Property Preserving Redesign of Specifications[1]

Artur Zawłocki[2], Grzegorz Marczyński[2], Piotr Kosiuczenko[3]

[2]Institute of Informatics, Warsaw University,
[3]Department of Computer Science, University of Leicester

Abstract. In the traditional formal approach to system specification and implementation, the software development process consists of a number of refinement steps which transform the initial specification into its correct realisation. This idealised view can hardly capture common situations when a specification changes in a non-incremental way, e.g. when client requirements change or new software technologies emerge. An extra flexibility can be added to the development process by allowing for a *redesign of specifications*, in addition to refinement steps. In this paper, the notion of specification redesign is formalised for an arbitrary institution. Basic properties of redesign are investigated and the formalism is applied to provide a formal semantics for UML class diagram transformations. As examples, two refactoring patterns are described in terms of class diagrams and interpreted as redesigns of corresponding algebraic specifications.

1 Introduction

In the contemporary software engineering the phases of the system specification and design occur in a series of interleaving steps. As the system specification changes due to a number of factors including changed or new client requirements, new technology enablers etc., an extensive re-engineering of the system specification and design is often needed. In the algebraic approach to the system specification [AKKB99] the progress of the software development process is often described in terms of refinement which by monotonicity assumption can not express the non-incremental changes to the system structure

We perceive a signature of the system specification as a description of the system structure. Therefore changes of the system structure should be reflected by a changed specification over a new signature. We define in an institution-independent way [BG92] a *redesign of specifications* via embeddings of two specifications to a intermediate specification over a "joint" signature. That intermediate specification determines the strength of the connection between these two.

Our definition of redesign of specifications has numerous applications. One of them is the ability to reason about the transformation of the class structure of the object-oriented systems.

[1] This research was supported by the EC 5th Framework project AGILE: Architectures for Mobility (IST-2001-32747).

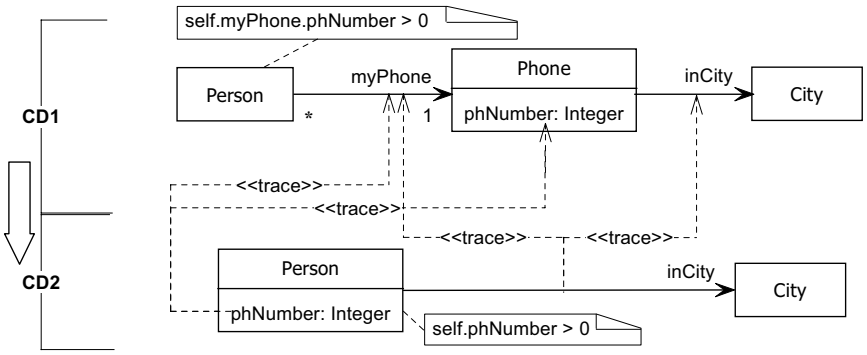


Fig. 1. An example of Inline Class refactoring pattern

Object-oriented modelling languages provide textual and diagrammatic means for system specification (like UML, cf. [OMG03]). Class diagrams specify a common structure and relationships between objects. The best known approach to redesigning the object-oriented systems is the refactoring method [Fow99]. It provides simple patterns to redesign the code and class structure in order to extend, improve and modify a system without altering its behaviour. Using our definition of *redesign class diagrams* (cf. Sec. 4) we make precise the otherwise ambiguous property of "preserving the system's behaviour".

The paper is organised as follows. In section 2 we introduce our approach by means of an example – we argue that an *Inline Class* refactoring pattern [Fow99] is indeed a redesign. Then (Sec. 3) we formally introduce a definition of a redesign diagram in an institution-independent way, describe a redesign of UML class diagrams (Sec. 4) and show (Sec. 5) an elaborate example — formal proof that an application of *Composite* design pattern [GHJV95] is a redesign. Finally we discuss related work (Sec. 6) and conclude (Sec. 7) the paper.

2 A Redesign Example – Inline Class Refactoring Pattern

In this section we present an example of *system redesign*, expressed on the level of UML class diagrams [OMG03]. The redesign is done according to the *Inline Class* refactoring pattern [Fow99]. This pattern allows us to join two classes, if one of them does not provide much functionality.

Let us consider class diagrams $CD1$ and $CD2$ shown on Fig. 1. The class Phone represents phone numbers, Person represents phone owners and City represents cities. Each person has a phone and every phone is located in a city. We attach an OCL constraint [OMG03] to the class Person, saying that every person's number is larger than 0. The class Phone does not provide much functionality and is only used by the class Person. Therefore, we join those classes in $CD2$. The class City is not affected by the redesign.

We formalise $CD1$ and $CD2$ as algebraic specifications SP_1 and SP_2, respectively. In this case the encoding is self-explanatory. More details are provided in Sec. 4.

spec SP1 =
  sorts *Person*, *Phone*, *City*;
  ops *myPhone* : *Person* → *Phone*;
    inCity : *Phone* → *City*;
    phNumber : *Phone* → *Int*
  vars *p* : *Person*
  • $phNumber(myPhone(p)) > 0$
end

spec SP2 =
  sorts *Person*, *City*;
  ops *inCity* : *Person* → *City*;
    phNumber : *Person* → *Int*
  vars *p* : *Person*
  • $phNumber(p) > 0$
end

The axioms in SP_1 and SP_2 result from the formalization of OCL constraints attached to $CD1$ and $CD2$, respectively.

