

A Heterogeneous Approach to UML Semantics^{*}

María Victoria Cengarle¹, Alexander Knapp², Andrzej Tarlecki³, and Martin Wirsing²

¹ Technische Universität München
cengarle@in.tum.de

² Ludwig-Maximilians-Universität München
{knapp,wirsing}@pst.ifi.lmu.de

³ Uniwersytet Warszawski
tarlecki@mimuw.edu.pl

Abstract. UML models consist of several diagrams of different types describing different views of a software system ranging from specifications of the static system structure to descriptions of system snapshots and dynamic behaviour. In this paper a heterogeneous approach to the semantics of UML is proposed where each diagram type can be described in its “natural” semantics, and the relations between diagram types are expressed by appropriate translations. More formally, the UML family of diagram types is represented as a “heterogeneous institution environment”: each diagram type is described as an appropriate institution where typically the data structures occurring in a diagram are represented by signature elements whereas the relationships between data and the dynamic behaviour of objects are captured by sentences; in several cases, the diagrams are themselves the sentences. The relationship between two diagram types is described by a so-called institution comorphism, and in case no institution comorphism exists, by a co-span of such comorphisms. Consistency conditions between different diagrams are derived from the comorphism translations. This heterogeneous semantic approach to UML is illustrated by several example diagram types including class diagrams, OCL, and interaction diagrams.

1 Introduction

Almost exactly 40 years ago, computer programs had become so large and complex that many software development projects failed and maintaining software programs was almost unmanageable. Neither pragmatic methods for software construction nor the scientific foundations of programming were established at the time; the need for systematic software development techniques was so urgent and evident that in 1968 the first software engineering conference was organized in Garmisch [39].

Today, software systems are larger than ever, and software is the central innovating factor of many high-tech products, services, and systems, e.g. in consumer electronics, automotive applications, telecommunications, and business. Our daily life and work depend more and more on such software-intensive systems. Driven by techniques such as

^{*} This work has been partially sponsored by the project SENSORIA IST-2005-016004, and the DFG projects InfoZert, MAEWA and rUML.

object-orientation, service-orientation, or model-transformation, practical software engineering methods have considerably evolved and many companies follow well-defined software development processes for constructing larger and larger software systems. A substantial body of theoretical foundations of programming is available and formal modelling and analysis techniques like abstraction and refinement techniques, model checking or theorem proving have undergone a steep development during the last years. However, at the same time we experience that many software systems are error-prone, unstable, have security holes, and do not meet the required quality standards. There is still a large gap between industrial practice and formal approaches: pragmatic modelling languages and techniques lack formal foundations, inhibiting the development of powerful analysis and development tools, while formal approaches are often too difficult to use, do not scale easily to complex software-intensive systems, and are not well-integrated with pragmatic methods. Aspects such as distribution, mobility, heterogeneity, quality of service, security, trust, and dynamically changing infrastructures and environments are not well supported by actual engineering methods.

Bridging this gap and advancing software engineering theory and methods is one of the main aims of our common research with Ugo Montanari during the last several years. In the project AGILE [1] we developed an architectural approach to software mobility which was based on a uniform mathematical framework to support sound methodological principles, formal analysis, and refinement. The aim of SENSORIA [48,50] is to develop a novel comprehensive approach to the engineering of service-oriented software systems where foundational theories, techniques and methods are fully integrated in a pragmatic software development process. With SENSORIA techniques software engineers can model a system in the usual way by using standard high-level visual modelling languages such as UML; but they get additional help for reasoning about functional and non-functional properties of the system by mathematical models which run in the background hidden from the developer.

As a prerequisite for the SENSORIA approach we investigate in this paper the semantics of UML. Models expressed in this language consist of several diagrams of different types describing different views of a software system ranging from specifications of the static system structure to descriptions of system snapshots and dynamic behaviour. For example, UML 1.x offers nine different types of diagrams for describing different static and dynamic aspects of a system. Depending on the modelling purpose one may employ e.g. class diagrams, component diagrams, state diagrams, sequence diagrams, activity diagrams, or instance diagrams. UML 2.0 adds several new diagram types and enhances the expressiveness and semantics of sequence diagrams and activity diagrams considerably.

We propose here a new “heterogeneous approach” to the semantics of UML which concentrates on the comparison and integration of different modelling formalisms, and in which

- each diagram type can be described in its “natural” semantics,
- relations between diagram types are expressed by appropriate translations, and
- consistency conditions can be derived between diagrams of different type.

More formally, we present the UML family of diagram types as a “heterogeneous institution environment”. An institution is given by a type of signatures, a type of sen-

tences, a notion of model, and a notion of validity of a sentence in a model. Each diagram type is described as an appropriate institution, and each diagram instance is a specification in the institution. Typically, the data structures occurring in a diagram are named by signature elements whereas the dynamic behaviour of objects and relationships between data are described by sentences; in several cases, the diagrams are themselves the sentences. The semantic concepts involved are captured by the models considered, with the validation between sentences and models determining the semantics of particular diagrams. The relationship between two diagram types is described by so-called institution comorphisms, and in case no institution comorphism exists, by a co-span of such comorphisms. Consistency conditions between different diagrams are derived from the comorphism translations or the co-span construction. We illustrate the heterogeneous semantic approach to UML by several diagram types including class diagrams, OCL, and interaction diagrams; we demonstrate the use of heterogeneous UML institutions by means of the e-learning case study of the SENSORIA project [50].

The remainder of this paper is structured as follows: In Sect. 2 we review the related work to the semantics on modelling languages and, in particular, on UML semantics. Sect. 3 introduces the theory of heterogeneous institution environments. In Sect. 4 we present sketches of the institutions for class diagrams, OCL, and interaction diagrams and in Sect. 5 we sketch how these different languages can be linked by institution comorphisms and co-spans. The implications for consistency conditions between diagrams of these different types are discussed in Sect. 6. Finally, in Sect. 7 we conclude with a short discussion of the results of the paper and an outlook on further research topics.

2 Related Work

Giving an integrated semantics to UML is a difficult task due to the complexity and variety of the different diagram types. The classical approaches to programming language semantics are adequate only for a restricted subset of the specification and modelling tasks of software development. For example, denotational semantics is an elegant framework for compositionally modelling functional behaviour but it is not so appropriate for the dynamic behaviour of concurrent processes; in contrast, structural operational semantics (SOS) is well-suited for the latter but treats data structures only in a syntactic way. Algebraic specifications are appropriate for modelling complex data types and associated operations, but they are not easily usable for specifying reactive systems, in spite of the elegant work started with [2] on axiomatic process algebra.

In order to model static functional aspects as well as dynamic concurrent behaviour, several researchers investigated extensions or combinations of these methods. Brody proposes a denotational “system model” based on stream-processing functions in combination with abstract data types (see e.g. [8,9]) and currently uses this system model for developing a complete UML semantics [5,6,7], where diagrams are taken as predicates that a system model instance has to satisfy. Other approaches propose the combination of CSP and Z [23,47] or a combination of algebraic specifications and labelled transition systems [42]. Rewriting logic [34] is a semantic framework for concurrency which extends the algebraic specification approach to concurrent systems.

Ugo Montanari’s tile model [24] is a system model for describing the behaviour of open systems; it is a SOS-like compositional framework where data structures are not to be restricted to syntactic terms, and it can be seen as an extension of rewriting logic by taking into account state changes with side effects and synchronisation [35]. Architectural Design Rewriting [10] is another novel elegant approach of Ugo Montanari where rewriting techniques are integrated with graph transformations in order to support the design of reconfigurable software architectures.

Our framework is inspired by these combination approaches; but instead of a tight integration of different modelling techniques we aim at a loose coupling and use a “heterogeneous specification approach” which concentrates on the comparison and integration of different specification formalisms, retaining the formalisms most appropriate for expressing parts of the overall problem.

For UML, this line of research started with our algebraic viewpoint approach [49] and the general categorical setting of Ehrig, Orejas, and Padberg (see e.g. [21]). In other contexts, institutions [27] and general logics [33] have been proposed as a formal basis establishing a powerful framework for heterogeneous specifications and heterogeneous proofs [3,45,36,37]. In particular, Goguen uses a heterogeneous institutional framework for database schema integration [25]. Our approach in this paper is particularly inspired by the KORSO development graph [40] and its subsequent formalisation by Mossakowski [37,38] in the heterogeneous institution setting.

