A semi-automatic process of identifying overlaps and inconsistencies between requirements specifications

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

Anthony Finkelstein
Department of Computer Science,
University College London,
Gower Street, London WC1E 6BT
email: a.finkelstein@cs.ucl.ac.uk

Abstract

Reconciliation is a method which supports the detection and verification of overlaps and the resolution of certain forms of inconsistencies between requirements specifications expressed in an object-oriented framework. The method identifies a set of candidate overlaps between two specifications by analysing their similarity. These overlaps are assessed by the authors of the specifications. If the authors disagree with the overlaps identified by analysis, the method guides them through an exploration activity aimed at (1) identifying inconsistencies in the modelling of the specifications with respect to the overlaps indicated by them, and (2) resolving these inconsistencies in a way which ensures that the results of further analysis will converge with overlaps indicated by the authors. This paper provides an overview of the method focusing on the process of identifying and resolving inconsistencies between specifications.

Keywords: inconsistency management, requirements engineering, object-oriented methods

Appeared in the proceedings of the 5th International Conference on Object-Oriented Information Systems (OOIS ‘98) published by Springer-Verlag (ISBN 1-85233-046-5).

1. Introduction

In software engineering settings where different stakeholders may have conflicting requirements for the system to be built, there is no point in worrying about the consistency of the resulting specifications unless you are pretty confident that they refer to the same things within a shared domain of discourse. In other words, unless there is an overlap between these specifications. This paper describes a method, called "reconciliation", which identifies overlaps and particular forms of inconsistencies between specifications expressed in an object-oriented framework, and guides the stakeholders (these may be the authors of the specifications or a third party) in taking steps towards the amelioration of these inconsistencies in a rationalised way. The paper builds upon previous work of the authors on the automated analysis of similarity between heterogeneous specifications [23]. An account of the method at an earlier stage in its development has also been presented in [25].

Overlap may be formally defined as a relation between the interpretations of the components of two specifications [26] and different types of overlap relations may be distinguished. In particular, a pair of specification components have a:

• *total overlap* if the sets of the objects they designate in some domain of discourse are the same (Spanoudakis et al [26] distinguish three types of overlap depending on the exact relation between the interpretations of the specifications)
• *null overlap* if the sets of the objects they designate have no elements in common.

Often, the presence of overlaps dictates the need to check whether specifications satisfy certain consistency rules. A consistency rule is a condition that the two specifications must jointly satisfy. A breach of such a rule manifests itself as a logical inconsistency. The consistency status of specifications needs to be checked and established with reference to specific sets of overlap relations and it might change given different such sets. Formally, two specifications S_i and S_j will be inconsistent with respect to a consistency rule CR when overlapping as indicated by a set of overlap relations $O(S_i,S_j)$ if: $\{S_i ; S_j; O(S_i,S_j)\}$ *entails* $\neg$ CR. Overlap and inconsistency are two levels of specification interference the first of which is a precondition for the second.

Consider for instance two classes in two object-oriented specifications which have different superclasses and are identified as totally overlapping components. These classes would violate a consistency rule demanding that totally overlapping classes must have exactly the same superclasses. This rule would not be violated if the classes did not overlap. In this case it would not even make sense to check the rule.

Specification interference is not undesirable since it provides scope for innovative thinking, deferment of commitments in teamwork, exploration of alternatives, elicitation of information, and enables the focus of attention to aspects of systems that may deserve further analysis. However, it delivers on these promises only if it is appropriately "managed", that is overlaps and inconsistencies are identified, traced and steps are taken towards the amelioration of inconsistencies [8]. Also, it should be clearly appreciated that specifications might have been constructed independently, by stakeholders with varying concerns, backgrounds and knowledge and expressed in different languages. As a consequence they might be at different levels of abstraction, granularity and formality; and might deploy different terminologies. The complexities arising from these forms of heterogeneity – set alongside the normal software engineering problems of scale – make the identification of overlaps and inconsistencies a very complex activity.

Reconciliation supports this activity. The method identifies a set of candidate overlaps between specifications based on an analysis of their structural and semantic similarities. Then the stakeholders review these candidate overlaps and identify those which in their view are "true" overlaps and those which are "false" overlaps. Based on this assessment the method suggests revisions to the specifications which would ensure that the results of further analysis converge with the assessment of the stakeholders. After a number of overlap identification and specification revision cycles the method delivers a set of agreed correspondences between the specifications and the specifications are revised so as to be consistent with these correspondences. The method is applicable to object-oriented specifications.

The rest of the paper is structured as follows. Section 2 provides an overview of existing work on overlap identification. Section 3 describes the identification of overlaps and their assessment by the stakeholders in Reconciliation. Section 4 describes the process by which Reconciliation guides the stakeholders in elaborating and ameliorating inconsistencies and section 5 overviews the tool support currently available. Section 6 gives an example of ameliorating inconsistencies and section 7 summarises the method and outlines directions for future research. An appendix with a formal definition of the similarity analysis model used by the method is also given.

2. Related work

Overlaps between independently constructed specifications have been traditionally identified by:

• *representation conventions* – the simplest and most common representation convention is to assume total overlaps between specification components with identical names and null overlaps between any other pair of elements [5,7,6,9,28,18]
• *shared ontologies* – overlaps are assumed between specification components that have been "tagged" with the same item in a shared ontology used for assigning interpretations to these components [2,14,22]
• *direct human inspection* – the stakeholders explore the specifications and identify overlaps [4]

In essence the approach in both the case of representation conventions and shared ontologies is to identify overlaps between components which satisfy specific consistency rules, known that should hold between overlapping components.

In the case of the "identical names" convention the rule is:
CR1: *If two components x and y overlap then x and y must have identical names*

In the case of the shared ontologies the rule is:
CR2: *If two components x and y overlap then x and y must be annotated with the same item in the ontology*

Note that the approach is *abductive*: the establishment of the consequent parts of the rules leads to the establishment of their antecedent parts. Clearly this way of identifying overlaps is weak since it assumes that overlapping components are always consistent! In reality – especially in the early stages of requirements acquisition and specification – this assumption turns out to be wrong: there may be components which satisfy the rule but are not overlapping (e.g. homonyms in the case of CR1) and components which overlap but are inconsistent with respect to the rule (e.g. overlapping components tagged with different items in a shared ontology by mistake).

Human inspection, on the other hand, might be safer especially if it is performed by the authors of the specifications. However, it is inefficient when it comes to specifications of substantial complexity. Hence a combination of the two approaches where overlaps are assumed between components satisfying specific consistency rules but need to be confirmed by inspectors seems to be the right way to go. Human intervention gives confidence in overlap identification which may benefit from automated reasoning, tool and method support, particularly in settings characterized by problems of scale.

3. Reconciliation: Identification and assessment of overlaps

The reconciliation method combines automated analysis with inspections by humans to detect and verify overlaps between specifications. The automated analysis available is based on a computational model which detects structural and semantic similarities between specifications classified and represented according to a particular meta-model. This meta-model expresses general, domain-independent, semantic modelling properties, and enables the representation of the specifications in a homogeneous way. Both the meta-model and the specifications are described in Telos, an object-oriented conceptual modelling language [17].

3.1 The meta-model

The meta-model consists of a kernel and a set of extensions. The kernel includes classes which represent common, domain-independent, semantic modelling constructs [16,27] and the extensions include classes which represent established specification languages in the common representation framework.

The specification components are classified as instances of the kernel classes subject to the properties they possess (some classes of the kernel are shown as light grey boxes in Figure 1). At a high level of abstraction the components are distinguished into those representing entities and those representing relations. Entity representing components are further distinguished into natural, nominal, place, event, activity, state, agent and physical quantity components. Components representing relations are initially distinguished by their arity (e.g. binary or n-ary relations). Binary relations are further specialised according to: cardinality constraints (e.g. 1:1, N:M, total and onto relations); mathematical properties (e.g. symmetric, transitive and set-inclusion relations); existential dependencies between the items they relate; and other general semantic constraints, such as the temporal coexistence, physical separability or homogeneity of the substance of the items they relate [27].

