable and existentially independent binary relations

(7) *Loans* onto *IsBorrowedBy*, as the most similar pair of N:1, optional, not onto, contemporaneous, non homogeneous, separable and existentially independent binary relations

The reconciliation of S1 and S2 starts from the assessment of I_s by viewpoint owners. Assuming that the mappings of *Borrower* onto *Student*, *HasCode* onto *HasStudentCard*, *HasAddress* onto *LivesAt* and *Loans* onto *IsBorrowedBy* are verified by viewpoint owners as correctly reflecting ontological overlaps, we concentrate on the other associations in I_s. According to viewpoint owners, the mappings of *CopyOfBook* onto *Publication*, *LoanedCopies* onto *BookCopy* and *LoanableCopies* onto *ReferenceBook* do not reflect correct ontological overlaps.

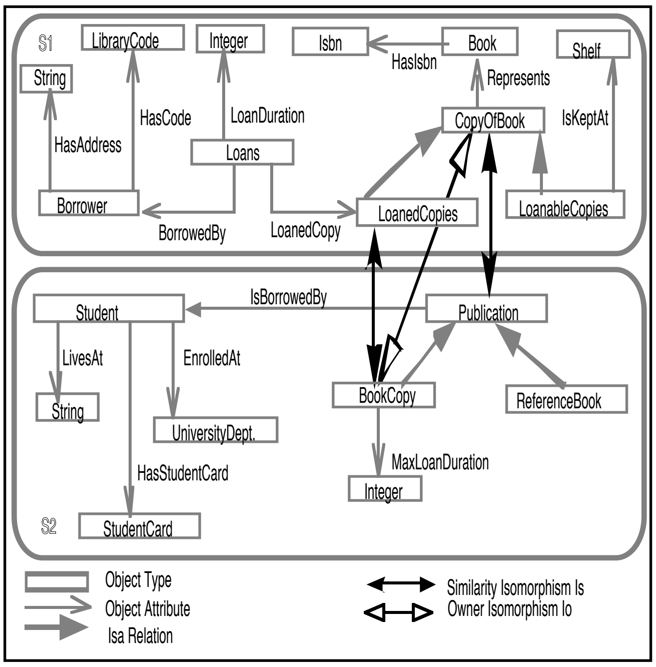

Figure 4. Scenario Step 2

In particular, *CopyOfBook* should have been mapped onto *BookCopy* rather than *Publication*, as indicated by the isomorphism I_o in Figure 4. Thus, according to our scheme *CopyOfBook* and *BookCopy* are WC3-components. The incorrect mapping might be explored by considering the *CopyOfBook* as a WU2-component that should have been mapped onto *BookCopy* and vice versa according to H16. Since *CopyOfBook* and *BookCopy* had been identically classified as nominal kind entities the application of H1 and H2 does not reveal any problem in respect of the criterion of semantic homogeneity. Also, suppose that the application of H3 and H4 does not reveal any accidental incorrect common classification for *LoanedCopies* and *BookCopy* or *CopyOfBook* and *Publication* (all of them had been correctly classified as nominal kind entities). On the contrary, as indicated by the computed partial distances of the components, the incorrect mapping was generated because of the unequal attribution distances (d_a*(LoanedCopies, BookCopy)* + d_a*(CopyOfBook, Publication)* < d_a*(CopyOfBook, BookCopy)* + d_a*(LoanedCopies, Publication)*). This suggests that we should apply H9, H10, H11 and H12.

More specifically, H10 leads to the realization that the subcomponent (attribute) *Represents* of *CopyOfBook* is a WU3-component. In other words, it is missing from the specification of *BookCopy* in S2. By way of H13, the viewpoint owners decide to create a new attribute, called *CopyOf*, for the object type *BookCopy* with value a new object type, called *BookManuscript*. However, the similarity analysis between S1 and S2 still fails to map *Book* onto *BookManuscript* because they are not classified identically with respect to the meta-model. Treated as WU1-components, they can be identically classified as nominal kind components due to H2. Similarly with the attribute *Represents*, the attribute *HasIsbn* of *Book* is identified as a WU3-component and through H13 and H2, a new attribute, having the same name and classification, is created for *BookManuscript*. These modifications result in the specifications and the similarity isomorphism, which is (partially) shown in Figure 5.