We use a dependency relationship with the $\langle\langle \mathsf{trace} \rangle\rangle$ stereotype to relate elements of both diagrams. According to the convention introduced in [Kos04], classes with the same names in both diagrams are implicitly related by a $\langle\langle \mathsf{trace} \rangle\rangle$ relationship. We only need to draw the dashed arrows between attribute and link names. To represent the $\langle\langle \mathsf{trace} \rangle\rangle$ relationship at the formal level, we form a "joint" signature Σ consisting of the sum of sorts and a disjoint sum of operation symbols from Σ_1 and Σ_2 (symbols from Σ_i will be indexed with $_i$ in Σ, for $i \in \{1, 2\}$). The dependency relationship can be then translated to a set Φ of Σ-equations:

$$\forall p : Person \cdot phNumber_1(myPhone_1(p)) = phNumber_2(p)$$
$$\forall p : Person \cdot inCity_1(myPhone_1(p)) = inCity_2(p)$$

Intuitively, the first equation states that for any person p, the phone number obtained by evaluating $phNumber(myPhone(p))$ in the original system will be the same as the one obtained by evaluating $phNumber(p)$ in the redesigned system. The second equation can be interpreted in a similar way.

SP_1 and SP_2 can be translated to the joint signature Σ simply by indexing all operation symbols in axioms with $_1$ and $_2$, respectively. After putting the translations together we add the equations from Φ to obtain the specification SP which can be treated as the encoding of the whole redesign diagram shown on Fig. 1:

spec SP =
  sorts *Person*, *Phone*, *City*;
  ops $myPhone_1$: *Person* → *Phone*;
    $inCity_1, inCity_2$: *Phone* → *City*;
    $phNumber_1, phNumber_2$: *Phone* → *Int*;
  vars *p* : *Person*;
  • $phNumber_1(myPhone_1(p)) > 0$
  • $phNumber_2(p) > 0$

- $phNumber_1(myPhone_1(p)) = phNumber_2(p)$
- $inCity_1(myPhone_1(p)) = inCity_2(p)$

end

Notice that we can remove either the first or the second axiom from SP without changing the semantics of the specification, since Φ implies their equivalence.

The important property of SP is that it is a *conservative extension* of both SP_1 and SP_2. Roughly, this means that SP does not put any constraints on interpretations of $phNumber_1$, $myPhone_1$ and $inCity_1$ other than those resulting from the translation of SP_1, and similarly for the operations from Σ_2.

For the rest of this section let us adopt the usual notion of a model of a specification $\langle \Sigma, \Psi \rangle$ as a first-order many-sorted Σ-structure satisfying all sentences from Ψ. Conservativeness of SP with respect to SP_1 means that any model M of SP_1 can be extended to a model of SP by interpreting $inCity_2$ as the composition of $myPhone^M$ and $inCity^M$ and interpreting $phNumber_2$ as the composition of $myPhone^M$ and $phNumber^M$. Such an extension can be restricted to a model of SP_2 which is a "refactored" version of M. Conservativeness of SP with respect to SP_2 is equivalent to the fact that every model of SP_2 can be obtained as a restriction of some model of SP.

3 A Formal Approach

As mentioned in the introduction, the redesign has to preserve the properties of the system being restructured. To make this statement precise, we must be able to compare the *semantics* of system descriptions. In the example above this involves encoding class diagrams in some specification language, translating and putting specifications together, as well as comparing specifications that use different signatures. Such operations can be carried out in an institutional framework. In this section we formally define a redesign of specifications in an institution-independent way.

3.1 Preliminaries

Let **Set** and **Cat** denote the category of all sets and the category of all categories, respectively. An *institution* [BG92] is a tuple $\langle \mathbf{Sig}, \mathbf{Mod}, \mathbf{Sen}, \models \rangle$, where

- **Sig** is the category of *signatures*;
- $\mathbf{Mod} : \mathbf{Sig}^{op} \to \mathbf{Cat}$ is the *model functor*, assigning a category $\mathbf{Mod}(\Sigma)$ of Σ-*models* to every signature $\Sigma \in |\mathbf{Sig}|$ and a functor $\mathbf{Mod}(\sigma) : \mathbf{Mod}(\Sigma') \to \mathbf{Mod}(\Sigma)$ to every signature morphism $\sigma : \Sigma \to \Sigma'$;
- $\mathbf{Sen} : \mathbf{Sig} \to \mathbf{Set}$ is the *sentence functor* assigning a set $\mathbf{Sen}(\Sigma)$ of Σ-*sentences* to every $\Sigma \in |\mathbf{Sig}|$ and a σ-*translation* function $\mathbf{Sen}(\sigma) : \mathbf{Sen}(\Sigma) \to \mathbf{Sen}(\Sigma')$ to every $\sigma : \Sigma \to \Sigma'$;
- $\models$ is a family $\{\models_\Sigma\}_{\Sigma \in |\mathbf{Sig}|}$ of *satisfaction relations*, where $\models_\Sigma \subseteq |\mathbf{Mod}(\Sigma)| \times \mathbf{Sen}(\Sigma)$

such that for any signature morphism $\sigma : \Sigma \to \Sigma'$ the functor $\mathbf{Mod}(\Sigma)$ and the translation function $\mathbf{Sen}(\Sigma)$ preserve the satisfaction relation, that is, for any $\varphi \in \mathbf{Sen}(\Sigma)$ and $M' \in |\mathbf{Mod}(\Sigma')|$

$$\mathbf{Mod}(\sigma)(M') \models_\Sigma \varphi \quad \text{iff} \quad M' \models_{\Sigma'} \mathbf{Sen}(\sigma)(\varphi)$$

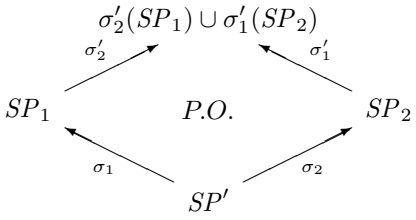
We write $M|_\sigma$ for $\mathbf{Mod}(\sigma)(M)$ and just $\sigma(\varphi)$ for $\mathbf{Sen}(\sigma)(\varphi)$.