The extensions to the kernel comprise classes which represent modelling constructs of established specification languages. One of the current extensions includes classes which represent the models of the Object-Oriented Software Engineering (OOSE) method [11].

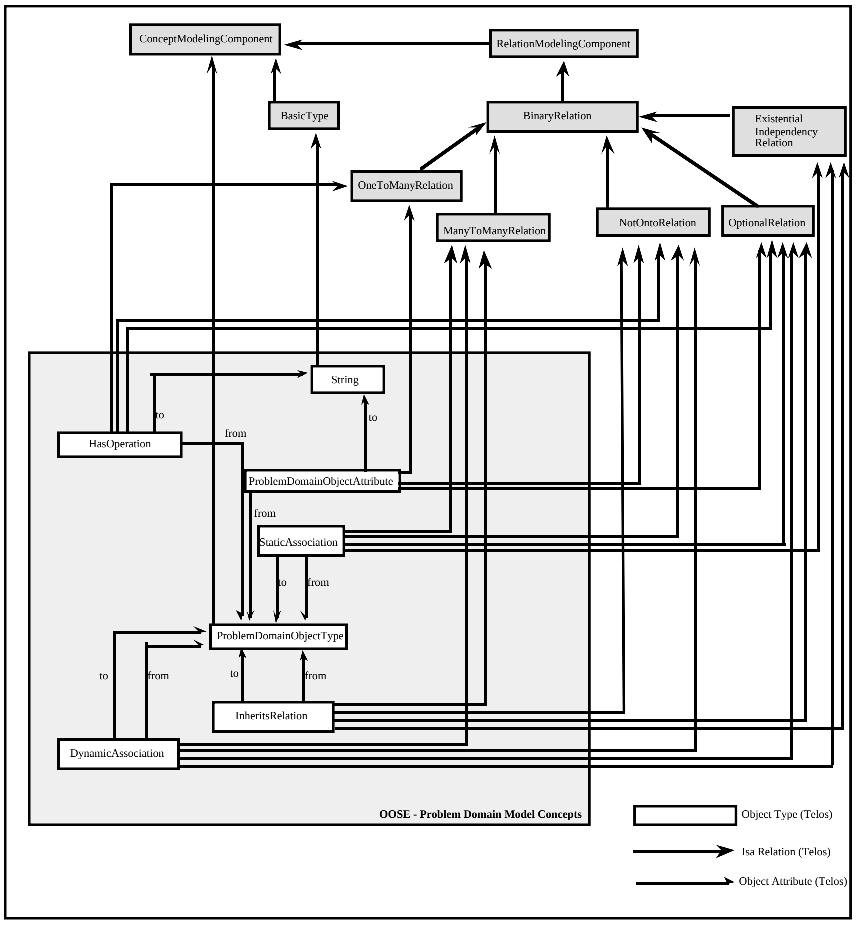


Figure 1: An extension of the kernel meta-model for the "Problem Domain Object Model" of OOSE

Figure 1 shows a part of the extension which represents the "Problem Domain Object Model" of OOSE (PDOM). The elements of PDOM have been introduced as subclasses of the classes in the kernel of the meta-model subject to their semantics. Consider for instance the "Inherits" relation in the PDOM. This relation is represented by the class *InheritsRelation* which associates "problem domain object types" and is introduced as a subclass of:

- *BinaryRelation* – Inherits is a binary relation between problem domain object types
- *ManyToManyRelation* – a problem domain object type may inherit from several problem domain object types and may be inherited by more than one problem domain object types
- *OptionalRelation* – there may be problem domain object types which do not inherit from any other problem domain object type
- *NotOntoRelation* – there may be problem domain object types which are not inherited by any other problem domain object type
- *ExistentialIndependencyRelation* – the existence of a problem domain object type does not depend on the existence any of the object types it inherits from

A detailed description of the kernel of the meta-model is given in [23]. The extensions of the meta-model for the various models of the OOSE method are described in [1].

3.2 Computational identification of overlaps

Specifications are represented by objects which are created as instances of the classes of the meta-model. The attributes of these objects are used to aggregate the components of specifications. This representation makes possible the analysis of the similarity of the specifications according to a computational model which is described in [24]. Similarity analysis is based on three main metric functions. These functions measure the conceptual distances between the specification-objects with respect to the classification and generalisation relations as well as the attributes that constitute their descriptions.

The computation of the distance between two specification-objects with respect to their attributes (called *attribution distance*) determines a morphism I_s between their components, which indicates a set of candidate overlaps between them. By virtue of the definition of the function which computes this distance (see function d_a in the appendix) overlaps are assumed between the components of specifications which satisfy the consequent part of the following consistency rule:
CR3: *If the components x and y overlap then they must be classified as instances of the same kernel classes of the meta-model*

In cases where components which are classified within the same kernel classes of the meta-model can be mapped in many ways, the attribution distance selects a mapping between components violates the consequent part of the following consistency rule to the minimum possible extent:
CR4: *If the components x and y overlap then they must have identical classifications within the classes of the extensions of the meta-model, identical attributes, and identical superclasses.*

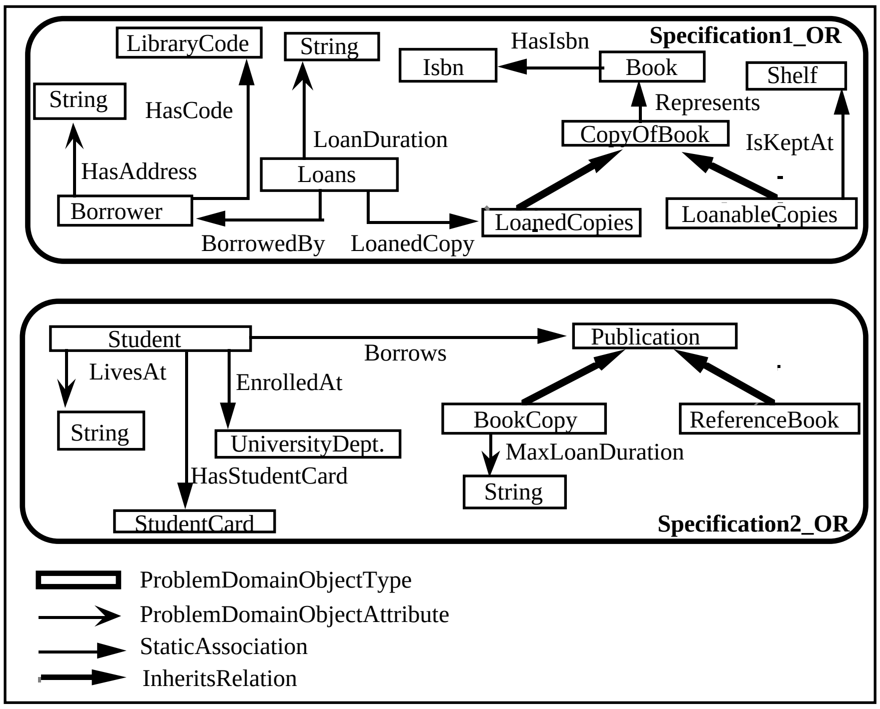
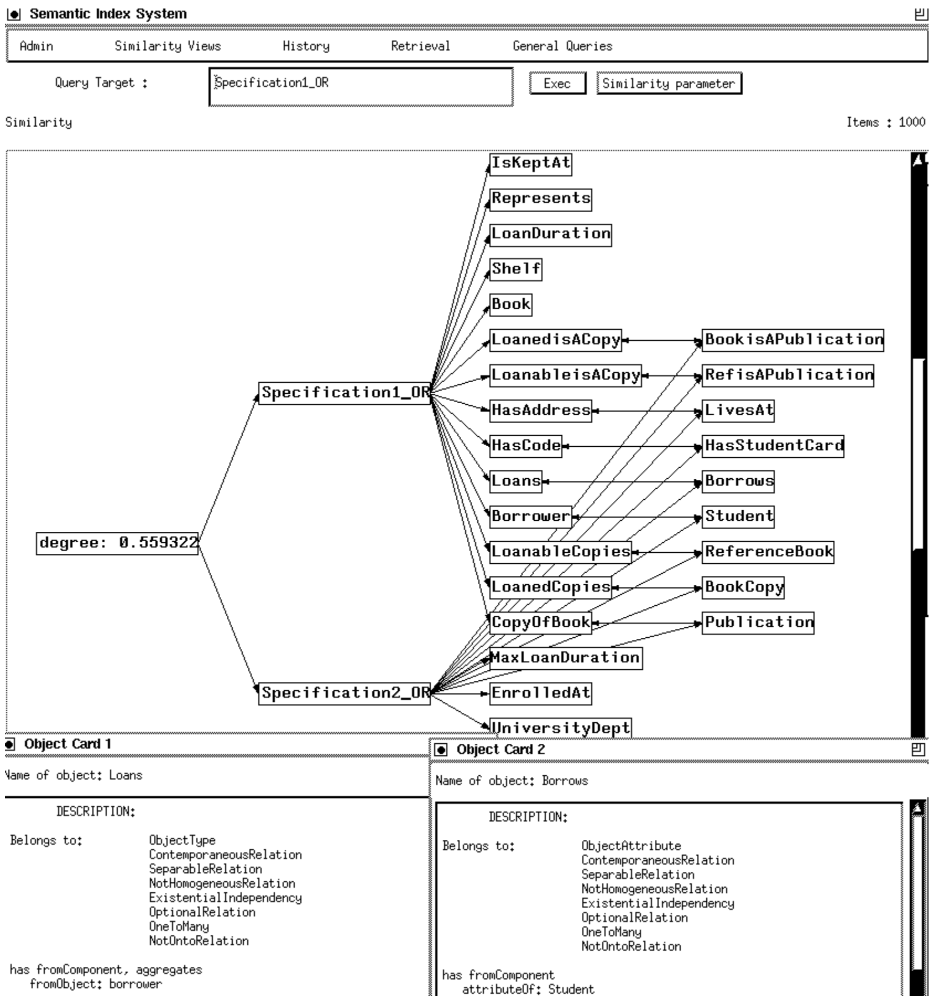