3 Heterogeneous Institution Environments

To cope with the multitude of different views of software systems as captured in UML by various diagram types we need to formally define what logical systems are (each corresponding to a different diagram type) and how they may be related (to provide precise semantic links between UML diagrams of different types). The theory of institutions, started by Goguen and Burstall [26,27] and then developed in a number of directions, from an abstract theory of software specification and development [46] to a very general version of abstract model theory [20], offers a suitable formal framework.

The usual presentation of the theory of institutions depends on category theory [31]. However, to follow the presentation here not much more is needed than some intuitive understanding of the basics: a category \mathbf{K} consists of a collection $|\mathbf{K}|$ of objects and morphisms between them (including identity morphisms) that can be composed in a natural way; a functor $\mathbf{F} : \mathbf{K} \rightarrow \mathbf{K}'$ between categories maps objects to objects and morphisms to morphisms preserving their source and target, identities and composition; and a natural transformation between “parallel” functors $\mathbf{F}, \mathbf{F}' : \mathbf{K} \rightarrow \mathbf{K}'$ consists of a family of morphisms in \mathbf{K}' that link the functor values on each object in $|\mathbf{K}|$ and change smoothly w.r.t. the functor values on each morphism in \mathbf{K} .

An *institution* \mathcal{I} consists of a category $\mathbf{Sign}_{\mathcal{I}}$ of *signatures* describing its language symbols; a functor $\mathbf{Sen}_{\mathcal{I}} : \mathbf{Sign}_{\mathcal{I}} \rightarrow \mathbf{Set}$,¹ describing its language in the form of *sentences*; a functor $\mathbf{Mod}_{\mathcal{I}} : \mathbf{Sign}_{\mathcal{I}}^{op} \rightarrow \mathbf{Set}$,² describing its *models*; and

¹ The category \mathbf{Set} has all sets as objects and all functions between them as morphisms.

² To keep things simple, we work with the version of institutions where morphisms between models, not needed here, are disregarded. To capture standard examples, we should allow here

for $\Sigma \in |\mathbf{Sign}_{\mathcal{I}}|$ a *satisfaction relation* $\models_{\mathcal{I},\Sigma} \subseteq \mathbf{Mod}_{\mathcal{I}}(\Sigma) \times \mathbf{Sen}_{\mathcal{I}}(\Sigma)$ describing which sentences are satisfied (hold) in which models. Expanding the above, the sentence functor $\mathbf{Sen}_{\mathcal{I}}$ yields a set $\mathbf{Sen}_{\mathcal{I}}(\Sigma)$ of Σ -sentences for each signature $\Sigma \in |\mathbf{Sign}_{\mathcal{I}}|$, and a function $\mathbf{Sen}_{\mathcal{I}}(\sigma) : \mathbf{Sen}_{\mathcal{I}}(\Sigma) \rightarrow \mathbf{Sen}_{\mathcal{I}}(\Sigma')$, denoted simply by σ , that represents the σ -translation of Σ -sentences to Σ' -sentences for each signature morphism $\sigma : \Sigma \rightarrow \Sigma'$. The model functor $\mathbf{Mod}_{\mathcal{I}}$ gives a set $\mathbf{Mod}_{\mathcal{I}}(\Sigma)$ of Σ -models for each signature $\Sigma \in |\mathbf{Sign}_{\mathcal{I}}|$, and a function $\mathbf{Mod}_{\mathcal{I}}(\sigma) : \mathbf{Mod}(\Sigma') \rightarrow \mathbf{Mod}(\Sigma)$, denoted by $_|\sigma$, that yields σ -reducts of Σ' -models for each signature morphism $\sigma : \Sigma \rightarrow \Sigma'$. The satisfaction relations have to satisfy the following *satisfaction condition* for all $\Sigma, \Sigma' \in |\mathbf{Sign}_{\mathcal{I}}|$, signature morphisms $\sigma : \Sigma \rightarrow \Sigma'$, Σ -sentences $\varphi \in \mathbf{Sen}_{\mathcal{I}}(\Sigma)$ and Σ' -models $M' \in \mathbf{Mod}_{\mathcal{I}}(\Sigma')$:

$$M' \models_{\mathcal{I},\Sigma'} \sigma(\varphi) \iff M'|\sigma \models_{\mathcal{I},\Sigma} \varphi.$$

We typically omit the subscript \mathcal{I} when referring to the components of an institution \mathcal{I} , and the subscript Σ on the satisfaction relations. For any signature Σ , the satisfaction relation extends naturally to sets of Σ -sentences and classes of Σ -models. Moreover, it determines the usual consequence relation: a Σ -sentence $\varphi \in \mathbf{Sen}(\Sigma)$ is a (*semantic consequence*) of a set $\Phi \subseteq \mathbf{Sen}(\Sigma)$ of Σ -sentences, written $\Phi \models \varphi$, if for all Σ -models $M \in \mathbf{Mod}(\Sigma)$, $M \models \Phi$ implies $M \models \varphi$.

The notion of institution is quite general, as it imposes only very mild requirements on the logical system. Apart from the implicit structural assumptions (like functoriality of sentence translations and model reducts) the key requirement is the satisfaction condition. Informally, it asserts that logical satisfaction is invariant under the change of signature, and so does not depend on the context of use of a sentence. This property may fail for some logical systems (for instance, when some version of “closed world assumption” is used). Nevertheless, typically the satisfaction of a sentence depends only on semantic interpretation of the symbols it actually involves, and the satisfaction condition then holds. Consequently, examples of institutions abound, and include standard logical systems like equational, first-order and higher-order logics, various modal logics, logics of partial functions, etc. We refrain from spelling out any examples for now, with examples of institutions capturing various UML diagram types to be presented below.

Given the definition of an institution to capture the informal notion of a logical system, we can make precise various ways in which logical systems can be related. The starting point was the definition of an *institution morphism* in [26,27]. Other notions followed, capturing different intuitions and various aspects of relating one logical system to another. We will use here institution comorphisms (named so in [29]; see “plain maps of institutions” in [33] and “institution representations” in [43,44]). Very informally, an institution comorphism $\rho : \mathcal{I} \rightarrow \mathcal{I}'$ captures how a weaker and poorer institution \mathcal{I} can be represented in a stronger and richer institution \mathcal{I}' , by representing \mathcal{I} -signatures as \mathcal{I}' -signatures and \mathcal{I} -sentences as \mathcal{I}' -sentences, and extracting \mathcal{I} -models from \mathcal{I}' -models.

for the use of classes, rather than just sets of models — but again, we will disregard such foundational subtleties here.

More precisely, for arbitrary institutions $\mathcal{I} = \langle \mathbf{Sign}, \mathbf{Sen}, \mathbf{Mod}, \langle \models_{\Sigma} \rangle_{\Sigma \in |\mathbf{Sign}|} \rangle$ and $\mathcal{I}' = \langle \mathbf{Sign}', \mathbf{Sen}', \mathbf{Mod}', \langle \models_{\Sigma'} \rangle_{\Sigma' \in |\mathbf{Sign}'|} \rangle$, an *institution comorphism* $\rho : \mathcal{I} \rightarrow \mathcal{I}'$ consists of a functor $\rho^{Sign} : \mathbf{Sign} \rightarrow \mathbf{Sign}'$; a natural transformation $\rho^{Sen} : \mathbf{Sen} \rightarrow \rho^{Sign} ; \mathbf{Sen}'$; and a natural transformation $\rho^{Mod} : (\rho^{Sign})^{op} ; \mathbf{Mod}' \rightarrow \mathbf{Mod}$, such that for any signature $\Sigma \in |\mathbf{Sign}|$ the translations $\rho_{\Sigma}^{Sen} : \mathbf{Sen}(\Sigma) \rightarrow \mathbf{Sen}'(\rho^{Sign}(\Sigma))$ of sentences and $\rho_{\Sigma}^{Mod} : \mathbf{Mod}'(\rho^{Sign}(\Sigma)) \rightarrow \mathbf{Mod}(\Sigma)$ of models preserve the satisfaction relation, that is, for any $\varphi \in \mathbf{Sen}(\Sigma)$ and $M' \in \mathbf{Mod}'(\rho^{Sign}(\Sigma))$:

$$M' \models'_{\rho^{Sign}(\Sigma)} \rho_{\Sigma}^{Sen}(\varphi) \iff \rho_{\Sigma}^{Mod}(M') \models_{\Sigma} \varphi .$$