In a fixed institution we can consider specifications as abstract objects, classified by signatures and defining classes of models. That is, we require the existence of operations *Sig* and $[\![\_]\!]$ on the class of specifications such that, for every specification SP, $Sig(SP) \in |\mathbf{Sig}|$ and $[\![SP]\!] \subseteq |\mathbf{Mod}(Sig(SP))|$. Moreover, we require that the class of specifications is closed under the following *specification-building operations* ([ST88a]):

- For any $\Sigma \in |\mathbf{Sig}|$ and $\Phi \subseteq \mathbf{Sen}(\Sigma)$, a *presentation* $\langle \Sigma, \Phi \rangle$ is a specification with $Sig(\langle \Sigma, \Phi \rangle) = \Sigma$ and $[\![\langle \Sigma, \Phi \rangle]\!] = \{M \in |\mathbf{Mod}(\Sigma)| \mid M \models \Phi\}$.
- For any signature morphism $\sigma : \Sigma \to \Sigma'$ and a specification SP such that $Sig(SP) = \Sigma$, the *translation of SP along* σ is a specification $\sigma(SP)$ such that $Sig(\sigma(SP)) = \Sigma'$ and $[\![\sigma(SP)]\!] = \{M' \in \mathbf{Mod}(\Sigma') \mid M'|_\sigma \in [\![SP]\!]\}$.
- For any specifications SP_1, SP_2 such that $Sig(SP_1) = Sig(SP_2)$, the *union* $SP_1 \cup SP_2$ is a specification with $Sig(SP_1 \cup SP_2) = Sig(SP_1)$ and $[\![SP_1 \cup SP_2]\!] = [\![SP_1]\!] \cap [\![SP_2]\!]$.
- For any signature morphism $\sigma : \Sigma \to \Sigma'$ and a specification SP' such that $Sig(SP') = \Sigma'$, the *reduct of* SP' *along* σ is a specification $SP'|_\sigma$ such that $Sig(SP'|_\sigma) = \Sigma$ and $[\![SP'|_\sigma]\!] = \{M'|_\sigma \mid M' \in [\![SP']\!]\}$.

Specifications in an arbitrary institution form a category: a *specification morphism* (Chapt. 4 in [AKKB99]) $\sigma : SP \to SP'$ is a signature morphism $\sigma : Sig(SP) \to Sig(SP')$ such that for every model $M' \in [\![SP']\!]$, $M'|_\sigma \in [\![SP]\!]$. A specification morphism $\sigma : SP \to SP'$ is *conservative* if for every $M \in [\![SP]\!]$ there exists $M' \in [\![SP']\!]$ such that $M = M'|_\sigma$. A composition of conservative morphisms is also conservative.

By Proposition 4.22 in [AKKB99] (see also [BG92]), if the category **Sig** is finitely cocomplete and the class of specifications is closed with respect to the operations listed above, then the category of specifications is also finitely cocomplete and every pushout diagram is of the form



From now on we consider only institutions with finitely cocomplete category of signatures and classes of specifications closed with respect to the specification-building operations.

Let D be a finite diagram of specifications. The functor **Mod** *preserves* a colimit of D if it maps a colimit in **Sig** of the corresponding diagram of signatures to a limit in **Cat**. We call such diagram D *an amalgamable diagram.*

3.2 An Abstract View of a Redesign

Adopting an institutional semantics encourages the formulation of concepts on the level as general as possible, hence we look for a definition of redesign independent of the logical system used. We view the structure of the system as described simply by the signature of system specification. A redesign of the specification amounts then to expressing system properties using a different signature.

Let SP_1 and SP_2 be specifications. It would be too restrictive to require the existence of a specification morphism from SP_1 to SP_2 in order to consider the latter a redesign of the former. For instance, in the example of Sec. 2 there exists no signature morphism from Σ_1 to Σ_2. Instead, we require the existence of an intermediate specification SP and two conservative morphisms $\sigma_1 : SP_1 \to SP$ and $\sigma_2 : SP_2 \to SP$. Conservativity means that the specification SP does not put any restrictions on the interpretation of symbols from $Sig(SP_1)$ and $Sig(SP_2)$ besides those already present in SP_1 and SP_2. However, SP can also relate the symbols from the two signatures, for instance define the ones from $Sig(SP_2)$ in terms of those from $Sig(SP_1)$.

Definition 1 (Redesign) *Let SP be a specification. Let $\sigma_1 : Sig(SP_1) \to Sig(SP)$ and $\sigma_2 : Sig(SP_2) \to Sig(SP)$ be signature morphisms. SP_2 is a* redesign of SP_1 via σ_1, σ_2 *if $\sigma_1 : SP_1 \to SP$ and $\sigma_2 : SP_2 \to SP$ are conservative specification morphisms. In such a case we say, that*

$$SP_1 \xrightarrow{\sigma_1} SP \xleftarrow{\sigma_2} SP_2 \tag{1}$$

is a redesign diagram.

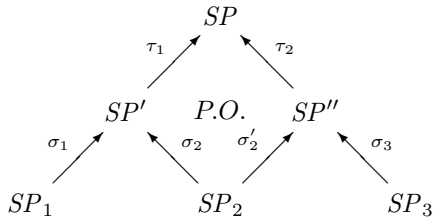
Let us observe that the specifications SP_1 and SP_2 play symmetric roles in the above definition: SP_2 is a redesign of SP_1 via σ_1, σ_2 iff SP_1 is a redesign of SP_2 via σ_2, σ_1. Moreover, if the model functor preserves coproducts, for arbitrary SP_1 and SP_2 there is a redesign diagram with the coproduct $SP_1 + SP_2$ as the intermediate specification. This redesign preserves a "minimal" behaviour, as components of the two signatures remain totally unrelated. We can require more properties to be preserved by strengthening the intermediate specification. For instance, if the invariant properties are expressed by a specification SP' with specification morphisms $\rho_1 : SP' \to SP_1$, $\rho_2 : SP' \to SP_2$, a pushout of the diagram

$$SP_1 \xleftarrow{\rho_1} SP' \xrightarrow{\rho_2} SP_2$$

can be a redesign, provided the pushout morphisms are conservative.