Figure 2: Partial library specifications

The extent to which each pair of components (x,y) in I_s violates the consequent part of CR4 is measured by the overall distance between x and y (see function *d* in the appendix). Spanoudakis and Constantopoulos [24] have shown that d(x,y)=0 if and only if x and y are identical and therefore satisfy the consequent part of CR4. For non-identical components the more the non-identical classes, attributes and superclasses of them the larger the overall distance between them and therefore the larger the extent to which they violate CR4.

The estimation of the pairwise distances d(x,y) between specification components uses a recursive generation of morphisms between their own substructures and depends on the classification and generalisation distances between these components (see functions d_c and d_g in the appendix). These two distance metrics are computed by identifying the non-common classes and superclasses of the components involved, measuring the importance of these classes, and aggregating these importance measures into distance measures.



```
Name of object: Loans

        DESCRIPTION:

 Belongs to:          ObjectType
                      ContemporaneousRelation
                      SeparableRelation
                      NotHomogeneousRelation
                      ExistentialIndependency
                      OptionalRelation
                      OneToMany
                      NotOntoRelation

 has fromComponent, aggregates
    fromObject: borrower
```

```
Name of object: Borrows

        DESCRIPTION:

 Belongs to:          ObjectAttribute
                      ContemporaneousRelation
                      SeparableRelation
                      NotHomogeneousRelation
                      ExistentialIndependency
                      OptionalRelation
                      OneToMany
                      NotOntoRelation

 has fromComponent
    attributeOf: Student
```

Figure 3: Similarity generated overlaps between two library specifications

Consider for example the two partial specifications of a university library system shown in Figure 2. The analysis of their similarity generates the morphism I_s shown in Figure 3. All the components mapped by this morphism satisfy the consequent part of the consistency rule CR3. For instance as shown in Figure 3, *Loans* and *Borrows* are identically classified as contemporaneous, separable, not homogeneous, optional, 1:M, and not onto binary relations. The fact that *Loans* is classified as a PDOM-ObjectType and *Borrows* as a PDOM-ObjectAttribute does not prevent their mapping because these two classes are not part of the kernel of the meta-model.

The morphism shown in Figure 3 was selected amongst other candidate morphisms. These morphisms mapped the components of Specification1_OR and Specification2_OR in ways which satisfy the consequent part of the rule CR3 but violated the rule CR4 to a larger extent. For instance, one of the other morphisms considered by similarity analysis was identical to I_s except that it mapped the PDOM object attribute *LivesAt* in Specification2_OR onto the attribute *LoanDuration* in Specification1_OR rather than the attribute *HasAddress*. The mapping of *LivesAt* onto *LoanDuration* does not violate the rule CR3 since these attributes are identically classified within the kernel classes of the meta-model. However, it violates the consequent part of the rule CR4 to a higher extent. This is because the object types of *LivesAt* and *LoanDuration* (*Student* and *Loans*) have more modelling discrepancies than the object types of *LivesAt* and *HasAddress* (*Student* and *Borrower*).

3.3 Assessment of overlaps

Clearly flaws, incompleteness or inability to express the correct semantics in the modelling of specifications might force similarity analysis to generate morphisms indicating overlaps which are not correct. After all the approach of the automated analysis is abductive as discussed in section 3.2. Reconciliation copes with this problem by requiring the stakeholders to review the overlaps detected and identify those which in their view are "true" overlaps and those which are "false" overlaps. The stakeholders may propose a different morphism I_o between specification components which reflects overlaps based on their own interpretation of specifications.