The naturality requirements amount to the facts that ρ^{Sen} and ρ^{Mod} are families of functions $\rho_{\Sigma}^{Sen} : \mathbf{Sen}(\Sigma) \rightarrow \mathbf{Sen}'(\rho^{Sign}(\Sigma))$ and $\rho_{\Sigma}^{Mod} : \mathbf{Mod}'(\rho^{Sign}(\Sigma)) \rightarrow \mathbf{Mod}(\Sigma)$, respectively, such that for $\sigma : \Sigma \rightarrow \Sigma'$ the diagrams in Fig. 1 commute.

$$\begin{array}{ccc} \mathbf{Sen}(\Sigma_2) & \xrightarrow{\rho_{\Sigma_2}^{Sen}} & \mathbf{Sen}'(\rho^{Sign}(\Sigma_2)) \\ \uparrow \sigma & & \uparrow \rho^{Sign}(\sigma) \\ \mathbf{Sen}(\Sigma_1) & \xrightarrow{\rho_{\Sigma_1}^{Sen}} & \mathbf{Sen}'(\rho^{Sign}(\Sigma_1)) \end{array} \quad \begin{array}{ccc} \mathbf{Mod}(\Sigma_2) & \xleftarrow{\rho_{\Sigma_2}^{Mod}} & \mathbf{Mod}'(\rho^{Sign}(\Sigma_2)) \\ \downarrow \neg \sigma & & \downarrow \neg \rho^{Sign}(\sigma) \\ \mathbf{Mod}(\Sigma_1) & \xleftarrow{\rho_{\Sigma_1}^{Mod}} & \mathbf{Mod}'(\rho^{Sign}(\Sigma_1)) \end{array}$$

Fig. 1. Naturality diagrams for an institution comorphism

The original notion of institution morphism [27] essentially differs from the above only in the direction of translation of models and sentences w.r.t. translation of signatures: an institution morphism from \mathcal{I} to \mathcal{I}' maps \mathcal{I} -signatures to \mathcal{I}' -signatures, \mathcal{I} -models to \mathcal{I}' -models, and \mathcal{I}' -sentences to \mathcal{I} -sentences, capturing quite a different intuition though (how a richer institution \mathcal{I} is built over the poorer institution \mathcal{I}'). Other possible notions of a mapping between institutions can be obtained by a similar manipulation of mutual directions of the translations involved [44], to capture yet different intuitions. It turns out, however, that all such variations can be expressed using institution comorphisms, albeit in general we may need a *span* of those. Namely, to relate two institutions \mathcal{I} and \mathcal{I}' , when a direct institution comorphism between them cannot be given, we can devise an auxiliary “intermediate” institution \mathcal{I}'' that incorporates the common features of \mathcal{I} and \mathcal{I}' , and relate those features using a span of comorphisms $\mathcal{I} \xleftarrow{\rho} \mathcal{I}'' \xrightarrow{\rho'} \mathcal{I}'$. In particular, using spans of comorphisms we can capture *semi-comorphisms*, which relate signatures and models of institutions as comorphisms do, but do not translate sentences at all (then the intermediate institution has the empty sets of sentences).

It turns out, however, that spans of comorphisms need not offer the most natural way to capture certain “consistency” (rather than “sharing”) requirements between models of two institutions. It is often more natural to join the two institutions then by a *sink* (*co-span*) of comorphisms $\mathcal{I} \xrightarrow{\rho} \mathcal{I}'' \xleftarrow{\rho'} \mathcal{I}'$ that embed the institutions \mathcal{I} and \mathcal{I}' into an auxiliary, richer institution \mathcal{I}'' , where the consistency conditions may be expressed.

(A perhaps even more natural alternative, which we will not discuss here, is to link the two institutions by a span of institution morphisms.)

Institutions with institution comorphisms (with rather obvious, component-wise composition) form a category $co\mathcal{INS}$. We will be working in a context of a number of logical systems, formalised as institutions, linked in various ways by institution comorphisms. We hence define a *heterogeneous institution environment* \mathcal{HIE} to be a diagram in the category $co\mathcal{INS}$, that is, for some underlying graph $\mathcal{G} = \langle \mathcal{N}, \mathcal{E} \rangle$,³ $\mathcal{HIE} = \langle \langle \mathcal{I}_n \rangle_{n \in \mathcal{N}}, \langle \rho_e \rangle_{e \in \mathcal{E}} \rangle$ consists of institutions \mathcal{I}_n , for $n \in \mathcal{N}$, and institution comorphisms $\rho_e : \mathcal{I}_n \rightarrow \mathcal{I}_m$, for $e : n \rightarrow m$ in \mathcal{E} .

As we have mentioned, quite a number of logical systems have been formalised as institutions in the literature. Similarly, quite a number of them have been linked by institution maps of various kinds, and hence by (spans of) comorphisms. Rarely, however, a number of these have been collected together to offer a framework for building heterogeneous specifications. One notable exception is the HETS family of institutions [37], supported by a tool to build and work with heterogeneous specifications [38].

In this paper we outline a family of institutions capturing various UML sublanguages and comorphisms that represent the expected semantic relationships between them — this too will form such a heterogeneous institution environment.

4 Institutions for the UML Sublanguages

The UML sublanguages, on the one hand, provide means for the design and specification of different aspects or views of a software system. On the other hand, institutions deliver a model theoretic characterisation of logics. Institutions, by the satisfaction condition, guarantee that certain properties still hold after renaming and/or identification of symbols. Consequently, and in the frame of an institution, two or more theories can be combined in such a way that the properties of each one of them do not get lost by putting them together. We investigate if the semantics of the UML sublanguages can be precisely captured by institutions, and how they can be linked.

Let us sketch institutions for the UML sublanguages of static structures (i.e., class diagrams), interaction diagrams, and OCL by means of a running example inspired by an e-learning case study for service-oriented computing [32]: In a university, thesis topics are managed by a central electronic office, where tutors announce topics that students can work on and where students may accept posted topics. The electronic office ensures certain conditions, like that no student is given more than one topic.

We do not detail here the institutions mentioned above fully formally, for more details see [19,18,17]. We also leave out for now possible definitions of other UML sublanguages as institutions.

4.1 Institution of Static Structures

Signatures for static structures declare class names, typed attributes and methods, and association names with corresponding association ends. For instance, the class diagram

³ \mathcal{G} is given by a set of nodes \mathcal{N} and a family of sets of edges $\mathcal{E} = \langle E_{n,m} \rangle_{n,m \in \mathcal{N}}$ unambiguously classified by their source and target. We identify \mathcal{E} with $\bigcup_{n,m \in \mathcal{N}} E_{n,m}$ and write $e : n \rightarrow m$ for $e \in E_{n,m}$.

in Fig. 2 declares

```
({EOffice, Topic, Tutor, Student, String, Void},
 {tname : Tutor → String,
  sname : Student → String,
  content : Topic → String,
  announce : EOffice × Tutor × Topic → Void,
  post : EOffice × Tutor × Topic → Void,
  accept : EOffice × Student × Topic → Void,
  register : EOffice × Student × Topic → Void},
 {tuteof ⊆ tutors : Tutor × eoffice : EOffice,
  tuttop ⊆ tutor : Tutor × topics : Topic,
  topeof ⊆ topics : Topic × eoffice : EOffice,
  topstu ⊆ topic : Topic × student : Student,
  stueof ⊆ students : Student × eoffice : EOffice})
```

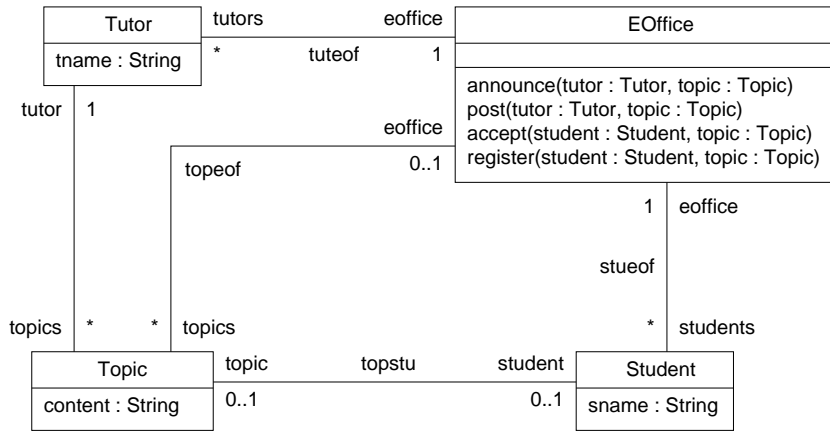


Fig. 2. E-office example: class diagram (1)

Sentences associated with a signature of the institution of static structures declare multiplicities for associations. The class diagram of Fig. 2 presents a theory axiomatized by the following sentences

```

association(tuteof, tutors : Tutor : 0..*, eoffice : EOffice : 1..1) ,
association(tuttop, tutor : Tutor : 1..1, topics : Topic : 0..*) ,
association(topeof, topics : Topic : 0..*, eoffice : EOffice : 0..1) ,
association(topstu, topic : Topic : 0..1, student : Student : 0..1) ,
association(stueof, students : Student : 0..*, eoffice : EOffice : 1..1) .
  