We show that under certain assumptions on the model functor, redesigns can be composed "vertically".

Fact 2 (Category of redesigns) *If the functor* **Mod** *preserves all pushouts then redesigns form a category in which objects are specifications and a morphism from SP_1 to SP_2 is any redesign diagram of the form* (1)*. Identity redesign diagram of specification SP with a signature Σ is $SP \xrightarrow{id_\Sigma} SP \xleftarrow{id_\Sigma} SP$. The composition of $SP_1 \xrightarrow{\sigma_1} SP' \xleftarrow{\sigma_2} SP_2$ and $SP_2 \xrightarrow{\sigma_2'} SP'' \xleftarrow{\sigma_3} SP_3$ is defined using pushout construction, as shown below.*



3.3 Redesigning Structured Specifications

Def. 1 allows us to decide whether a given diagram of specifications is a redesign. However, in a common scenario the specification SP_2 may not be known in advance: developers want to redesign a specification SP_1 to a new signature Σ_2. They relate the symbols of Σ_2 to those of $\Sigma_1 = Sig(SP_1)$ by means of a specification SP_{rel} over a "joint" signature Σ with signature morphisms $\sigma_1 : \Sigma_1 \to \Sigma$, $\sigma_2 : \Sigma_2 \to \Sigma$, such that $\sigma_1 : SP_1 \to \sigma_1(SP_1) \cup SP_{rel}$ is a conservative specification morphism. The problem now is to find a redesigned specification SP_2 over Σ_2, such that $\sigma_2 : SP_2 \to \sigma_1(SP_1) \cup SP_{rel}$ is a conservative specification morphism. Such a specification always exists, since we assumed that the class of specifications is closed with respect to reducts along signature morphisms.

Fact 3 *Let $\sigma_1 : SP_1 \to SP$ be a conservative specification morphism and let $\sigma_2 : \Sigma_2 \to Sig(SP)$ be a signature morphism. Then*

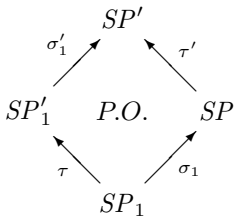
$$SP_1 \xrightarrow{\sigma_1} SP \xleftarrow{\sigma_2} SP|_{\sigma_2}$$

is a redesign diagram.

However, in many applications it is preferable to obtain SP_2 as a presentation. For instance, if SP_1 is a finite presentation resulting from encoding an UML class diagram with OCL constraints, we would also like SP_2 to consist of a list of axioms corresponding to OCL constraints for the redesigned specification. If SP_1 is obtained by application of specification-building operations, then we can use the next lemma to find SP_2 by following, to some extent, the structure of SP_1.

Lemma 4 *Let $SP_1 \xrightarrow{\sigma_1} SP \xleftarrow{\sigma_2} SP_2$ be a redesign diagram.*

i. (translation) Let $\tau : SP_1 \to SP'_1$ be a conservative specification morphism. If the following pushout diagram is amalgamable



then

$$SP'_1 \xrightarrow{\sigma'_1} SP' \xleftarrow{\sigma_2;\tau'} SP_2$$

is a redesign diagram.

ii. (union) Let SP'_1 and SP'_2 be specifications such that $Sig(SP'_1) = Sig(SP_1)$ and $Sig(SP'_2) = Sig(SP_2)$ and also

$$[\![SP]\!] \cap [\![\sigma_1(SP'_1)]\!] = [\![SP]\!] \cap [\![\sigma_2(SP'_2)]\!]$$

Then

$$SP_1 \cup SP'_1 \xrightarrow{\sigma_1} SP \cup \sigma_1(SP'_1) \xleftarrow{\sigma_2} SP_2 \cup SP'_2$$

is a redesign diagram.

iii. (coproduct) Let

$$SP'_1 \xrightarrow{\sigma'_1} SP' \xleftarrow{\sigma'_2} SP'_2$$

be a redesign diagram. If the following three coproduct diagrams are amalgamable

$$SP_1 \to SP_1 + SP'_1 \leftarrow SP'_1$$
$$SP \to SP + SP' \leftarrow SP'$$
$$SP_2 \to SP_2 + SP'_2 \leftarrow SP'_2$$

then

$$SP_1 + SP'_1 \xrightarrow{\rho_1} SP + SP' \xleftarrow{\rho_2} SP_2 + SP'_2$$

is also a redesign diagram, where ρ_1 and ρ_2 are universal morphisms from coproducts $SP_1 + SP'_1$ and $SP_2 + SP'_2$, respectively.

iv. (reduct) Let $\rho : \Sigma' \to Sig(SP_1)$ be a signature morphism. Then

$$SP_1|_\rho \xrightarrow{\rho;\sigma_1} SP \xleftarrow{\sigma_2} SP_2$$

is a redesign diagram.

The property (i) allows us to translate either the original or the redesigned specification via a conservative specification morphism and obtain a redesign diagram. This covers situations such as renaming symbols in a signature via an

injective signature morphism or extending the specification with new symbols and axioms concerning only the new symbols. By (ii), both SP_1 and SP_2 can be enriched as long as the enriching parts "correspond to each other modulo SP". The property (iii) states that if the original specification is a disjoint sum of two components, each of the components can be redesigned separately in order to obtain a redesign of the sum. Finally, by (iv), either the original or the redesigned specification can be reduced along an arbitrary signature morphism.