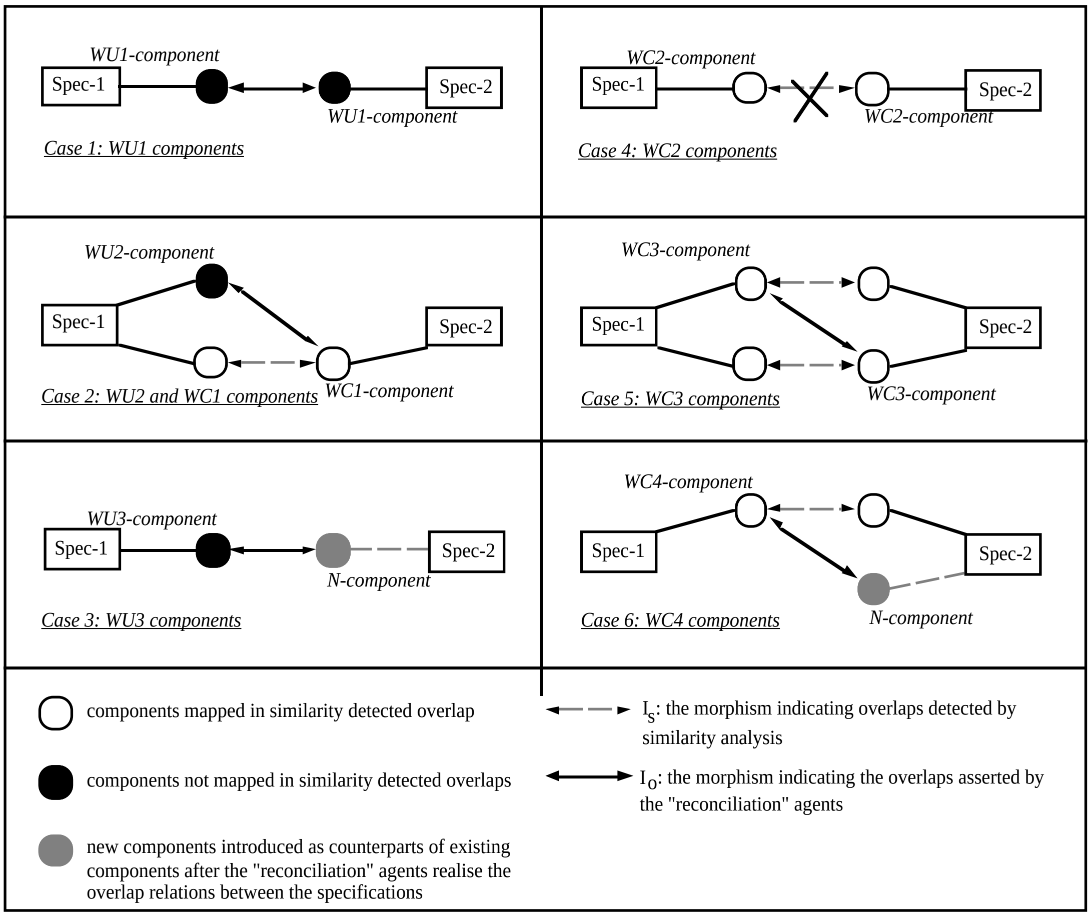

WU1-component
Spec-1
Spec-2
WU1-component
Case 1: WU1 components
WC2-component
Spec-1
Spec-2
WC2-component
Case 4: WC2 components
WU2-component
Spec-1
Spec-2
Case 2: WU2 and WC1 components
WC1-component
WC3-component
Spec-1
Spec-2
Case 5: WC3 components
WC3-component
WU3-component
Spec-1
Spec-2
N-component
Case 3: WU3 components
WC4-component
Spec-1
Spec-2
Case 6: WC4 components
N-component
components mapped in similarity detected overlap
components not mapped in similarity detected overlaps
new components introduced as counterparts of existing components after the "reconciliation" agents realise the overlap relations between the specifications
I_S: the morphism indicating overlaps detected by similarity analysis
I_O: the morphism indicating the overlaps asserted by the "reconciliation" agents

Figure 4: Types of components with "false" overlaps

Based on the morphisms I_s and I_o (both assumed to be directed from the specification Spec1 to the specification Spec2 in Figure 4) we distinguish among 7 different types of wrongly mapped components:

- **WU1-components:** WU1 components (wrong unique components of type 1) are components for which no overlap relation was found by similarity analysis. According to the stakeholders these components have overlapping counterparts which in fact are existing components of the specifications that similarity analysis did not map to any other component (Case 1 of Figure 4). Formally, a component x of a specification Spec1 and a component y of a specification Spec2 form a pair of WU1 components if: $I_s(x)=nil$[1] *and* $I_s^{-1}(y)=nil$ *and* $I_o(x)=y$

- **WU2-components and WC1-components:** WU2 components (wrong unique components of type 2) are components for which no overlap relation was found by similarity analysis. According to the stakeholders WU2 components have overlapping counterparts. These counterparts have been mapped on to different components by similarity analysis which will be referred to as WC1 components (wrong corresponding components of type 1) (Case 2 of Figure 4). Formally, a component x of a specification Spec1 and a component y of a specification Spec2 form a pair of WU1 and WC1 components if:
$I_s(x)=nil$ *and* $I_s^{-1}(y)=z$ *and* $I_o^{-1}(y)=x$ *and* $x \neq z$

- **WU3-components:** WU3 components (wrong unique components of type 3) are components for which no overlapping counterpart was found by similarity analysis. The stakeholders confirm this result but indicate that a new component *(N-component)* must be introduced in the specifications to serve as a counterpart of the WU3 component (Case 3 of Figure 4). Formally, a component x(y) of a specification Spec1(Spec2) is a WU3 component if: $I_s(x)=nil$ *and* $I_o(x)=N$[2] $(I_s^{-1}(y)=nil$ *and* $I_o^{-1}(y)=N)$

- **WC2-components:** WC2 components (wrong corresponding components of type 2) are components which similarity analysis has mapped onto each other. According to the stakeholders these components do not overlap not only with each other but also with any other component of the specifications (Case 4 of Figure 4). Formally, a component x of a specification Spec1 and a component y of a specification Spec2 form a pair of WC2 components if: $I_s(x)=y$ *and* $I_o(x)=nil$ *and* $I_o^{-1}(y)=nil$

- **WC3-components:** WC3 components (wrong corresponding components of type 3) are components for which the stakeholders identify overlapping counterparts which are different from those detected by similarity analysis. The counterparts that the stakeholders indicate have been mapped onto different components by similarity analysis (Case 5 of Figure 4). Formally, a component x of a specification Spec1 and a component y of a specification Spec2 form a pair of WU1 components if: $I_s(x)=u$ *and* $I_s^{-1}(y)=w$ *and* $I_o(x)=y$ *and* $u \neq y$ *and* $w \neq x$

- **WC4-components:** A WC4 component (wrong corresponding component of type 4) is a component which the similarity analysis has mapped onto a counterpart but the stakeholders think that should correspond to a new component

[1] *nil* indicates that a component has no counterpart in the morphism.
[2] *N* indicates a new component introduced in a specification as a counterpart of a component of the other specification.

(N-component) that has to be introduced in the specifications (Case 6 of Figure 4). Formally, a component x(y) of a specification Spec1(Spec2) is a WC4 component if: $I_s(x)=u$ *and* $I_o(x)=N$ ($I_s^{-1}(y)=w$ *and* $I_o^{-1}(y)=N$)

4. The process of overlap elaboration and inconsistency amelioration

The components mapped by I_o but not I_s violate the consequent parts of at least one of the rules CR3 and CR4 since similarity analysis failed to map them. Reconciliation guides the stakeholders through an exploration activity aimed at (1) identifying the inconsistencies in the modelling of these components with respect to CR3 and CR4 given the overlaps indicated by I_o, and (2) suggesting resolutions of these inconsistencies.

This activity is described by a process model which articulates a way of identifying elements in the modelling of specification components that caused the inconsistencies and prevented similarity analysis from generating the "true" overlaps indicated by I_o. This process model has been specified as an instance of the NATURE process meta-model [20,21] and represented in Telos.

4.1 Specification of process models in NATURE

According to the process meta-model of NATURE a process is described as a graph of *contexts*. A context represents the decision to pursue a specific goal (*intention*) in a given *situation*. A situation is defined as a condition which regards the state of the product being manipulated by the process or has to be confirmed by the person who enacts the process model. Conditions may be atomic or composite (disjunctions or conjunctions of atomic conditions). Contexts are distinguished into:

• *executable contexts* – these are contexts which have intentions that can be directly satisfied by performing an *action* which changes the state of the product
• *plan contexts* – these are contexts which have intentions decomposed into a set of sub-goals and can be achieved only by satisfying these sub-goals in a specific order
• *choice contexts* – these are contexts which have intentions which are refined into one or more alternative goals and can therefore be satisfied by satisfying either of these goals

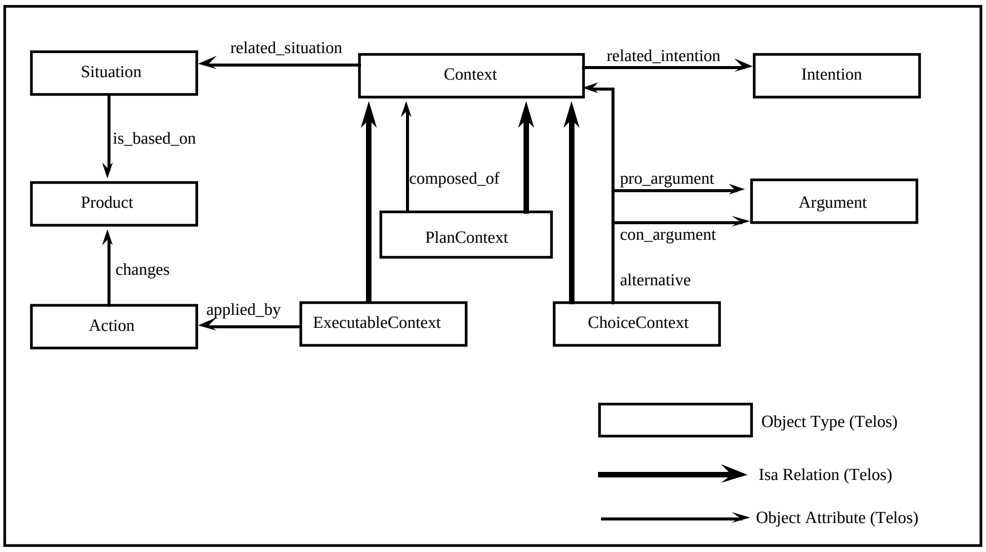


Figure 5: The process meta-model of NATURE (simplified version of Fig. 38 in [20])

Pohl [20] shows how the process meta-model of NATURE can be described in Telos and subsequently how process models can be described as instances of it in the same language. Figure 5 shows a graphical view of the description of the process meta-model in Telos.