```

Models of a class diagram signature are given as sets of object states. Object states are sets of created object identifiers of the declared class names, together with functions that interpret attributes and methods, as well as relations that interpret associations. Moreover, models of a presentation are required to satisfy the constraints put on associations. In our example we require, for instance, that for each topic there is at most one student and for each student there is at most one topic, so that if we navigate from a topic to its student, then we can navigate back to the topic, and vice versa.

For signature morphisms, translations, and reducts, consider again the class diagram in Fig. 2 and the class diagram in Fig. 3 with an additional method `remove` for class `EOffice`, and different multiplicities for association `tuteof`. A signature morphism σ from the signature induced by the former to the signature induced by the latter can be defined with $\sigma(x) = x$ for every element of the simpler signature. The reduct of any model simply “forgets” the interpretation of the method `remove`. Signature morphisms canonically extend to sentences: the axioms of the simple signature are not rephrased in the context of the complex signature, whose axioms in fact are stronger. Indeed, whereas the class diagram of Fig. 2 allows an arbitrary number of tutors for an e-office, the class diagram of Fig. 3 requires at least one tutor per e-office. Therefore, for any model satisfying the stronger axiom, its reduct also satisfies the weaker axiom.

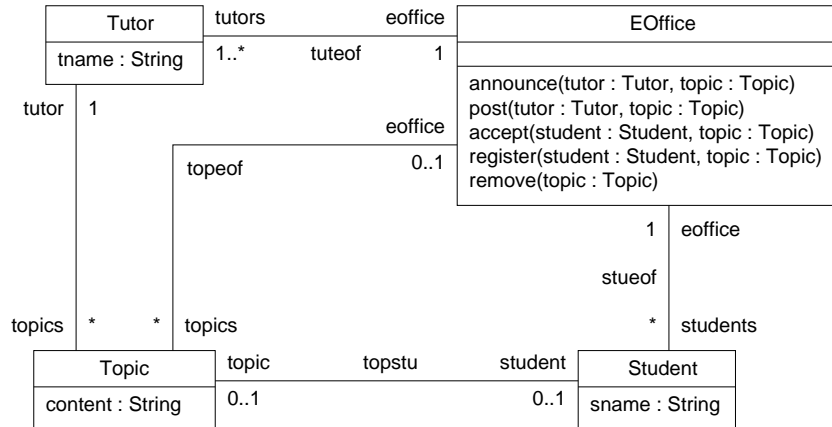


Fig. 3. E-office example: class diagram (2)

4.2 Institution of Interactions

Signatures for interactions simply declare class names and class-typed messages. Given a set of variables, typed over declared class names, a signature induces a set of sentences as follows. Atomic formulas are sequence diagrams (mathematically represented using labelled pomsets, see [41]) and their composition using interaction-building operators like for instance `seq`, `par`, `loop`, etc. (for details see [16]). Well-formed formulas com-

bine atomic ones using conjunction, negation, universal quantification, and equality of variables. As usual, sentences are closed formulas.

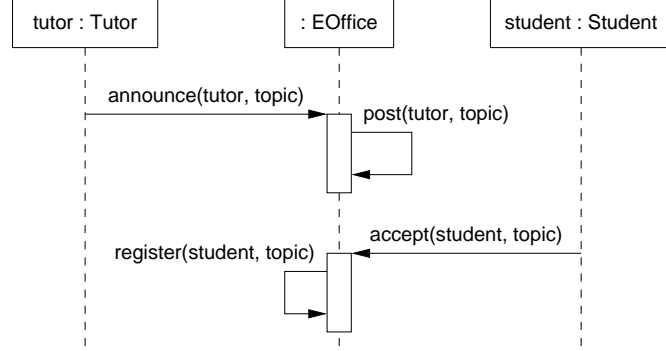


Fig. 4. E-office example: sequence diagram for accepting a topic

For instance the interaction diagram in Fig. 4 declares classes Tutor, Topic, EOffice, and Student, and messages $\text{announce}(\text{Tutor}, \text{Topic})$, $\text{post}(\text{Tutor}, \text{Topic})$, $\text{accept}(\text{Student}, \text{Topic})$, and $\text{register}(\text{Student}, \text{Topic})$. Given variables $tutor : \text{Tutor}$, $eoffice : \text{EOffice}$, and $student : \text{Student}$, the only sentence represented by the diagram is the atomic formula given by the following pomset:

$$\begin{aligned}
 & [\{ X_1, X_2, X_3, X_4, X_5, X_6, X_7, X_8 \}, \\
 & \{ X_1 < X_2 < X_3 < X_4 < X_6 < X_7 < X_8, X_5 < X_6 \}, \\
 & \{ X_1 \mapsto \text{snd}(tutor, eoffice, \text{announce}(tutor, topic)), \\
 & \quad X_2 \mapsto \text{rcv}(tutor, eoffice, \text{announce}(tutor, topic)), \\
 & \quad X_3 \mapsto \text{snd}(eoffice, eoffice, \text{post}(tutor, topic)), \\
 & \quad X_4 \mapsto \text{rcv}(eoffice, eoffice, \text{post}(tutor, topic)), \\
 & \quad X_5 \mapsto \text{snd}(student, eoffice, \text{accept}(student, topic)), \\
 & \quad X_6 \mapsto \text{rcv}(student, eoffice, \text{accept}(student, topic)), \\
 & \quad X_7 \mapsto \text{snd}(eoffice, eoffice, \text{register}(student, topic)), \\
 & \quad X_8 \mapsto \text{rcv}(eoffice, eoffice, \text{register}(student, topic)) \}]
 \end{aligned}$$

In the institution of interactions, an interpretation maps class names to sets of object instances and messages to message instances. Object instances may interchange message instances; we define send events as triples of a sender instance, a receiver instance, and a message instance; receive events are defined similarly. Traces are sequences of send/receive events. Models for interactions are defined to be pairs (P, N) of sets of traces.

In order to grasp the notion of models for interactions and of satisfaction relation of an interaction by a model, recall that interactions depict possible message interchange scenarios, and perhaps forbid (other) scenarios by use of the interaction-building operator neg . In general, hence, interactions do not completely specify a software system.

Given an interpretation, given a valuation of variables by object instances (of the correct class), an interaction induces *positive* and *negative* traces of send/receive events. These induced sets, of all positive traces and of all negative traces, need not be disjoint, and their union need not contain every possible trace [16]. Given a valuation, a model (P, N) satisfies an interaction if the negative traces of the interaction diagram under that valuation are contained in N , and the genuine positive ones (i.e., positive and non-negative) are included in P . Equality, negation, conjunction and universal quantification are interpreted as usual.

On the one hand, for example the trace

$$\begin{aligned} & \text{snd}(\text{tutor}, \text{eoffice}, \text{announce}(\text{tutor}, \text{topic})) \cdot \text{rcv}(\text{tutor}, \text{eoffice}, \text{announce}(\text{tutor}, \text{topic})) \cdot \\ & \text{snd}(\text{eoffice}, \text{eoffice}, \text{post}(\text{tutor}, \text{topic})) \cdot \text{rcv}(\text{eoffice}, \text{eoffice}, \text{post}(\text{tutor}, \text{topic})) \cdot \\ & \text{snd}(\text{student}, \text{eoffice}, \text{accept}(\text{student}, \text{topic})) \cdot \text{rcv}(\text{student}, \text{eoffice}, \text{accept}(\text{student}, \text{topic})) \cdot \\ & \text{snd}(\text{eoffice}, \text{eoffice}, \text{register}(\text{student}, \text{topic})) \cdot \text{rcv}(\text{eoffice}, \text{eoffice}, \text{register}(\text{student}, \text{topic})) \end{aligned}$$

positively satisfies the interaction diagram of Fig. 4, where *tutor*, *topic*, *eoffice* and *student* are instances of the corresponding classes, and the message instances coincide with the messages declared in the signature of above. On the other hand, no trace negatively satisfies the interaction, i.e., the set of negative traces of the interaction is empty.