The above lemma can be applied to the example of Sec. 2. By the property (ii) in order to conclude that the diagram in Fig. 1 describes a redesign it suffices to show that

$$\langle \Sigma_1, \emptyset \rangle \xrightarrow{\sigma_1} \langle \Sigma, \Phi \rangle \xleftarrow{\sigma_2} \langle \Sigma_2, \emptyset \rangle$$

is a redesign diagram, and then to check that

$$phNumber_1(myPhone_1(p)) > 0 \iff phNumber_2(p) > 0$$

follows from Φ.

3.4 Interpretation Functions

Let us present a partial solution to the problem of finding Σ_2-sentences equivalent to given Σ_1-sentences for the case of institutions with sentences containing term equalities. *Interpretation functions* [Kos04] (see also [Kos01]) can be very useful as a vehicle for an automatic transformation of OCL constraints when changes to class diagrams are performed.

An interpretation function is a partial function generated by term mappings with ortogonal domain [Kos04]. These functions have several useful properties. They preserve equational proofs, proofs using propositional tautologies, resolution rule and proofs by induction [Kos04]. From our perspective, the following property is the most important: Let the redesign diagram like the one in Def. 1 be given. If the interpretation function $f : \mathbf{Sen}(\Sigma_1) \to ? \mathbf{Sen}(\Sigma_2)$ translates a Σ_1-sentence ϕ_1 to a Σ_2-sentence ϕ_2, then we have

$$[\![SP \cup \sigma_1(\phi_1)]\!] = [\![SP \cup \sigma_2(\phi_2)]\!]$$

Lemma 4, (ii), can then be applied in order to add ϕ_1 to SP_1 and ϕ_2 to SP_2.

We use an interpretation function to automatically transform the OCL constraint in the example in Sec. 5.

4 Redesign of UML Class Diagrams

We apply the notions developed in previous sections for reasoning about transformations of UML class diagrams. The idea is to formalise such diagrams as specifications in the institution of CASL ([CoF04]) — a variant of order sorted, partial first-order logic — and then to generate an intermediate specification from dependency relationships between elements of the diagrams. Note that since the category of CASL signatures is finitely cocomplete such intermediate specification always exists. A transformation preserves essential system properties if it gives raise to a redesign diagram in the category of CASL specifications.

4.1 Formalising UML Class Diagrams

We represent UML class diagrams annotated with OCL constraints (cf. [OMG03]) as algebraic specifications in CASL institution following [BHTW99]. The only difference is that in order to make the presentation more readable we omit the concept of *environements* used there to represent methods with side-effects.

For a class diagram CD we create a specification SP. In the corresponding signature Σ each sort name corresponds to a class name from CD. These sorts represent collections of objects of that class.

Class inheritance is handled by the ordering on corresponding sorts. Carriers of abstract classes are disjoint unions (up to an isomorphism) of carriers of all its direct subclasses. We additionally require (it wasn't required in [BHTW99]) that in every specification there is an appropriate axiom guaranteeing that.

As we are aware that this kind of encoding does not permit to express overriding we refer the reader to e.g. [ACZ99] for a possible solutions of this problem (see also [Mar04]).

All query methods (these that do not change system state) and attributes are encoded as functions with additional first parameter representing *self* object. We claim that all problems related to side effects, local object states, global system environement etc. are orthogonal to the problems described herein. Thus we do not care about encoding of any non-query methods. For the similar reason we assume that the only OCL constraints contained in class diagrams are class invariants (i.e. there are no method pre- and postconditions). Translation of these OCL constraints to CASL logic is straightforward using the method described in [BHTW99]. As in [Kos04] we call the function that takes an OCL annotated UML class diagram and produces a specification in CASL institution a *translation function Trans*. Formally, since the whole class diagram can be described as an OCL sentence, *Trans* is a mapping of OCL terms to CASL formulas. Specifications SP_1 and SP_2 in Sec. 2 are example results of this translation.

In what follows we only consider diagrams consisting of two class diagrams, say $CD1$ and $CD2$, with trace relationships (dependencies marked with the $\langle\langle$trace$\rangle\rangle$ stereotype) connecting corresponding components in both of them (Fig. 1 contains an example of such a diagram). Let us call such diagrams *class diagrams with traces*.

All UML trace dependencies have a mappingExpression attribute used to capture the relationship between elements linked by trace dependencies. In our examples only two values of mappingExpression are used — *composition* and *product*. For instance a trace dependency linking myPhone and inCity in $CD1$ with inCity in $CD2$ on Fig. 1 has mappingExpression set to *composition*, which means that inCity in $CD2$ is a composition of myPhone and inCity in $CD1$. In cases when the intended relationship is obvious, mappingExpression may be omitted from the class diagram.

4.2 Redesign Class Diagrams

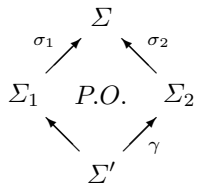
Given an UML class diagram with traces CDT that consists of two class diagrams *CD1* and *CD2* such that the *CD2* is some transformation of *CD1* we

would like to decide whether the transformation is a redesign. We assume that *CD2* is already annotated with OCL constraints e.g. by use of some interpretation function (cf. Sec. 3.4). We use the translation *Trans* (described in Sec. 4.1) to represent *CD1* as a specification SP_1 with a signature Σ_1, and *CD2* as a specification SP_2 with a signature Σ_2.