4.2 The process model of "reconciliation"

The process model of Reconciliation is shown in Figure 6. Note that the process meta-model of NATURE leaves open the specification of the conditions which define situations. In the case of the Reconciliation process model these conditions are specified as queries over the state of the product of the process and queries that are to be answered by the stakeholders (e.g. atomic conditions *correctIn(x,S)* and *incorrectIn(x,S)* in Figure 6; the stakeholder is assumed to give a "yes" answer if he/she activates the relevant context). The product of the process of Reconciliation comprises four objects that may be modified during it. These include the pair of the specifications involved (Spec1 and Spec2), an object representing the morphism I_s and an object representing the morphism I_o. I_s may change as similarity analysis may be re-performed after a certain number of specification revisions. I_o may also be expanded to reflect overlaps between components overlooked at an earlier stage in the process or modified if the stakeholders change their mind about the interpretations of the specifications and consequently about the overlaps between them. The main paths of the process of Reconciliation are described in the next section.

4.2.1 Starting Reconciliation

Reconciliation starts when the stakeholders activate similarity analysis in order to generate the morphism I_s which indicates the candidate overlaps. This is possible by activating the executable context C2. Subsequently they may assess these overlaps by activating the executable context C3. This context invokes an interactive form which is used to define the correct morphism I_o. If I_s and I_o are not identical, the stakeholders may explore the modelling of the components involved in "false" overlaps. This requires the activation of the choice context C4. C4 gives them a number of options, each dedicated to exploring one of the types of the wrongly mapped components discussed in section 3.4.

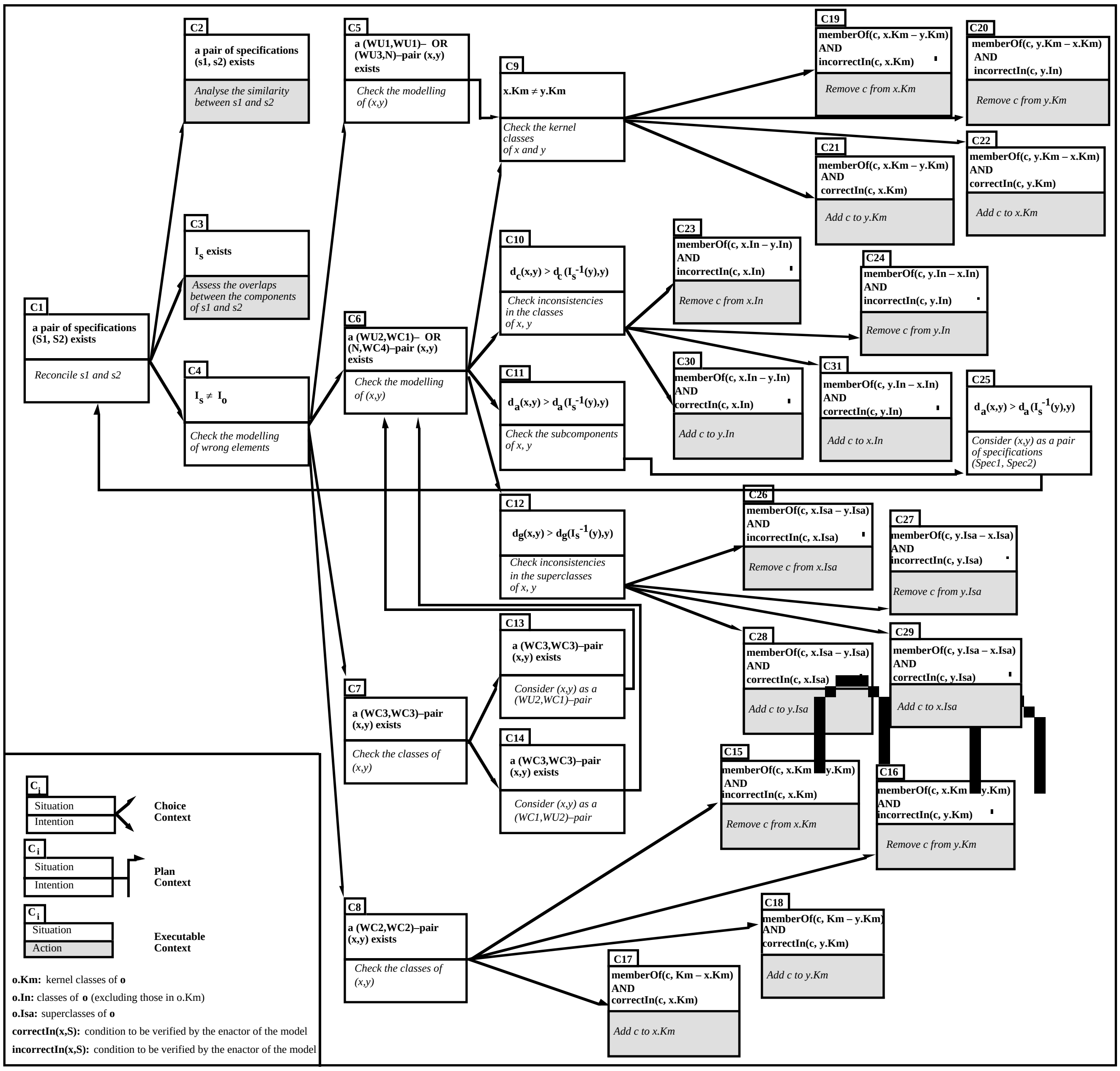

C1
a pair of specifications (S1, S2) exists
Reconcile s1 and s2
C2
a pair of specifications (s1, s2) exists
Analyse the similarity between s1 and s2
C3
Is exists
Assess the overlaps between the components of s1 and s2
C4
Is ≠ Io
Check the modelling of wrong elements
C5
a (WU1,WU1)– OR (WU3,N)–pair (x,y) exists
Check the modelling of (x,y)
C6
a (WU2,WC1)– OR (N,WC4)–pair (x,y) exists
Check the modelling of (x,y)
C7
a (WC3,WC3)–pair (x,y) exists
Check the classes of (x,y)
C8
a (WC2,WC2)–pair (x,y) exists
Check the classes of (x,y)
C9
x.Km ≠ y.Km
Check the kernel classes of x and y
C10
dc(x,y) > dc (Is-1(y),y)
Check inconsistencies in the classes of x, y
C11
da(x,y) > da (Is-1(y),y)
Check the subcomponents of x, y
C12
dg(x,y) > dg(Is-1(y),y)
Check inconsistencies in the superclasses of x, y
C13
a (WC3,WC3)–pair (x,y) exists
Consider (x,y) as a (WU2,WC1)–pair
C14
a (WC3,WC3)–pair (x,y) exists
Consider (x,y) as a (WC1,WU2)–pair
C15
memberOf(c, x.Km – y.Km) AND incorrectIn(c, x.Km)
Remove c from x.Km
C16
memberOf(c, x.Km – y.Km) AND incorrectIn(c, y.Km)
Remove c from y.Km
C17
memberOf(c, Km – x.Km) AND correctIn(c, x.Km)
Add c to x.Km
C18
memberOf(c, Km – y.Km) AND correctIn(c, y.Km)
Add c to y.Km
C19
memberOf(c, x.Km – y.Km) AND incorrectIn(c, x.Km)
Remove c from x.Km
C20
memberOf(c, y.Km – x.Km) AND incorrectIn(c, y.In)
Remove c from y.Km
C21
memberOf(c, x.Km – y.Km) AND correctIn(c, x.Km)
Add c to y.Km
C22
memberOf(c, y.Km – x.Km) AND correctIn(c, y.Km)
Add c to x.Km
C23
memberOf(c, x.In – y.In) AND incorrectIn(c, x.In)
Remove c from x.In
C24
memberOf(c, y.In – x.In) AND incorrectIn(c, y.In)
Remove c from y.In
C25
da(x,y) > da (Is-1(y),y)
Consider (x,y) as a pair of specifications (Spec1, Spec2)
C26
memberOf(c, x.Isa – y.Isa) AND incorrectIn(c, x.Isa)
Remove c from x.Isa
C27
memberOf(c, y.Isa – x.Isa) AND incorrectIn(c, y.Isa)
Remove c from y.Isa
C28
memberOf(c, x.Isa – y.Isa) AND correctIn(c, x.Isa)
Add c to y.Isa
C29
memberOf(c, y.Isa – x.Isa) AND correctIn(c, y.Isa)
Add c to x.Isa
C30
memberOf(c, x.In – y.In) AND correctIn(c, x.In)
Add c to y.In
C31
memberOf(c, y.In – x.In) AND correctIn(c, y.In)
Add c to x.In
Ci
Situation
Intention
Choice Context
Ci
Situation
Intention
Plan Context
Ci
Situation
Action
Executable Context
o.Km: kernel classes of o
o.In: classes of o (excluding those in o.Km)
o.Isa: superclasses of o
correctIn(x,S): condition to be verified by the enactor of the model
incorrectIn(x,S): condition to be verified by the enactor of the model