An equivalent abstract representation of the diagram of Fig. 4 is $\text{seq}(S_1, S_2)$, where S_1 is the labelled pomset

$$\begin{aligned} & [\{X_1, X_2, X_3, X_4\}, \\ & \{X_1 < X_2 < X_3 < X_4\}, \\ & \{X_1 \mapsto \text{snd}(\text{tutor}, \text{eoffice}, \text{announce}(\text{tutor}, \text{topic})), \\ & \quad X_2 \mapsto \text{rcv}(\text{tutor}, \text{eoffice}, \text{announce}(\text{tutor}, \text{topic})), \\ & \quad X_3 \mapsto \text{snd}(\text{eoffice}, \text{eoffice}, \text{post}(\text{tutor}, \text{topic})), \\ & \quad X_4 \mapsto \text{rcv}(\text{eoffice}, \text{eoffice}, \text{post}(\text{tutor}, \text{topic})) \quad \}] \end{aligned}$$

and S_2 the labelled pomset

$$\begin{aligned} & [\{Y_1, Y_2, Y_3, Y_4\}, \\ & \{Y_1 < Y_2 < Y_3 < Y_4\}, \\ & \{Y_1 \mapsto \text{snd}(\text{student}, \text{eoffice}, \text{accept}(\text{student}, \text{topic})), \\ & \quad Y_2 \mapsto \text{rcv}(\text{student}, \text{eoffice}, \text{accept}(\text{student}, \text{topic})), \\ & \quad Y_3 \mapsto \text{snd}(\text{eoffice}, \text{eoffice}, \text{register}(\text{student}, \text{topic})), \\ & \quad Y_4 \mapsto \text{rcv}(\text{eoffice}, \text{eoffice}, \text{register}(\text{student}, \text{topic})) \quad \}] \end{aligned}$$

and seq combines its arguments sequentially in a “instance-wise” manner: For each instance, all events for this instance in the first argument must happen before all events for this instance in the second argument. Hence, we could moreover have split the pomset S_2 into S_2^1 and S_2^2 with S_2^1 representing the sending and reception of the message $\text{accept}(\text{student}, \text{topic})$, and S_2^2 the sending and reception of the message $\text{register}(\text{student}, \text{topic})$, and combine S_2^1 and S_2^2 using the seq interaction-building operator. In other words, the abstract representation of a diagram needs not be unique. These different representations are nevertheless equivalent, i.e., have the same models.

Signature morphisms in the interactions institution work similarly to the signature morphisms in the static structure institution, renaming classes and messages. The translation of sentences along a signature morphism is canonical, and reducts forget all traces mentioning events only expressible over the target signature.

4.3 OCL Institution

OCL signatures declare class names, query names (that correspond to attributes and query methods), and method names. Class names are equipped with a partial order relation representing the inheritance hierarchy. Default (or built-in) types extend these declarations. So for instance the set of class names is closed under application of the type constructors Set and Sequence (which is equivalent to list construction). This extended type system is used to define a (unique) type for each query name and each method name. The inheritance hierarchy together with a built-in subtype relation induce the OCL-subtype relation, that comprises class names as well as query and method names. The sentences defined by an OCL signature are invariants and pre/postconditions, as in the example shown in Fig. 5. The corresponding signature declares, possibly among others, class names EOffice, Student, and Topic, query names topic and student for EOffice, student for Topic, topic for Student, and a method name register for EOffice. OCL presentations consist of an OCL signature and a set of OCL sentences over that signature.

```
context EOffice::accept(student : Student, topic : Topic)
pre: topic.student->empty() and student.topic->empty()
post: topic.student = student and student.topic = topic
```

Fig. 5. E-office example: OCL specification of accept

The OCL interpretations map class names to sets of created objects, and provide a mechanism to retrieve functions that implement query names and method names; see [15]. The former functions do not modify the state of objects, whereas the latter may modify the state. Models of OCL theories are state transition systems, whose states are sets of created objects and whose transitions are labelled by a method invocation and the corresponding return value, so that the target state of a transition is the result of applying the method on the origin state. Moreover, every state of these models observes the invariants of the theory, and any two adjacent states satisfy the pre-/postconditions required for the method that labels the transition connecting these two states.

Signature morphisms, translations and reducts, again, are built similarly to the corresponding notions in the static structure institution.

5 Linking UML Institutions

We study how particular views of a given software system, as represented in different languages designed for their specification, can be linked with each other. In particular, we investigate the natural links between the UML institutions as sketched above.

Following the presentation in Sect. 3, there are two possibilities to link institutions: direct translation from one institution to another via a comorphism, and definition of a new mediating institution to which both institutions are embedded (see definition of *sink* in Sect. 3).

```

context Tutor inv: eoffice->count() = 1
context Tutor inv: eoffice->tutors->includes(self)
context EOffice
  inv: tutors->forall(x | x.eoffice->includes(self))

```

Fig. 6. EOffice example: multiplicity axiom for tuteof translated into OCL

The institution of class diagrams can easily be embedded into the OCL institution. Indeed, class names are mapped to class names, typed attributes to queries, typed method to method names, and role names to set-valued queries (this means, the same class name possibly gets more queries when translated). Sentences of a theory presentation given by a class diagram are translated into OCL invariants. So, for instance, the first *association*-sentence associated with Fig. 2 is translated into the OCL invariants of Fig. 6. In general, for a binary association $a \subseteq r_1 : c_1 \times r_2 : c_2$ with declared multiplicity $association(a, r_1 : c_1 : m_1, r_2 : c_2 : m_2)$, the translation of each of the multiplicities consists of a constraint for navigability and up to two constraints for cardinality: if m_1 is of the form $n_1..n_2$, then

```

context  $c_1$  inv:  $r_2$ ->forall(x | x. $r_1$ ->contains(self))
context  $c_2$  inv:  $r_1$ ->count() >=  $n_1$  and  $r_1$ ->count() <=  $n_2$  ;

```

if m_1 is of the form $n..*$, then

```

context  $c_1$  inv:  $r_2$ ->forall(x | x. $r_1$ ->contains(self))
context  $c_2$  inv:  $r_1$ ->count() >=  $n_1$  ;

```

and similarly for m_2 .

This translation gets somewhat more involved for other than binary associations and the various kinds of multiplicities. Nevertheless, the translation is rather straightforward: the same as above, it takes care of back navigability and cardinalities within the bounds imposed by multiplicities.

Given an OCL model for a signature, we extract from it a model of an embedded class diagram signature by taking the set of states of the OCL model.

The institution of class diagrams cannot so easily be embedded in the institution of interactions. Again, class names could be mapped to class names, and typed methods to sets of messages. But it is not trivial how to embed typed attributes and association names, nor how to translate declarations of multiplicities for associations. Since class diagrams can be embedded into OCL, this matter, however, is not so crucial if the OCL institution and the interactions institution can be linked.

For this an auxiliary institution OCL+I can be devised that contains all the elements of the OCL institution as well as all the elements of the institution of interactions. Signatures declare class names (as OCL signatures and interaction signatures do), query

names and method names (as in the OCL institution); query and method names, together with variables typed over declared class names, induce messages (which correspond to messages in the institution of interactions). Sentences are either OCL sentences over class names, query names and method names, or interaction sentences over class names and induced messages. A model is a set of so-called runs; cf. [14,11]. Runs are sequences of pairs, each pair consisting of a set of created objects and a set of events. An event is a send or receive event (cf. models for interactions in the previous section) or a mark that indicates that a univocally identified method invocation has come to an end.⁴ Given pairs (ω_k, H_k) and (ω_l, H_l) of a run, with $k < l$, we say that the events in H_k occur before the events in H_l ; events within the same set are considered to occur simultaneously. A model satisfies an OCL+I sentence if every single run in the model satisfies the OCL+I sentence. The satisfaction condition for single runs, a relation between a run and an OCL+I sentence, is defined in three parts, namely for pre/postcondition pairs, for invariants, and for interactions.