Let X be a countable set of variables. Let us use traces connecting classes on CDT to generate a partial *sort mapping* $sm : Sorts(\Sigma_1) \rightarrow ? \; Sorts(\Sigma_2)$. We require that sm preserve the subsort relation and be injective (for the reasons described below). Similarily, using traces connecting methods and/or attributes and using the deriviation strategy described by mappingExpression of each trace, we define a partial many sorted *term mapping* $tm : T_{\Sigma_1}(X) \rightarrow ? \; T_{\Sigma_2}(X)$ such that sm is a sort mapping associated with tm (i.e. sm and tm coincide on sorts). The way of translation of trace endpoints to terms is straightforward (e.g. the composition of myPhone and phNumber is translated to a term $phNumber(myPhone(x))$ where $x \in Person$ as in Sec. 2)

The term mapping tm forms a connection between terms over two different signatures. We require the denotations of corresponding terms to be equal. In general it is impossible to express such property as a sentence from either $\mathbf{Sen}(\Sigma_1)$ or $\mathbf{Sen}(\Sigma_2)$. Thus we construct a bigger signature Σ containing all that is needed to express the equality of terms being mapped by tm.

Let signature Σ_{rts} describe the "rest of the system" i.e. translation of all classes, attributes, methods etc. depicted neither in $CD1$ nor in $CD2$. We assume that Σ_{rts} is part of both Σ_1 and Σ_2. As the sort mapping sm is functional and injective we can assume also that all sorts connected by sm are common (up to renaming) to them both. Thus let us define the signature Σ' as Σ_{rts} and additionally all sorts from $dom(sm)$ (Σ_{rts} and $dom(sm)$ are disjoint). The perfect candidate for a "joint" signature Σ is the pushout of Σ_1 and Σ_2 over the signature Σ'



where γ is a signature morphism mapping all Σ_{rts} symbols to themselves and additionally mapping all sorts in $s \in dom(sm)$ to $sm(s)$. Since we require that sm preserves the subsort relation, γ is indeed a CASL subsorted signature morphism. Having Σ we can express the desired equality on terms. We define the following set of equations:

$$\Phi = \{\forall X \cdot \sigma_1(t) = \sigma_2(tm(t)) \mid t \in T_{\Sigma_1}(X), t \in dom(m)\}$$

Note that $\Phi \subseteq \mathbf{Sen}(\Sigma)$.

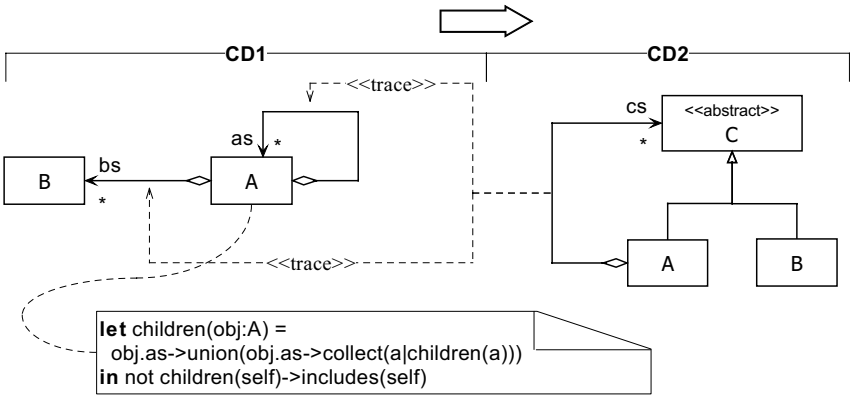
Finally, we are able to formalise the requirements that need to be imposed on a class diagram to represent a redesign.

Definition 5 (Redesign class diagram) *Using the notation introduced above, the class diagram CDT is a* redesign class diagram *over an signature Σ' if translations of $CD1$ and $CD2$ to the* CASL *institution together with the specification $\sigma_1(SP_1) \cup \langle \Sigma, \Phi \rangle$ and signature morphisms σ_1, σ_2 form the following redesign diagram (in the sense of Def. 1)*

$$SP_1 \xrightarrow{\sigma_1} \sigma_1(SP_1) \cup \langle \Sigma, \Phi \rangle \xleftarrow{\sigma_2} SP_2$$

Note that the requirement that *sm* is injective is very sensible since every non-injective sort mapping leads to a non-conservativeness of σ_1.

5 An Elaborate Redesign Example – Composite Pattern



In the above figure the class diagram $CD1$ describes *Directed Acyclic Graph (DAG)* data structure. Objects of the class A represent internal graph nodes, objects of the class B represent leaves. The OCL invariant of A guarantees that the structure is indeed a DAG (i.e. it doesn't contain a cycle).

The class diagram $CD2$ is a result of the application of the *Composite design pattern* [GHJV95] to the system described by $CD1$. Note the lack of OCL constraint describing an invariant of the class A in $CD2$. Thus $CD2$ does not necessarily describe a DAG. To fix the problem we use the interpretation function to generate the appropriate invariant (cf. Sec. 3.4 and [Kos04] for details). The following mapping

$$\begin{aligned} \texttt{x.as} &\mapsto \texttt{x.cs->collect(a | a.isKindOf(A))} \\ \texttt{x.bs} &\mapsto \texttt{x.cs->collect(b | b.isKindOf(B))} \end{aligned}$$

is orthogonal and thus it could be extended [Kos04] to the interpretation function that we use to transform the invariant of A in $CD1$

```
context A inv :
 let
  children(obj : A) = obj.as->union(obj.as->collect(a | children(a)))
 in not children(self)->includes(self)
```

to the following invariant in $CD2$

```
context A inv :
 let
  children(obj : A) = obj.cs−> collect(a | a.isKindOf(A))−> union(
      obj.cs−> collect(a | a.isKindOf(A))−> collect(a | children(a)))
 in not children(self)−> includes(self)
```