Figure 6: The process model of Reconciliation

4.2.2 Exploring inconsistencies between WU1 and WU3 components

The WU1 components are components which similarity analysis failed to map because they were not classified as instances of the same kernel classes of the meta-model and therefore they violate the rule CR3. If these components had been classified as instances of the same kernel classes of the meta-model, similarity analysis would have mapped them (rather than leaving them without counterparts) since this would minimise the overall distance between the specifications (or equivalently it would reduce the extent to which the rule CR4 is violated). Similar classification discrepancies might prevent the mapping of WU3 components onto their newly introduced counterparts (N-components). Thus, the investigation of possible inconsistencies between WU1 or WU3 and N components should focus on their non-common kernel classes of the meta-model (context C9). The activation of C9 leads to the identification of the kernel classes of the meta-model which are classes of only one of the components involved (this is the result of the process enactment mechanism checking the conditions *memberOf(c, x.Km – y.Km)* and *memberOf(c, y.Km – x.Km)* in order to identify which of the options of C9 become available. The classification of one of the components involved as an instance of a kernel class which is not a class of the other might be deemed to be incorrect and consequently removed (executable contexts C19 and C20). Alternatively, if a non common kernel class is confirmed to be correct then the only way to resolve the inconsistency is to add it to the list of the classes of the other component (executable contexts C21 and C22).

4.2.3 Exploring inconsistencies between WU2 and WC1 or WC4 components

A pair of a WU2 and a WC1 component or a pair of WC4 components might also have not been mapped on to each other by similarity analysis because of classification inconsistencies with respect to the kernel classes of the meta-model which violate the rule CR3. These inconsistencies can be explored by activating the context C9.

Another reason that may lead to a pair of a WU2 and a WC1 component or a pair of WC4 components is that their mapping in I_o would result in a relatively extended violation of the rule CR4 than their mapping in I_s. In other words the mapping of these components suggested by the stakeholders would result in a relatively higher distance than the mapping suggested by similarity analysis: $d(I_s^{-1}(y),y) < d(x,y)$, $x=I_o^{-1}(y)$. The overall distance between a component y and its counterpart in I_s, $I_s^{-1}(y)$, might be lower than the overall distance between y and its counterpart in I_o, $I_o^{-1}(y)$ if one or more of the following inequalities regarding the classification, generalisation or attribution distances between these pairs of components holds: $d_c(I_s^{-1}(y),y) < d_c(x,y)$, $d_g(I_s^{-1}(y),y) < d_g(x,y)$ or $d_a(I_s^{-1}(y),y) < d_a(x,y)$. Depending on which of these inequalities holds the stakeholders may check for inconsistencies in the classes, superclasses or the subcomponents of the components involved.

If $d_c(I_s^{-1}(y),y) < d_c(x,y)$ is true the stakeholders have the option of activating the context C10 to explore and resolve inconsistencies regarding the classifications of x and y. The process is similar to the process of exploring classification inconsistencies in the case of WU1 and WU3 components, except that the non kernel classes of the components involved are taken into account. This part of the process is described by the contexts C29, C30, C24, and C3.

If $d_g(I_s^{-1}(y),y) < d_g(x,y)$ is true the stakeholders have the option of activating the context C12 to explore and resolve inconsistencies regarding the superclasses of x and y. The classes which are not superclasses of both the components involved are identified (this is the result of checking the conditions memberOf(c, x.Isa – y.Isa) and memberOf(c, y.Isa – x.Isa)) and the stakeholders have the option of declaring them as superclasses of both the components (contexts C28 and C29) or remove the relevant generalisation relationships (contexts C26 and C27).

If $d_a(I_s^{-1}(y),y) < d_a(x,y)$ is true the stakeholders have the option of exploring the discrepancies between the subcomponents of x and y (context C11 and C25). This exploration is similar to the exploration of the modelling of S1 and S2 and therefore may be carried out by considering the components x and y as top level specifications and activating the context C1.

4.2.4 Exploring inconsistencies between WC3 components

A pair of WC3 components might not have been mapped onto each other for the same reasons that do not allow a pair of a WC1 and a WU2 component to map onto each other. Thus WC3 components may be explored by activating the contexts of the process model available for WC1 and WU2 components (contexts C7, C13, and C14).

4.2.5 Exploring inconsistencies between WC2 components

WC2 components may occur due to the assumption made by similarity analysis that overlaps exist between components which satisfy the rule CR4 to a maximum possible extent. Note that due to this assumption even dissimilar components may be mapped onto each other provided that they are classified within the same kernel classes of the meta-model (i.e. they satisfy the rule CR3). This because if these components had been left without counterparts the overall distance between the specifications would be larger. If the stakeholders insist that such components should not overlap, it is worth investigating if their classifications within the kernel classes of the meta-model are identical by incident. If they are the stakeholders have the option to modify them in order to prevent similarity analysis from generating the wrong mappings (executable contexts C15 and C16). Alternatively, they may identify kernel classes of the meta-model which should have been declared as classes of only one of the components involved and add the relevant classification links (executable contexts C17 and C18). This would make the classification of the components involved non identical and therefore it would prevent analysis from generating the wrong mappings.

5. Tool support

Currently Reconciliation is supported by a prototype built as a customisation of the Semantic Index System(SIS [3]). SIS is a tool for developing, browsing and querying Telos object bases. The customised SIS tool has been integrated with a module which performs the similarity analysis. 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 description is supported by a tool of interactive data entry forms used to develop SIS object bases. This tool has been customised to support the task of classifying specification components. Similarly the Reconciliation process model has been implemented as a set of SIS objects classified using a schema which defines the process meta-model. The system enables the browsing of the contexts which are found to be applicable in a current situation as determined by SIS queries which express the conditions of the process model.

6. Amelioration of inconsistencies

The process outlined in section 4.2 ideally leads to a point where similarity analysis generates a morphism reflecting overlaps which are verified by the stakeholders unless they decide to abandon it before reaching this point. Along the way the specifications are modified in a way that makes them more consistent with the rules CR3 and CR4 given the overlaps indicated by the stakeholders. Note that they may never become entirely consistent with these rules but as we discussed in section 1 targeting full consistency at early stages of requirements specification is not always desirable. Regardless of whether or not full consistency is achieved one of the novel features of Reconciliation is that it provides a way of ameliorating specific forms of inconsistencies and measuring the progress made in this respect.