A run satisfies an OCL pre/postcondition for a method m whenever for any two pairs (ω_k, H_k) and (ω_l, H_l) of the run, with k the precondition time (i.e., H_{k-1} containing the reception of a call on method m), and with l the corresponding postcondition time (i.e., H_{l-1} containing the mark that indicates that the call on method m at time k has come to an end), the following property holds: ω_k and the call satisfy the precondition, and ω_k, ω_l , the return value and the call satisfy the postcondition (we have to include the call when checking both the pre- and the postcondition, since these conditions may refer to the arguments of the call).

A run satisfies an OCL invariant if each set of created objects sans the objects currently executing a method (cf. definition of pre- and postcondition times above) observes the invariant.

Given a variable valuation, a run satisfies an interaction if any trace obtained from the run by first eliminating the sets of objects, then eliminating the marks, and finally linearising simultaneous events, is positive and non-negative for the interaction under the valuation.

For signatures and sentences, the embedding of the OCL institution as well as the embedding of the institution of interactions in the institution OCL+I are straightforward. The transformation of OCL+I models into OCL models and the one into interaction models can be sketched as follows. On the one hand, an OCL+I model defines an OCL model, i.e., a state transition system, whose set of states is the union of all the sets of objects of all the runs of the OCL+I model, and whose transitions are labelled $v.m(v_1, \dots, v_n) : v'$ and connect an origin state ω_k with a target state ω_l if these states are the precondition and postcondition times, respectively, of a method call m on v with arguments (v_1, \dots, v_n) and return value v' ; in case there is no return value, then the transition is labelled simply $v.m(v_1, \dots, v_m)$. Notice that a set of objects ω_i can occur more than one time within a run and within an OCL+I model; in the OCL model, the corresponding state may thus have many transitions arriving to and departing from

⁴ A termination mark is useful for asynchronous methods and signal processing. There may be also a termination mark for a synchronous method, however; in this case, the mark and the send event for the result value of the method execution must be contained in the same set H_i of events of the run (they occur simultaneously, so to speak).

it. On the other hand, an OCL+I model defines an interaction model (P, N) where P is the set of traces obtained from the runs of the OCL+I model by the procedure described above (elimination of sets of objects, deletion of marks from sets of events, and linearisation of simultaneous events) and N is the complement of P .

Notice that the set of sentences of a signature of OCL+I is, so to speak, the union of the set of sentences of the embedded OCL signature and the set of sentences of the embedded interactions signature, these two simpler signatures sharing somehow the method names (that are declared in the OCL signature and which occur in the messages of the interaction signature). Instead, we could moreover allow intertwined sentences as in Fig. 7.

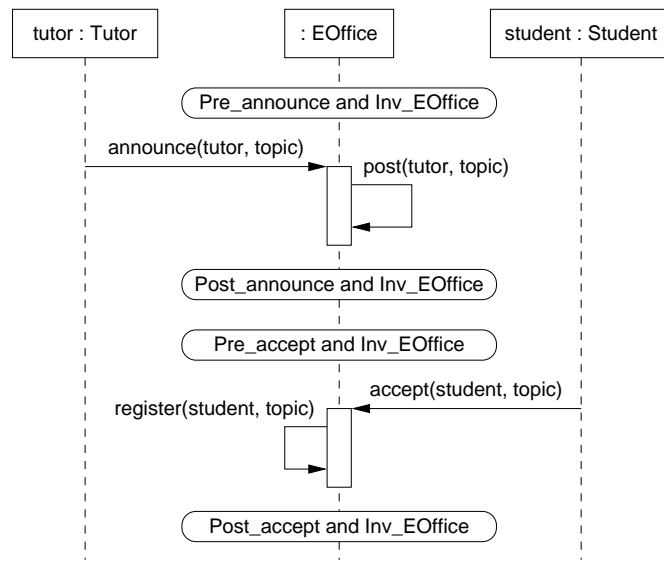


Fig. 7. E-office example: sequence diagram incl. OCL sentences for accepting a topic

These matters as well as criteria for devising a kind of (least) upper bound, i.e., a reasonable sink for two given institutions are subject of ongoing study.

6 Consistency Conditions

Working with a specification formalism with different viewpoints, like UML, raises the question concerning consistency of specifications developed in different viewpoints. In a heterogeneous institution environment, as the one sketched here for UML, this question is equivalent to the problem of characterising when specifications in different but linked institutions have, in some sense, a common model.

We begin by a concept of consistency within a single institution \mathcal{I} . Intuitively, two sets of sentences are consistent if their union admits a model. We want, however, to be

able to identify different symbols or to point at common symbols used by sentences in these two sets. We thus define *consistency* of two sets of sentences Φ_1 and Φ_2 over signatures Σ_1 and Σ_2 , respectively, with respect to a third signature Σ and signature morphisms $\sigma_1 : \Sigma_1 \rightarrow \Sigma$ and $\sigma_2 : \Sigma_2 \rightarrow \Sigma$ as follows: there is a model $M \in \mathbf{Mod}(\Sigma)$ such that $\sigma_1(\Phi_1) \cup \sigma_2(\Phi_2)$ is satisfied by M , or equivalently, $M|_{\sigma_1}$ satisfies Φ_1 and $M|_{\sigma_2}$ satisfies Φ_2 .

This concept of consistency can be lifted to a heterogeneous institution environment. Let us first consider two institutions \mathcal{I}_1 and \mathcal{I}_2 linked by a comorphism $\rho : \mathcal{I}_1 \rightarrow \mathcal{I}_2$. A set of sentences $\Phi_1 \subseteq \mathbf{Sen}_{\mathcal{I}_1}(\Sigma_1)$ is *consistent* with a set of sentences $\Phi_2 \subseteq \mathbf{Sen}_{\mathcal{I}_2}(\Sigma_2)$ with respect to a signature $\Sigma \in |\mathbf{Sign}_{\mathcal{I}_2}|$ and signature morphisms $\sigma_1 : \rho^{Sign}(\Sigma_1) \rightarrow \Sigma$ and $\sigma_2 : \Sigma_2 \rightarrow \Sigma$ if $\rho_{\Sigma_1}^{Sen}(\Phi_1)$ and Φ_2 are consistent with respect to Σ , σ_1 and σ_2 , i.e., if there is a model $M \in \mathbf{Mod}_{\mathcal{I}_2}(\Sigma)$ such that $\sigma_1(\rho_{\Sigma_1}^{Sen}(\Phi_1)) \cup \sigma_2(\Phi_2)$ is satisfied by M , or equivalently, $\rho_{\Sigma_1}^{Mod}(M|_{\sigma_1})$ satisfies Φ_1 and $M|_{\sigma_2}$ satisfies Φ_2 .

Generalizing this further, if institutions \mathcal{I}_1 and \mathcal{I}_2 are linked by a co-span $\mathcal{I}_1 \xrightarrow{\rho_1} \mathcal{I} \xleftarrow{\rho_2} \mathcal{I}_2$, two sets of sentences $\Phi_1 \subseteq \mathbf{Sen}_{\mathcal{I}_1}(\Sigma_1)$ and $\Phi_2 \subseteq \mathbf{Sen}_{\mathcal{I}_2}(\Sigma_2)$ are *consistent* with respect to a signature $\Sigma \in |\mathbf{Sign}_{\mathcal{I}}|$ and signature morphisms $\sigma_1 : \rho_1^{Sign}(\Sigma_1) \rightarrow \Sigma$ and $\sigma_2 : \rho_2^{Sign}(\Sigma_2) \rightarrow \Sigma$ if $\rho_1_{\Sigma_1}^{Sen}(\Phi_1)$ and $\rho_2_{\Sigma_2}^{Sen}(\Phi_2)$ are consistent with respect to Σ , σ_1 and σ_2 . This is equivalent to the existence of a model $M \in \mathbf{Mod}_{\mathcal{I}}(\Sigma)$ such that $\rho_1_{\Sigma_1}^{Mod}(M|_{\sigma_1}) \in \mathbf{Mod}_{\mathcal{I}_1}(\Sigma_1)$ satisfies Φ_1 and $\rho_2_{\Sigma_2}^{Mod}(M|_{\sigma_2}) \in \mathbf{Mod}_{\mathcal{I}_2}(\Sigma_2)$ satisfies Φ_2 .