However, even the new isomorphism is viewed as ontologically incorrect because of the mapping of *LoanedCopies* onto *Publication* and *LoanableCopies* onto *ReferenceBook*. As identified by the mapping of *CopyOfBook* onto *BookCopy*, *LoanedCopies* and *LoanableCopies* should correspond to subtypes of *BookCopy*. Since no such subtypes exist in S2 *LoanedCopies* and *LoanableCopies* are considered as WC4-components. Consequently (H17), two new object types called *CheckedOutCopies* and *BorrowableCopies*, which are meant to be their counterparts are incorporated in S2 (Figure 6). The mere incorporation of these two components is insufficient to force similarity analysis to map them as required. Thus, *CheckedOutCopies* and *BorrowableCopies* are treated as WU2-components and are classified identically (H2).

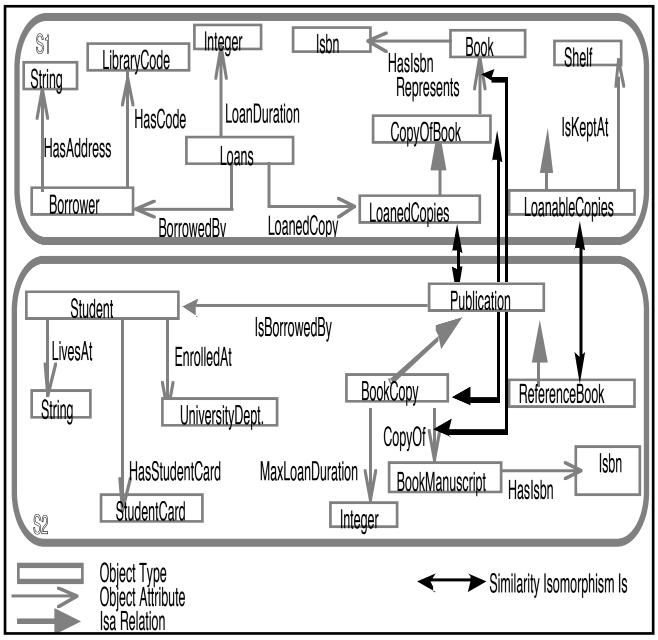

Figure 5. Scenario Step 3

Despite this classification, similarity analysis still maps *LoanedCopies* onto *Publication* rather than *CheckedOutCopies* because of the lower attribution distance. In fact, *LoanedCopies* has one corresponding (*Loans* which corresponds to *IsBorrowedBy*) and one unique subcomponent (the isa relation between *LoanedCopies* and *CopyOfBook*) when compared to *Publication*. On the other hand, both these subcomponents are unique when compared to *CheckedOutCopies*. In trying to reverse this inequality, H10 and H13 lead to the specification of an Isa relation between *CheckedOutCopies* and *BookCopy*, through which *CheckedOutCopies* inherits *IsBorrowedBy*. After these modifications, the attribution distance between *LoanedCopies* and *CheckedOutCopies* becomes lower than the attribution distance between *LoanedCopies* and *Publication* since the former pair has two corresponding subcomponents while the latter has two unique. Hence, *LoanedCopies* is also mapped onto *BorrowableCopies* by similarity analysis.

Similarly, the identical classification of *BorrowableCopies* and *LoanableCopies* is insufficient to enforce a mapping since the attribution distance between *LoanableCopies* and *ReferenceBook* is lower than the attribution distance between *LoanableCopies* and *BorrowableCopies*. In fact, *BorrowableCopies* and *LoanableCopies* have one unique subcomponent (the attribute *IsKeptAt*) and one pair of corresponding subcomponents (the isa relations connecting *LoanableCopies* with *CopyOfBook* and *ReferenceBook* with *Publication*), while *LoanableCopies* and *ReferenceBook* have two unique subcomponents (the attribute *IsKeptAt* and the isa relation connecting *LoanableCopies* with *CopyOfBook*).