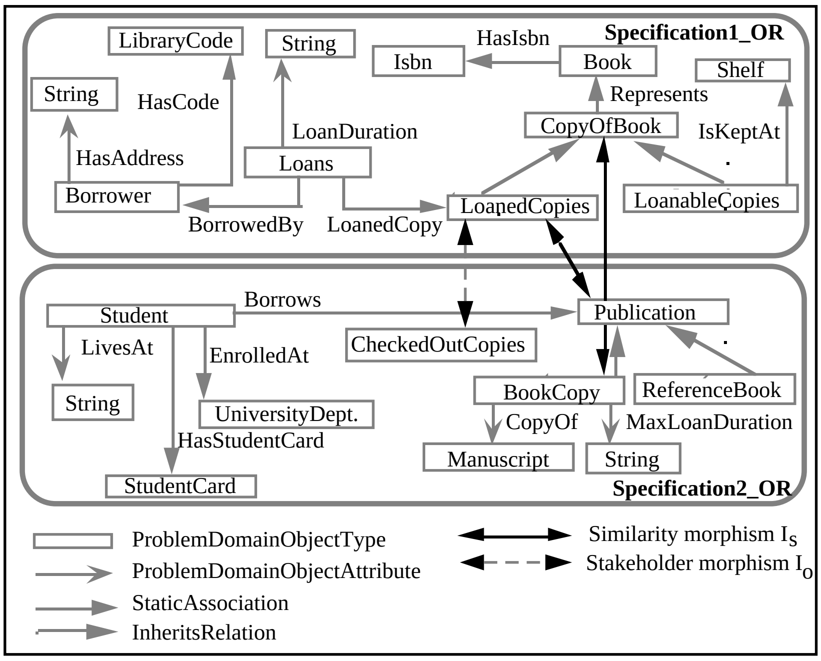


Figure 7 : "False" overlaps

Consider as an example the enactment of the reconciliation process model assuming the specifications and the morphisms I_s and I_o between them shown in Figure 7. *LoanedCopies* and *CheckedOutCopies* constitute a pair of a WC1 and a WU2 component. Thus, the context C6 may be activated to explore the inconsistencies between them. In this particular example, *LoanedCopies* and *CheckedOutCopies* were not mapped onto each other by similarity analysis because as a pair they violate the rule CR4 more than the pair *LoanedCopies* and *Publication*. This is because *LoanedCopies* has three subcomponents (the inherits relation to *CopyOfBook*, the static association *Represents* which is inherited from *CopyOfBook*, and the problem domain object type *Loans*) which have no counterparts in *CheckedOutCopies*. This reflects on the attribution distance between these two components which is d_a(LoanedCopies,CheckedOutCopies)=1. On the other hand, *LoanedCopies* has only two components without counterparts in *Publication* (the inherits relation to *CopyOfBook* and the static association *Represents*) and a subcomponent (*Loans*) with a counterpart in it (the static association *Borrows*). Note also, that although *Loans* and *Borrows* are not identical they are similar (e.g. they both express binary relations of the same cardinalities). The attribution distance between these components is d_a(LoanedCopies,Publication)=0.502. Thus if *LoanedCopies* and *Publication* are taken as overlapping components (as indicated by I_s) they violate the rule CR4 less than *LoanedCopies* and *CheckedOutCopies*.

Since d_a(LoanedCopies,Publication) < d_a(LoanedCopies,CheckedOutCopies) the context C11 becomes applicable and may be activated by the stakeholders to explore the inconsistencies between the subcomponents of *LoanedCopies* and *CheckedOutCopies*. A possible way of exploring and resolving these inconsistencies is to enact the following sequences of contexts of the process model:

• C11, C25, C1,C3: an inherits relation directed to *BookCopy* (N-component) is introduced to *CheckedOutCopies* as the counterpart of the inherits relation of *LoanedCopies* (the N-component is introduced while the stakeholders assess the morphism between the subcomponents of *CheckedOutCopies* and *LoanedCopies*)
• C2, C4, C5, C9, C21: the N-component in *CheckedOutCopies* is classified within the kernel classes of the inherits relation in *LoanedCopie*s

After these modifications since *CheckedOutCopies* has only one subcomponent with no counterparts in *LoanedCopies* (the logical attribute *MaxLoanDuration* inherited from *BookCopy*), the extent to which – together with *LoanedCopies* – it violates CR4 is reduced. The formerly unique subcomponents of *LoanedCopies* (i.e. the inherits relation to *CopyOfBook*, the problem domain object type *Loans* and the static association *Represents*) now have got counterparts in *CheckedOutCopies*. These are the inherits relation to *BookCopy*, the static association *Borrows* (inherited from *Publication*) and the static association *CopyOf* (inherited from *BookCopy*). These modifications reduce the attribution distance between *LoanedCopies* and *CheckedOutCopies* and ameliorate their inconsistency with respect to the rule CR4. A quantitative indication of this amelioration is given by the new attribution distance between these components: d_a(LoanedCopies,CheckedOutCopies)=0.34. Note also that as a result of these modifications similarity analysis maps *LoanedCopies* onto *CheckedOutCopies*, too.

7. Conclusions & future work

Reconciliation, the method discussed in this paper supports the management of overlaps and inconsistencies between specifications. It detects overlaps by analysing similarities between specifications and guides stakeholders through a process of assessing and verifying these overlaps and ameliorating the inconsistencies that may arise as their consequence. We believe that the method has promise though there is clearly considerable scope for further work.

Our immediate plan is to apply the method in a large industrial case study involving the integration of independently developed enterprise models of a large British organisation. This case study requires the extension of the meta-model of the method so as to become applicable to specifications expressed in the Unified Modelling Language (UML) [12]. This extension has already started. Two further important issues we hope to investigate thoroughly in this case study regard the effectiveness of the method in reconciling behavioural specifications (for instance sequence and statechart diagrams in UML) and the use of domain specific ontologies to improve the accuracy of the identification of overlaps during the analysis stage of the method. We are also looking at extending the tool support beyond the functionality which is available from the current prototype. One of the extensions that we seek to introduce to the tool is to give the stakeholders the option to reconcile only subparts of the specifications. Our plan is to develop a tool that would support the method in a way providing value-adding services to existing commercial CASE tools supporting the UML.

References

[1] Apostolopoulos G., (1996), "The Object-Oriented Software Engineering Method in the Viewpoints Framework", MSc Thesis, Department of Business Computing, City University, London, UK

[2] Boehm B., In H., (1996), "Identifying Quality Requirements Conflicts", IEEE Software, March 1996, pp. 25-35.

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

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

[5] Easterbrook, S., A., Finkelstein, J. Kramer, and B. Nuseibeh (1994), "Co-Ordinating Distributed ViewPoints: The Anatomy of a Consistency Check", International Journal on Concurrent Engineering: Research and Applications 2, 3, CERA Institute, pp. 209-222.

[6] Easterbrook, S. and B. Nuseibeh (1995), "Managing Inconsistencies in an Evolving Specification", In Proceedings of the IEEE International Conference on Requirements Engineering, York, England, pp. 48-55.

[7] Finkelstein, A., Gabbay, D., Hunter, A., Kramer, J., and Nuseibeh, B., (1994). "Inconsistency Handling In Multi-Perspective Specifications", IEEE Transactions on Software Engineering, 20, 8, pp. 569-578.

[8] Finkelstein A., Spanoudakis G., Till D.(1996), "Managing Interference", Joint Proceedings of the Viewpoints 96: An International Workshop on Multiple Perspectives in Software Development, San Francisco, USA, pp.172-174.

[9] Heitmeyer, C., B. Law, and D. Kiskis (1995), "Consistency Checking of SCR-Style Requirements Specifications", In Proceedings of the IEEE International Conference on Requirements Engineering, York, England, pp. 56-63.

[10] Hunter, A. and B. Nuseibeh (1995), "Managing Inconsistent Specifications: Reasoning, Analysis and Action", Technical Report TR-95/15, Department of Computing, Imperial College, London, UK.

[11] Jacobson, I. (1995), "Object Oriented Software Engineering: A Use Case Driven Approach", Addison-Wesley, NY.