To decide whether the above class diagram with an additional invariant for A in $CD2$ is a redesign class diagram we translate $CD1$ and $CD2$ augmented with an invariant to SP_1 and SP_2, respectively.

spec SP1 =
sorts A, B
ops $as : A \to Set[A]$;
$bs : A \to Set[B]$;
$children : A \to Set[A]$;
vars $a, a' : A$
- $a' \in children(a) \Leftrightarrow$
  $(a' \in as(a) \vee$
  $\exists a'' \bullet a'' \in as(a) \wedge$
  $a' \in children(a''))$
- $a \notin children(a)$

end

spec SP2 =
sorts $A, B, C; A \leq C; B \leq C;$
ops $cs : A \to Set[C]$;
$children : A \to Set[A]$;
vars a, a': A; $c : C$
- $a' \in children(a) \Leftrightarrow$
  $(a' \in A \wedge (a' \in cs(a) \vee$
  $\exists a'' \bullet a'' \in cs(a) \wedge$
  $a' \in children(a'')))$
- $a \notin children(a)$
- $c \in A \Leftrightarrow c \notin B$

end

Note that the function *children* defined locally in the OCL invariant has been added the global function to both specifications. We know by (iv) of Lemma 4 that it is enough to prove that specifications with global *children* are mutual redesigns to conclude that their versions with *children* defined locally are redesigns as well (via the same intermediate specification).

First two axioms of both specifications are translations of invariants of A. The last axiom in $SP2$ says that C is a disjoint union of A and B (the encoding of the $\langle\langle$abstract$\rangle\rangle$ stereotype on C).

The trace relationships (the omitted mappingExpression clearly signifies that they are of *product* type, cf. Sec. 4.1) result in the following sort mapping s and term mapping m

$$s = \{A \mapsto A; \quad B \mapsto B\}$$

$$m = \{as(x) \mapsto \pi_1(i(cs(x))); \quad bs(x) \mapsto \pi_2(i(cs(x))); \quad children(x) \mapsto children(x)\}$$

where i is an obvious isomorphism between $Set[C]$ and $Set[A] \times Set[B]$ (justified by the requirement expressed by axioms in SP_2 saying that $C = A \uplus B$), π_1 and π_2 are product projections defined elsewhere.

We use the procedure described in Sec. 4.2 to define a signature Σ' common to both SP_1 and SP_2, $\Sigma' = \{A, B\} \cup \Sigma_{rts}$, where Σ_{rts} is a signature with all

standard sorts (e.g. integers, booleans, etc.) and operations on them (it was implicitly assumed to be a part of signatures of SP_1 and SP_2). Let Σ be a pushout of Σ_1 and Σ_2 over Σ'. The joint specification SP is the following:

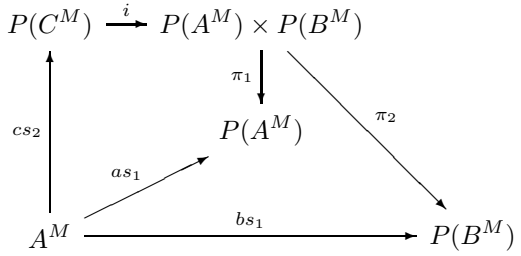
spec SP =
  sorts $A, B, C; A \leq C; B \leq C;$
  ops $as_1 : A \to Set[A];$
    $bs_1 : A \to Set[B];$
    $cs_2 : A \to Set[C];$
    $children_1, children_2 : A \to Set[A];$
  vars $a, a' : A, c : C$
  - $as_1(a) = \pi_1(i(cs_2(a)))$
  - $bs_1(a) = \pi_2(i(cs_2(a)))$
  - $children_1(a) = children_2(a)$
  - $a' \in children_1(a) \Leftrightarrow (a' \in as(a) \vee \exists a'' \bullet a'' \in as(a) \wedge a' \in children_1(a''))$
  - $a \notin children_1(a)$
  - $c \in A \Leftrightarrow c \notin B$

end

Note that the first three axioms are equalities resulting from the term mapping m (set Φ in Sec. 4.2).

It is clear that $\sigma_1 : SP_1 \to SP$ and $\sigma_2 : SP_2 \to SP$ are specification morphisms. In the following we show that they are conservative.

First we prove it for σ_1. Let us assume $M_1 \in [\![SP_1]\!]$. We need to find a model $M \in [\![SP]\!]$ such that $M \mid_{\sigma_1} = M_1$. M can be constructed by putting $A^M = A^{M_1}$, $B^M = B^{M_1}$, $as_1^M = as^{M_1}$, similarly bs_1^M and $children_1^M$. Let $C^M = A^M \uplus B^M$. The function $cs_2 : A^M \to P(C^M)$ is the only function that causes the commutativity of the following diagram in **Set** (by $P(S)$ we mean set of all finite subsets of S – i.e. the CASL sort $Set[S]$).



The existence of cs_2 follows from the universal property of the product $P(A^M) \times P(B^M)$. Obviously $M = M \mid_{\sigma_1}$.

To show that σ_2 is conservative let us assume $M_2 \in [\![SP_2]\!]$. We need to find such $M \in [\![SP]\!]$ that $M \mid_{\sigma_2} = M_2$. As C is an abstract class i.e. $C^{M_2} = A^{M_2} \uplus B^{M_2}$, we construct M as M_2 and additionally interpret two functions $as_1 : A^M \to P(A^M)$ and $bs_1 : A^M \to P(B^M)$ as compositions of $cs_2; i$ with π_1 and π_2 respectively. Again is easy to see that $M_2 = M \mid_{\sigma_2}$.