In the heterogeneous institution environment for UML, the institutions for static structures and for OCL are linked by a comorphism. Given a class diagram as a specification in the institution of static structures and a set of invariants and operation specifications, we are interested in their consistency with respect to shared class names, attributes, and translated association ends. For example, when linking the translation of the class diagram in Fig. 2 with the OCL specification in Fig. 5, the conjuncts of the postcondition of `accept` imply each other due to the *association*-axiom for `topstu`.

As long as both specifications do not require any particular objects of the classes in the class diagram to exist, their consistency can always be witnessed by a model, i.e., a state transition system, whose states are empty sets of object instances. However, for our e-office example, if the OCL specification contains, e.g.,

```
context Tutor
inv: Tutor.allInstances()->count() >= 2
```

a common model must show at least two instances of `Tutor`, as this symbol is identified with `Tutor`. Thus, the invariants induced in the comorphism translation of class diagrams to OCL, which, in particular, require then at least one e-office to exist, must not contradict the invariants in the extended OCL specification.

The institution OCL+I integrates the semantics of OCL and of interactions via a co-span of institution comorphisms. It does so by presupposing a particular way of linking operation specifications from OCL with sequencing obligations from interactions. Here, we are interested in their consistency with respect to shared class names on the one hand, and identifying query and method calls with messages on the other. Consider, for example, the operation specification for `accept` in Fig. 5 and the interaction

in Fig. 4. Then, in order to be able to obtain a common model in OCL+I, it is required that the precondition of `accept` does not contradict the postcondition of `announce`, as the postcondition time of `announce` is the same as the precondition time of `accept`; indeed, this is the case in our specification when `accept` is called only by a student lacking a topic.

The particular integrating institution OCL+I with the co-span of institution comorphisms presented above is by no means the only possibility to link the OCL and the interactions institution. We could also choose a looser definition of satisfaction of an interaction by a run: a trace is obtained from a run by not only eliminating the set of objects and eliminating the marks but also skipping all those events that happen on objects that are not mentioned in the interaction. In this situation, the postcondition of an operation and the precondition of another operation, albeit the termination of the first and the start of the second happen immediately after each other in an interaction, do not have to show any correlation, as always some event, external to the interaction, could interfere and restore the precondition of the second operation.

7 Conclusions

We have presented a general framework for constructing the semantics of different UML diagram types in a flexible way. The framework is used as a mathematical basis of UML in the *SENSORIA* development approach for service-oriented systems. It relies on the mathematical theory of institutions and offers a new approach to the semantics of heterogeneous system specifications in a “heterogeneous institution environment”. It allows one to describe each diagram type in its “natural” semantics. Different diagram types are integrated via appropriate translations (into each other or into intermediate institutions), and in this way their semantic consistency can be analysed. Another advantage of our approach is that other system models can be easily integrated. For instance, rewriting logic and temporal logic are themselves institutions [12,13].

Institutions provide an elegant and robust framework for programming in the large and in particular for compositionality. Indeed, the satisfaction condition ensures that properties fulfilled by parts of a development do not get lost when putting those parts together. This is also true when, even at different places, those parts are made more specific using a refinement relation based on model-class inclusion. The trade-off is the loss of expressive power regarding some reflective properties like closed-world assumption (cf. the OCL constructs `Type` and `Type.allInstances`). Depending on the application, however, this is a price we are willing to pay, since the compositionality gained applies not only to the development of the software system as such but also to the verification of the whole system, which may proceed by verification of the parts.

In contrast to Mossakowski [36], we keep the different institutions (of the heterogeneous institution environment) separate and do not aim at integrating them into a single (heterogeneous) institution using the so called Grothendieck construction. The latter only puts the institutions side by side and allows at most the sharing of syntactic constructs. As a consequence, from this construction we do not get additional insight like e.g. consistency conditions, and may moreover lose the intuitive separation of

the diverse views offered by the individual institutions in a heterogeneous institution environment.

The ideas in this paper present only a first step to a comprehensive heterogeneous approach to system development which will support also model transformations, refinement, and deployment to particular programming languages environments, and provide relationships to “single” system models such as Broy’s stream-based system model. Currently we are studying the embedding of service-oriented concepts into our heterogeneous system model approach, ranging from declarative specifications of SCA in SRML [22] and Montanari’s Architectural Design Rewriting [10] to process algebraic specifications of the dynamic behaviour of services [4,28,30].

Acknowledgements. We would like to thank an anonymous referee for many valuable suggestions and José Meseguer for fruitful discussions.

References

1. L. Andrade, P. Baldan, H. Baumeister, R. Bruni, A. Corradini, R. De Nicola, J. L. Fiadeiro, F. Gadducci, S. Gnesi, P. Hoffman, N. Koch, P. Kosiuczenko, A. Lapadula, D. Latella, A. Lopes, M. Loreti, M. Massink, F. Mazzanti, U. Montanari, C. Oliveira, R. Pugliese, A. Tarlecki, M. Wermelinger, M. Wirsing, and A. Zawłocki. AGILE: Software architecture for mobility. In M. Wirsing, D. Pattinson, and R. Hennicker, editors, *Recent Trends in Algebraic Development Techniques—16th International Workshop, WADT 2002, Frauenchiemsee, Germany, Sept. 24–27, 2002*, volume 2755 of *LNCS*. Springer, Nov. 2003.
2. J. A. Bergstra and J. W. Klop. Process Algebra for Synchronous Communication. *Information and Control*, 60(1–3):109–137, 1984.
3. G. Bernot, S. Coudert, and P. L. Gall. Towards Heterogenous Formal Specifications. In M. Wirsing and M. Nivat, editors, *Proc. 5th Int. Conf. Algebraic Methodology and Software Technology*, volume 1101 of *Lect. Notes Comp. Sci.*, pages 458–472. Springer, Berlin, 1996.
4. M. Boreale, R. Bruni, L. Caires, R. De Nicola, I. Lanese, M. Loreti, F. Martins, U. Montanari, A. Ravara, D. Sangiorgi, V. Vasconcelos, and G. Zavattaro. SCC: a Service Centered Calculus. In M. Bravetti and G. Zavattaro, editors, *Proc. 3rd Int. Wsh. Web Services and Formal Methods (WS-FM 2006)*, volume 4184 of *Lect. Notes Comp. Sci.*, pages 38–57. Springer, 2006.
5. M. Broy, M. V. Cengarle, and B. Rumpe. Semantics of UML – Towards a System Model for UML: The Structural Data Model. Technical Report TUM-I0612, Institut für Informatik, Technische Universität München, June 2006.
6. M. Broy, M. V. Cengarle, and B. Rumpe. Semantics of UML – Towards a System Model for UML: The Control Model. Technical Report TUM-I0710, Institut für Informatik, Technische Universität München, Feb. 2007.
7. M. Broy, M. V. Cengarle, and B. Rumpe. Semantics of UML – Towards a System Model for UML: The State Machine Model. Technical Report TUM-I0711, Institut für Informatik, Technische Universität München, Feb. 2007.
8. M. Broy and K. Stølen. *Specification and Development of Interactive Systems: FOCUS on Streams, Interfaces, and Refinement*. Springer, 2001.
9. M. Broy and M. Wirsing. Algebraic State Machines. In T. Rus, editor, *Proc. 8th Int. Conf. Algebraic Methodology and Software Technology*, volume 1816 of *Lect. Notes Comp. Sci.*, pages 89–118. Springer, 2000.