[12] UML, (1997), "Unified Modelling Language", Version 1.0, Rational Software Corporation, CA, USA *(*available by ftp from *http://www.rational.com)*

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

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

[15] Meyers, S. 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, pp. 202-209.

[16] Motschnig-Pitrik, P. (1993), "The Semantics of Parts vs. Aggregates in Data Knowledge Modeling", In Proceedings of CAiSE '93, LNCS 685, Paris, France, Springer-Verlang, Berlin, pp. 352-373.

[17] Mylopoulos, J., A. Borgida, M. Jarke, and M. Koubarakis (1990), "Telos: Representing Knowledge About Information Systems", ACM Transactions on Information Systems 8, 4, pp. 325-362.

[18] Nissen H., Jeusfeld M., Jarke M., Zemanek G., Huber H., (1996), "Managing Multiple Requirements Perspectives with Metamodels", IEEE Software, pp. 37-47.

[19] 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.

[20] Pohl K. (1996). Process-Centered Requirements Engineering, Advanced Software Development Series, J. Kramer (ed), Research Studies Press Ltd., ISBN 0-86380-193-5, London

[21] Rolland C., Souveyet C., Moreno M. (1995). An Approach for Defining Ways-Of-Working, Information Systems, 20, 4, 337-359

[22] Robinson, W. and S. Fickas (1994), "Supporting Multi-Perspective Requirements Engineering", In Proceedings of the IEEE Conference on Requirements Engineering, IEEE Computer Society Press, Los Alamitos, CA, pp. 206-215.

[23] Spanoudakis, G. and P. Constantopoulos (1995), "Integrating Specifications: A Similarity Reasoning Approach", Automated Software Engineering Journal 2, 4, pp. 311-342.

[24] Spanoudakis G., Constantopoulos P. (1996). "Elaborating Analogies from Conceptual Models", International Journal of Intelligent Systems, Vol. 11, No 11, pp. 917-974.

[25] Spanoudakis G., Finkelstein A. (1997) "Reconciling requirements: a method for managing interference,inconsistency and conflict", Annals of Software Engineering, Special Issue on Software Requirements Engineering, 3, pp. 459-475.

[26] Spanoudakis G., A. Finkelstein, D. Till. (1997). "Interference in Requirements Engineering: The Level of Ontological Overlap", Technical Report Series, TR-1997/01, ISSN 1364-4009, Department of Computer Science, City University, 1997

[27] Storey, V. (1993), "Understanding Semantic Relations", Journal of Very Large Data Bases 3, pp. 455-488.

[28] van Lamsweerde A. (1996), "Divergent Views in Goal-Driven Requirements Engineering", Joint Proceedings of the Sigsoft ‘96 Workshops – Viewpoints ‘96, ACM Press, pp. 252-256.

Appendix: The similarity model

The similarity model is composed of *distance* measuring functions defined on Telos objects. This appendix gives an account of the structure of the objects in Telos and includes the definitions of the main functions of the model.

• **Telos objects:** Telos objects are partitioned according to their roles into objects which model entities called "individuals", and objects which model association or aggregation relations called "attributes". These two categories of objects may be represented using the following tuple forms:

Individual objects: o_i = [L, In, Isa, Sub, A]; Attribute objects: o_i = [L, From, In, Isa, Sub, A, To]

where,

- i is a unique identifier for object o_i;
- L is a string that constitutes the logical name of object o_i (the logical names are unique to individual objects but may be shared by more than one attribute objects as long as they belong to different classes);
- In is a set of object identifiers which denote the classes of o_i (o_i is an instance of each of the classes in In) ;
- Isa is a set of object identifiers which denote the superclasses of o_i;
- Sub is a set of object identifiers which denote the subclasses of o_i;
- A is a set of object identifiers which denote the direct attributes of oi (these are the attributes introduced by o_i as opposed to the attributes inherited by o_i);
- From is the identifier of the object which o_i is an attribute of; and
- To is the identifier of the object which constitutes the value (class-range) of an attribute object (attribute class) o_i.

• **The distance functions**

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

$$[illegible]_{id}(o_i,o_j) = [illegible] \text{ if } i=j \text{ and } d_{id}(o_i,o_j) = 1 \text{ if } i\neq [illegible]$$

Classification distance: The classification [illegible]stance d_c between two objec[illegible] is d[illegible]ined as:

$$d_c(o_i,[illegible]) = (b_c\ D_c(o_i,o_j))/(b_c\ D_c(o_i,o_j) + 1),\ b[illegible]$$

$$D_c(o_i,o_j) = [illegible]_x [illegible]^{-1},\ NCC_{ij} = (o_i.In - o_j.In) [illegible] (o_j.In - o_i.In)$$

where

- b_c is a normalisation parameter evaluated so that d_c equals 0.5 when D_c takes its average value given a specific set of objects (i.e. a context-sensitive estimation of the classification distance).
- *SD(x)* is the maximum length (number of links) of the paths connecting class *x* with the most general class of its generalisation taxonomy, called *specialisation depth* of x.

Generalisation distance: The ge[illegible]ralisation distance d_g between two [illegible] and [illegible]ed as:

$$d_g(o_i,o_j) = ([illegible]_g\ D_g(o_i,o_j))/(b_g\ D_g(o_i,o_j) + 1),\ [illegible]\ o_i,\ o_j\ [illegible]$$

$$d_g(o_i,o_j) = d_o(o_i,o_j) \text{ if } o_i,\ o_j\ [illegible]$$

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

$$d_g(o_i,o_j) = 0 \text{ if } OC(i)=OC(j) \text{ OR } d_g(o_i,o_j) = 1 \text{ if } OC(i)\neq OC(j)$$

b_g is similar to and evaluated like b_c.

Attribution distance: Th[illegible]attribution distance d_a betwe[illegible] two objects o_i [illegible]ed as:

$$d_a(o_i,o_j) = (b_a\ D_a(o_i,o_j))/(b_a\ D_a(o_i,o_j) + 1),\ b[illegible]$$

$$[illegible]D_a(o_i,o_j) = \infty \text{ if } o_i.A [illegible] \varnothing \text{ or } o_j.A = [illegible] \text{ OR}$$

$$D_a(o_i,o_j) = \min_m [illegible]_{(x1,x2)} [illegible] s(x_2)\ d(x_1,x_2) + [illegible]_{x3} [illegible]^2 + [illegible]_{x4} [illegible]) \text{ otherwise}$$

where

- *$I(o_i,o_j)$:* is the set [illegible]ossible morphisms between the semantically homogeneous attributes of o_i and o_j (two attribute objects [illegible]e semantically homogeneous if and only if *OCL[k]= OCL[l] where OCL[x] ={ y | (y = OC(z)) and (z* [illegible]
- *$o_i[m]$ ($o_j[m]$)*: is the set with the attributes of o_i (o_j) that map onto no attribute of o_j (o_i) given the morphism *m*
- *s(x):* is the salience of attribute class x computed as described in [24]
- b_a is similar to and evaluated as b_c.

Overall distance: The overall distance *d* between two objects [illegible] and o_j is d[illegible]

$$d(o_i,o_j) = (b_{oo}\ D(o_i,o_j))/(b_o[illegible] + 1),\ b_{oo}\ [illegible]$$

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

$$\text{if } o_i,o_j\ [illegible]$$

$$d(o_i,o_j) = (b_{oa}\ D(o_i,o_j))/(b_{oa}\ D(o[illegible]),\ b_{oa}\ [illegible]$$

$$D(o_i,o_j)=(d_{id}(o_i,o_j)^2+ d_c(o_i,o_j)^2 +36d_g(o_i,o_j)^2+d_a(o_i,o_j)^2+d(o_i.To,[illegible]2d_c(o_i,o_j)\ d[illegible],o_j)+d_c(o_i,o_j)\ d_a(o_i,o_j)+12d_g(o_i,o_j)$$

$$d_a(o_i,o_j))^{1/2} \text{ if } o_i,o_j\ [illegible]$$