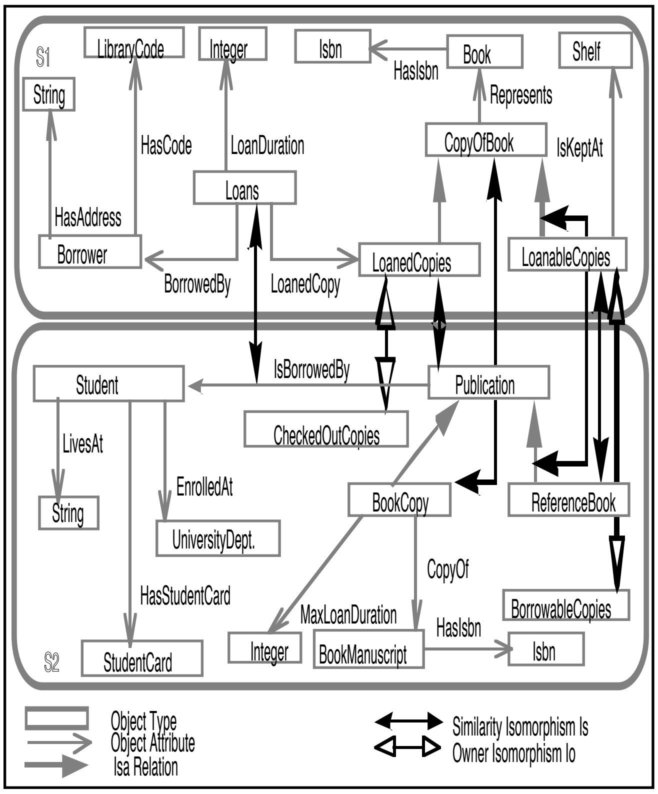

Figure 6. Scenario Step 4

In trying to reverse this inequality, H9 leads to the identification of the relation connecting *LoanableCopies* with *CopyOfBook* as a WU3-component. By way of H13, a new isa relation, connecting *BorrowableCopies* with *BookCopy*, is created, intended to be its counterpart. However, this modification does not reverse the attribution distance inequality, since the new isa relation gives rise to the inheritance of the attribute *IsBorrowedBy* from *Publication* to *BorrowableCopies* through *BookCopy*. This subcomponent is unique in respect of the similarity isomorphism between *BorrowableCopies* and *LoanableCopies*. As a result of H10, *IsBorrowedBy* is realized as a WU4-subcomponent for *BorrowableCopies*, since a borrowing relation cannot involve items which have not been checked out from the library, as specified in S1.

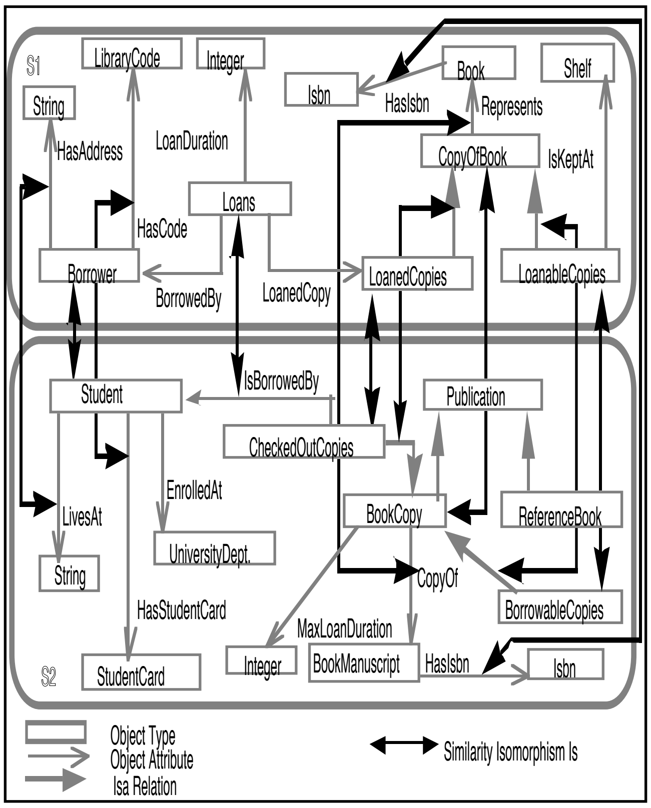

Figure 7. Scenario Step 5

The operationalisation of H15 in this case involves the re-modeling of *IsBorrowedBy* as an attribute of *CheckedOutCopies* rather than *Publication*. Thus, it is no longer inherited by *BorrowableCopies*, the attribution distance between *BorrowableCopies* and *LoanableCopies* is reduced and these two components are mapped onto each other by similarity analysis. Specifications S1 and S2 have now been revised and their similarity analysis generates an isomorphism which has been verified as ontologically correct by viewpoint owners, see Figure 7. At this point reconciliation may stop.

Along the way S1 and S2 have undergone modifications that made them compatible with their owners assessment about the existence of ontological overlaps between them. New information, originally missing from it, was elicited in S2 as a result of its comparison with S1. Also, through the process, viewpoint owners established a shared understanding about the ontological overlaps in their specifications and the potential for introducing further inconsistencies in them.