5.1 Changing a Redesign Diagram

Imagine the situation that just after we have proved that the above class diagram is a redesign class diagram we discovered that actually the names of classes on $CD2$ were written incorrectly. They should have been (a it is in [GHJV95]) Component instead of C, Composite instead of A and Leaf instead of B and also there should have been additional method name in a class Component. We do not need to redo a proof that a transformation is a redesign. Since all above described changes are conservative, we can use[4] property (i) of Lemma 4 that any translation of SP_2 by a conservative specification morphism leads to a specification that is also a redesign of SP_1 (via the same intermediate specification).

6 Related Work

A number of approaches to redesigning UML class models exist already. The best known is the *refactoring* [Fow99]. This approach provides simple patterns for code and class structure redesign to extend, improve and modify a system without altering its behavior. Model transformation gain a lot of interest in recent time (see [MCG03]). Interpretation functions, used to formalise UML class digram transformations in [Kos01,Kos04] originate in abstract algebra. In [Tay73], an interpretation function transforms a single operation symbol into a complex term. Lano's approach [Lan95] uses a form of interpretation function and some axiomatic/equational extension of theories to define the notion of refinement for the Real Time Action Logic; it is based on the Object Calculus [FM91]. Graph rewriting systems may be used to describe transformation of a specification (cf. e.g. [GRPPS98]).

Our notion of redesign is related to the concept of *implementation* ([ONS96,ST88b]). The implementation of SP_1 by SP_2, as defined in [ONS96], is also parametrised by an intermediate specification. The main difference between the two notions is the constructive nature of the implementation: a *constructor* operation must be provided that transforms every model of SP_2 to a model of the intermediate specification that can be then reduced to the model of SP_1. A redesign diagram $SP_1 \xrightarrow{\sigma_1} SP \xleftarrow{\sigma_2} SP_2$ does not prescribe how to provide such operations, it merely guarantees that any *persistent constructors* ([ST88b]) from $[\![SP_2]\!]$ to $[\![SP]\!]$ would implement SP_1 by SP_2.

7 Conclusion and Future Work

In our paper we have defined a formal notion of the redesign of specifications. Our approach allows one to reason about such transformation of the system structure that are incomparable by means of a signature morphism. We have also shown (see Lemma 4) that, under certain assumptions, a structured specification can be redesigned in a step-by-step manner, to a specification structured similary to the initial one.

[4] As it is easy to prove that the resulting pushout diagram is amalgamable in CASL

As a practical application of our work we have presented a formalisation of the redesign of UML class diagrams. We have defined the conditions that need to be imposed on a class diagram for it to describe a property preserving redesign.

Translating of UML class diagrams to CASL we cared about query methods only. Moreover we did not handle method overriding in the subclasses. We plan to investigate these issues in coming future.

References

[ACZ99] D. Ancona, M. Cerioli, and E. Zucca. A formal framework with late binding. In *Proceedings of the 2nd Int. Conf. on Fundamental Approaches to Software Engineering*. Springer-Verlag, 1999.

[AKKB99] E. Astesiano, H.-J. Kreowski, and B. Krieg-Brukner, editors. *Algebraic Foundations of Systems Specification*. IFIP State-of-the-Art Report. 1999.

[BG92] R.M. Burstall and J.A. Goguen. Institutions: Abstract model theory for specification and programming. *Journ. of the ACM*, 39(1), 1992.

[BHTW99] M. Bidoit, R. Hennicker, F. Tort, and M. Wirsing. Correct realization of interface constraints with OCL. In R. France and B. Rumpe, editors, *UML'99 - The UML. Beyond the Standard.*, volume 1723. Springer, 1999.

[CoF04] CoFI. CASL *Reference Manual*. 2960 (IFIP Series). 2004.

[FM91] J. Fiadeiro and T. Maibaum. Describing, structuring and implementing objects. In J. W. de Bakker, W. P. de Roever, and G. Rozenberg, editors, *Foundations of Object-Oriented Languages*. Springer, 1991.

[Fow99] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-OrientedSoftware*. Addison-Wesley, 1995.

[GRPPS98] M. Große-Rhode, F. Parisi-Presicce, and M. Simeoni. Refinements and modules for typed graph transformation systems. In *WADT*, 1998.

[Kos01] P. Kosiuczenko. Formal redesign of UML class diagrams. In A. Evans, R. France, A. Moreira, and B. Rumpe, editors, *Proc. of UML Workshop on Prac. UML Based Rigorous Dev. Meth.*, LN in Informatics. 2001.

[Kos04] P. Kosiuczenko. Redesign of UML class diagrams: a formal approach. Technical report, submitted for publication, 2004.

[Lan95] K. Lano. *Formal Object-Oriented Development*. Springer-Verlag, 1995.

[Mar04] G. Marczyński. Specifications of internally dependent structures. Technical report, In preparation, 2004.

[MCG03] T. Mens, K. Czarnecki, and P. Van Gorp. A taxonomy of model transformations. *LNCS*, 2003.

[OMG03] OMG. *Unified Modeling Language*, 1.5 edition, 2003.

[ONS96] F. Orejas, M. Navarro, and A. Sánchez. Algebraic implementation of abstract data types: a survey of concepts and new compositionality results. *Mathematical Structures in Computer Science*, 6(1):33–67, 1996.

[ST88a] D. Sannella and A. Tarlecki. Specifications in an arbitrary institution. *Information and Computation*, 76:165–210, 1988.

[ST88b] D. Sannella and A. Tarlecki. Toward formal development of programs from algebraic specifications: Implementations revisited. *Acta Informatica*, 25(3):233–281, 1988.

[Tay73] W. Taylor. Characterizing Malcev conditions. *Algebra Universalis*, 3:351–397, 1973. Springer, Berlin.