5 TOOL SUPPORT

The analysis stage in specification reconciliation is currently supported by a prototype built as a customisation of the Semantic Index System (SIS), a tool for representing, storing and retrieving objects described in the Telos language [4]. This is integrated with an implementation of the similarity model and provides queries for detecting similarities between Telos objects. The meta-model for specification analysis has been implemented as a kernel SIS object base, which is used as a schema for describing specifications. Specifications are described as SIS objects classified using this schema and are therefore amenable to similarity analysis. This process of description is supported by interactive data entry forms, built-in the SIS, and customized to support the task of classifying specification components.

To support the full reconciliation method discussed above we are building a process model, using techniques presented in [16], describing the activity of exploring and rectifying wrong unique and corresponding components based on the heuristics. The model will allow viewpoint owners to switch between the stages of analysis and revision if they feel it is necessary, and guide — rather than forcing — them to adopt specific, predefined types of reconciliation [8].

6 RELATED WORK

Research in requirements engineering has primarily concentrated on the detection and resolution of logical inconsistencies, taking for granted the detection of ontological overlap. Some of the approaches focus on the detection of particular types of inconsistencies between viewpoints expressed in specific representation schemes [7, 8, 11, 22], while other are concerned with inconsistency in general [28]. Considerably less work has examined the detection of ontological overlap. This work has been based either on the generation of canonical representations of viewpoints in a common underlying language, hence making it easier to compare them [13,17] or on elaborating analogies between viewpoints. This has been carried out either by matching viewpoints with classes of requirements engineering problems [18] or by establishing analogies from annotations of viewpoints with terms in domain-specific dictionaries [15].

Interference is also a central issue for the interoperability of multiple database systems and has been dealt with by tight and loose coupling approaches. The tight-coupling approaches [1, 2, 5, 24] integrate local database schemas into one or more global schemas after detecting semantic equivalencies and disparities between them. Integration is not fully automated. The loose-coupling approaches [3, 23] ensure the consistent exchange of semantically equivalent information by deploying conversion functions, supplied by local database systems delegates.

7 CONCLUSION

The construction of complex software systems involves many agents with different perspectives or views of the system they are trying to describe, which give rise to many partial specifications (or viewpoints). Viewpoints "interfere" with each other to the extent they refer to, or assert properties of common aspects of the system under development and its domain (i.e. ontological overlap), which in turn might be inconsistent with each other. This interference needs to be "managed". Reconciliation, the method discussed in this paper is a method of loose interference management. It detects ontological overlaps (a prerequisite for detecting inconsistencies) by analysing similarities between viewpoints and guides viewpoint owners through a process of assessing and verifying them, thus establishing a shared understanding among these owners of the potential for inconsistency. We believe that the method has promise though there is clearly considerable scope for further work. In particular we will be looking at extending the tool support and at larger scale examples which would constitute a more realistic test.

REFERENCES

[1] Arens, Y. and Knoblock, C. 1992. Planning and Reformulating Queries for Semantically-Modeled Multidatabase Systems, In Proceedings of the 1st International Conference on Information and Knowledge Management, pp. 92-101.

[2] Batini, C., Lenzerini, M. and Navathe, S. 1986. A Comparative Analysis of Methodologies for Database Schema Integration, ACM Computing Surveys, 18(4), pp. 322-364.

[3] Bright, M., Hurson, A. and Pakzad, P. 1994. Automated Resolution of Semantic Heterogeneity in Multidatabases, ACM Transactions on Database Systems, 19(2), June, pp. 212-253.

[4] Constantopoulos, P. and Doerr, M. 1993. The Semantic Index System: A Brief Presentation, Institute of Computer Science, Foundation for Research and Technology-Hellas, Heraklion, Crete, Greece (available from:*http://www.ics.forth.gr/proj/isst/Systems/SIS/index.html*).

[5] Collet, C., Huhns, N. and Shen, W-M. 1991. Resource Integration Using a Large Knowledge Base in Carnot, IEEE Computer, 24(12), December, pp. 55-63.

[6] Easterbrook, S. 1991. Handling Conflict between Domain Descriptions with Computer-Supported Negotiation, Knowledge Acquisition, 3, pp. 255-289.

[7] Easterbrook, S. et al. 1994. Co-Ordinating Distributed ViewPoints: The Anatomy of a Consistency Check, International Journal on Concurrent Engineering: Research & Applications, 2,3, CERA Institute, USA, pp. 209-222.

[8] Finkelstein, A. et al. 1994. Inconsistency Handling in Multi-Perspective Specifications, IEEE Transactions on Software Engineering, 20(8), pp. 569-578.

[9] Finkelstein, A. and Sommerville, I. 1996. The Viewpoints FAQ, Software Engineering Journal: Special Issue on Viewpoints for Software Engineering, 11(1), pp 2-4.

[10] Goh, C., Madnick, S. and Siegel, M.. 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

[11] Heitmeyer, C. et al. 1995. Consistency Checking of SCR-Style Requirements Specifications, In Proceedings of the IEEE International Conference on Requirements Engineering, York, England, pp. 56-63.

[12] Hunter, A. and Nuseibeh, B. 1995. Managing Inconsistent Specifications: Reasoning, Analysis and Action, Department of Computing Technical Report Number 95/15, Imperial College, London, UK, October.

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

[14] Kotonya, G. and Sommerville, I. 1992. Viewpoints for Requirements Definition, Software Engineering Journal, 7(6), pp. 375-387.

[15] Leite, J. and Freeman, P. 1991. Requirements Validation Through Viewpoint Resolution, IEEE Transactions on Software Engineering 17(12), pp. 1253-1269.

[16] Leonhard, U. et al. 1995. Decentralised Process Enactment in a Multi-Perspective Development Environment, In Proceedings of the 17th International Conference on Software Engineering(ICSE-17), Washington, USA, pp. 255-264.

[17] Meyers, S. and Reiss, S. 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, pp. 202-209.

[18] Maiden, N. et al. 1995. Computational Mechanisms for Distributed Requirements Engineering, In Proceedings of the 7th International Conference on Software Engineering & Knowledge Engineering, USA, pp. 8-16.

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

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

[21] Nuseibeh, B. et al., 1994. A Framework for Expressing the Relationship between Multiple Views in Requirements Specification, IEEE Transactions on Software Engineering, 20(10), pp. 760-773.

[22] Robinson, W. and Fickas, S. 1994. Supporting Multi-Perspective Requirements Engineering, In Proceedings of the IEEE Conference on Requirements Engineering, pp. 206-215.

[23] Sciore, E., Siegel, E., and Rosenthal, A. 1994. Using Semantic Values to Facilitate the Interoperability Among Heterogeneous Information Systems, ACM Transactions on Database Systems, 19(2), pp. 254-290.

[24] Sheth, A.and Larson, J. 1990. Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases, ACM Computing Surveys, 22(3), pp. 183-236.

[25] Spanoudakis, G. and Constantopoulos, P. 1996. Elaborating Analogies from Conceptual Models, International Journal of Intelligent Systems, (ed) Yager R., J.Wiley and Sons, 1996 (to appear).

[26] Spanoudakis, G. and Constantopoulos, P. 1995. Integrating Specifications: A Similarity Reasoning Approach, Automated Software Engineering Journal, Vol. 2, No 4, pp. 311-342.

[27] Storey, V. 1993. Understanding Semantic Relations, VLDB Journal 3.

[28] Zave, P. and Jackson, M. 1993. Conjunction as Composition, ACM Transactions on Software Engineering and Methodology, 2(4), pp. 379-411.

APPENDIX: THE SIMILARITY MODEL

The similarity model is composed of *distance* measuring functions defined on Telos objects. In this Appendix, we formally introduce these objects and functions.

A1 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 (objects modeling entities) and Attributes (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 , *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).

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 their unique identifiers:

Definition 1: The identification distance d_{id} between [illegible]o objects is defined as:

$$d_{id}(o_i,o_j) = 0 \ \textit{if } i=j \textit{ and } [illegible]$$
$$d_{id}(o_i,o_j) = 1 \ \textit{if } i \quad j$$

(ii) The Classification Distance. The classification distance between two objects is measured by identifying and measuring the importance of their non-com[illegible] [illegible]e is measured by the specialisation de[illegible], *SD(x)*, of each class, fo[illegible]ally defined as:

Definition 2: *SD(x)* is th[illegible]maximum length (number of links)[illegible] the paths connecting class *x* [illegible]ith the most general clas[illegible] of its generalisation taxonomy, call[illegible] *specialisation depth* of x.

Given SD(x), the classification [illegible]stance is defined as fo[illegible]ws:

Definition 3: The [illegible]lassifi[illegible]on distance d_c betwe[illegible] two o[illegible]ects [illegible] defined as:

$$d_c(o_i,o_j) [illegible] (b_c\ D_c(o_i,o_j))/(b_c\ D_c(o_i,o_j) [illegible] 1),\ b_c \quad R^{+}$$
$$D_c(o_i,o_j) = \quad {}_{x \quad NCij} [illegible] {}_j \ ([illegible] {}_j.In) \quad (o_j.In - o_i.In)$$

Thus, in measuring the classification distance between two objects, their non-common classes, which are placed at higher levels in generalisation taxonomies are cons[illegible] those placed at lower levels. b_c is a no[illegible]alization parame[illegible] eva[illegible] so that d_c equals 0.5 when D_c takes its av[illegible]ge value [illegible]en a specific set of objects(i.e. a context-sensitive estima[illegible]n).

(iii) generalisation Distance

Definition 4: The generalisation distan[illegible]e d_g betwee[illegible]wo objects is defined as:

$$d_g(o_i,o_j) = (b_c\ D_c(o_i,o_j))/(b_c\ D_c([illegible] o [illegible] + 1) [illegible] b_c [illegible] o_i, o_j [illegible] {}_c$$
$$d_g(o_i\ ,o_j) = d_o(o_i,o_j) [illegible] o_i, o_j \quad A_c$$

$$D_g(o_i,o_j) = \quad {}_{x \quad NCSij} [illegible]$$
$$NCS_{ij} = (o_i.Isa - \ o_j.Isa) \quad (o_j.Isa - o_i.Isa) [illegible] \{i,j\}$$

$$d_o(o_i,o_j) = 0 \ \textit{if } OC(i)=OC(j) \textit{ and}$$
$$d_o(o_i,o_j) = 1 \ \textit{if } OC(i) \quad OC(j)$$

The generalisation distance between individual classes is measured like their classification distance, except that their superclasses are taken into account. The generalisation distance between attribute classes depends on the identity of their original classes and distinguishes between refined specialisations of the s[illegible] attribute and specialisations between attributes with shared b[illegible]non-identical semantics. b_g is simila[illegible] 3.

(iii) Attribution Distance

Definition 5: The attribution distanc[illegible] [illegible]wo o[illegible]ects is defined as:

$$d_a(o_i,o_j) = (b [illegible] D_a(o_i,o_j)) [illegible] b_a [illegible] (o_i,o [illegible] + [illegible], [illegible] R^{+}$$
$$D_a([illegible]$$
$$\textit{if } o [illegible] A \ = [illegible] \textit{ or} [illegible] \textit{ and} [illegible]$$

$$D_a(o_i,o_j) = \min_{m \quad I(o_i,o_j)} ([illegible] {}_{x_3} [illegible]$$
$$s(x_3)^2 + \quad {}_{x_4 \quad o_j[m]} [illegible]$$
$$\textit{otherwise}$$

where

I(o

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

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

s(x): is the salience of attribute class x computed as described in [25].

[illegible]bution distance is estimated by searching for a minimum distance isomorphism between the attributes of two objects. It is recursively defined through the overall distance between the values of attributes (definition 6 below). Thus, it generates optimal isomorphisms between attributes at all the successive levels of the [illegible] of the objects. b_a is similar to and eva[illegible] [illegible]efinition 3.

(iv) Overall Object Distance. The overall distance is an a[illegible]gregation of the identification, classification, generalisa[illegible] [illegible] distances between objects:

Definition 6: The overall distance *d* betw[illegible]n two object[illegible]s defined as:

$$[illegible] (o_i,o_j) = (b\ D(o_i,o_j))/(b\ D(o [illegible] + 1),\ b \quad R [illegible]$$

$$D(o_i,o_j) = (d_{id}(o_i,o_j)^2 + d_c(o_i,o_j)^2 + d_g([illegible] o_i,o_j)^2 + d_c(o_i,o_j)$$
$$d_g(o_i,o_j) + d_c(o_i,o_j)\ d_a(o_i,o_j) + d [illegible])^{1/2}$$
$$\textit{if } o_i,o_j \quad (I_t [illegible]$$

$$D(o_i,o_j)=(d_{id}(o_i,o_j)^2 + d_c(o_i,o_j)^2 + 3 [illegible] (o_i,o_j)^2 + d [illegible] (o_i,o_j)^2 +$$
$$d(o_i.To,o_j.T [illegible] +$$
$$12\ d_c(o_i,o_j)\ d_g(o_i,o_j) + d_c(o_i,o_j)\ d_a(o_i,o_j) [illegible] 12\ d_g(o_i,o_j)\ d_a(o_i,o_j)\ )^{1/2}$$
$$\textit{if } o_i,o_j \quad (A_t \quad A [illegible]$$