10. R. Bruni, A. Lluch-Lafuente, U. Montanari, and E. Tuosto. Style-based architectural recon-figurations. Technical Report TR-07-17, Computer Science Department, University of Pisa, 2007.
11. D. Calegari. UML 2.0 Interactions with OCL/RT Constraints. Master's thesis, InCo-PEDECIBA, 2007. Technical report 07-17.
12. M. V. Cengarle. The Rewriting Logic institution. Technical Report 9801, Ludwig-Maximilians-Universität München, Institut für Informatik, 1998.
13. M. V. Cengarle. The Temporal Logic institution. Technical Report 9805, Ludwig-Maximilians-Universität München, Institut für Informatik, 1998.
14. M. V. Cengarle and A. Knapp. Towards OCL/RT. In L.-H. Eriksson and P. A. Lindsay, editors, *Proc. 11th Int. Symp. Formal Methods Europe (FME'02)*, volume 2391 of *Lect. Notes Comp. Sci.*, pages 390–409. Springer, 2002.
15. M. V. Cengarle and A. Knapp. OCL 1.4/1.5 vs. OCL 2.0 Expressions: Formal Semantics and Expressiveness. *Softw. Syst. Model.*, 3(1):9–30, 2004.
16. M. V. Cengarle and A. Knapp. UML 2.0 Interactions: Semantics and Refinement. In J. Jür-jens, E. B. Fernandez, R. France, and B. Rumpe, editors, *Proc. 3rd Int. Wsh. Critical Systems Development with UML (CSDUML'04)*, pages 85–99. Technical Report TUM-I0415, Institut für Informatik, Technische Universität München, 2004.
17. M. V. Cengarle and A. Knapp. An Institution for OCL 2.0. Technical Report 0801, Institut für Informatik, Ludwig-Maximilians-Universität München, 2008.
18. M. V. Cengarle and A. Knapp. An Institution for UML 2.0 Interactions. Technical Report TUM-I0808, Institut für Informatik, Technische Universität München, 2008.
19. M. V. Cengarle and A. Knapp. An Institution for UML 2.0 Static Structures. Technical Report TUM-I0807, Institut für Informatik, Technische Universität München, 2008.
20. R. Diaconescu. *Institution-independent Model Theory*. Birkhäuser, 2008. to appear.
21. H. Ehrig, J. Padberg, and F. Orejas. From basic views and aspects to integration of specifi-cation formalisms. In G. Paun, G. Rozenberg, and A. Salomaa, editors, *Current Trends in Theoretical Computer Science: Entering the 21th Century*, pages 202–214. World Scientific, 2001.
22. J. L. Fiadeiro, A. Lopes, and L. Bocchi. A Formal Approach to Service Component Archi-tecture. In M. Bravetti and G. Zavattaro, editors, *Proc. 3rd Int. Wsh. Web Services and Formal Methods (WS-FM 2006)*, volume 4184 of *Lect. Notes Comp. Sci.*, pages 193–213. Springer, 2006.
23. C. Fischer. CSP-OZ: How to Combine Z with a Process Algebra. In H. Bowman and J. Derrick, editors, *Proc. 2nd Int. Conf. Formal Methods for Open Object-Based Distributed Systems (FMOODS'97)*, volume 2, pages 423–438. Chapman & Hall, Boston, 1997.
24. F. Gadducci and U. Montanari. The Tile Model. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language, and Interaction: Essays in Honour of Robin Milner*, Foundations Of Computing Series, pages 133–166. The MIT Press, 2000.
25. J. A. Goguen. Data, schema, ontology and logic integration. *Logic J. IGPL*, 13(6):685–715, 2005.
26. J. A. Goguen and R. M. Burstall. Introducing Institutions. In E. Clarke and D. Kozen, editors, *Logics of Programs*, volume 164 of *Lect. Notes Comp. Sci.*, pages 221–256. Springer, 1984.
27. J. A. Goguen and R. M. Burstall. Institutions: Abstract Model Theory for Specification and Programming. *J. ACM*, 39(1):95–146, 1992.
28. C. Guidi, R. Lucchi, N. Busi, R. Gorrieri, and G. Zavattaro. SOCK: A Calculus for Service-Oriented Computing. In A. Dan and W. Lamersdorf, editors, *Proc. 4th Int. Conf. Service-Oriented Computing (ICSOC'06)*, volume 4294 of *Lect. Notes Comp. Sci.*, pages 327–338. Springer, 2006.
29. J. A. Goguen and G. Rosu. Institution Morphisms. *Form. Asp. Comp.*, 13(3-5):274–307, 2002.

30. A. Lapadula, R. Pugliese, and F. Tiezzi. A Calculus for Orchestration of Web Services. In R. D. Nicola, editor, *Proc. 16th Europ. Symp. Programming Languages and Systems (ESOP'07)*, volume 4421 of *Lect. Notes Comp. Sci.*, pages 33–47, 2007.
31. S. MacLane. *Categories for the Working Mathematician*. Springer, 1971.
32. P. Mayer, A. Schroeder, and N. Koch. A Model-Driven Approach to Service Orchestration. In *Proc. IEEE Int. Conf. Services Computing (SCC'08)*, 2008. Submitted.
33. J. Meseguer. General Logics. In H. D. Ebbinghaus, J. Fernandez-Prida, M. Garrido, and D. Lascar, editors, *Logic Colloquium '87*, pages 275–329. North Holland, 1989.
34. J. Meseguer. Conditional Rewriting Logic as a Unified Model of Concurrency. *Theo. Comp. Sci.*, 96:73–155, 1992.
35. J. Meseguer and U. Montanari. Mapping Tile Logic into Rewriting Logic. In F. Parisi-Presicce, editor, *Sel. Papers 12th Int. Wsh. Recent Trends in Algebraic Development Techniques (WADT'97)*, volume 1376 of *Lect. Notes Comp. Sci.*, pages 62–91. Springer, 1997.
36. T. Mossakowski. Heterogenous Development Graphs and Heterogeneous Borrowing. In M. Nielsen and U. Engberg, editors, *Proc. 5th Int. Conf. Foundations of Software Science and Computation Structures*, volume 2303 of *Lect. Notes Comp. Sci.*, pages 326–341. Springer, Berlin, 2002.
37. T. Mossakowski. Heterogeneous Specification and the Heterogeneous Tool Set. Habilitation thesis, Universität Bremen, 2005.
38. T. Mossakowski, C. Maeder, and K. Lüttich. The Heterogeneous Tool Set. In O. Grumberg and M. Huth, editors, *Proc. 13th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07)*, volume 4424 of *Lect. Notes Comp. Sci.*, pages 519–522. Springer, 2007.
39. P. Naur and B. Randell. *Software Engineering — Report on a Conference sponsored by the NATO Science Committee*. NATO Sci. Affairs Div., Bruxelles, Garmisch, 1969.
40. P. Pepper and M. Wirsing. A Method for the Development of Correct Software. In M. Broy and S. Jähnichen, editors, *KORSO: Methods, Languages, and Tools for the Construction of Correct Software, Final Report*, volume 1009 of *Lect. Notes in Comp. Sci.*, pages 27–57. Springer, 1995.
41. V. Pratt. Modeling Concurrency with Partial Orders. *Int. J. Parallel Program.*, 15(1):33–71, 1986.
42. G. Reggio and L. Repetto. CASL-CHART: A Combination of Statecharts and the Algebraic Specification Language CASL. In T. Rus, editor, *Proc. 8th Int. Conf. Algebraic Methodology and Software Technology (AMAST'00)*, volume 1816 of *Lect. Notes Comp. Sci.*, pages 243–272. Springer, Berlin, 2000.
43. A. Tarlecki. Institution representation. Unpublished note, Dept. of Computer Science, University of Edinburgh, 1987.
44. A. Tarlecki. Moving between Logical Systems. In M. Haveraaen, O.-J. Dahl, and O. Owe, editors, *Sel. Papers 11th Int. Wsh. Specification of Abstract Data Types (ADT'95)*, volume 1130 of *Lect. Notes Comp. Sci.*, pages 478–502. Springer, 1996.
45. A. Tarlecki. Towards heterogeneous specifications. In D. Gabbay and M. de Rijke, editors, *Frontiers of Combining Systems 2, Studies in Logic and Computation*, pages 337–360. Research Studies Press, 2000.
46. A. Tarlecki. Abstract Specification Theory: An Overview. In M. Broy and M. Pizka, editors, *Models, Algebras, and Logics of Engineering Software*, volume 191 of *NATO Science Series — Computer and System Sciences*, pages 43–79. IOS Press, 2003.
47. H. Wehrheim. Behavioural Subtyping in Object-Oriented Specification Formalisms. Habilitationsschrift, Carl-von-Ossietzky-Universität Oldenburg, 2002.
48. M. Wirsing, A. Clark, S. Gilmore, M. Hölzl, A. Knapp, N. Koch, and A. Schroeder. Semantic-Based Development of Service-Oriented Systems. In E. Najn et al., editor, *Proc.*

- 26th IFIP WG 6.1 International Conference on Formal Methods for Networked and Distributed Systems (FORTE'06), Paris, France*, volume 4229 of *Lect. Notes Comp. Sci.*, pages 24–45. Springer-Verlag, 2006.
49. M. Wirsing and A. Knapp. View Consistency in Software Development. In M. Wirsing, A. Knapp, and S. Balsamo, editors, *Proc. 9th Int. Wsh. Monterey. Radical Innovations of Software and Systems Engineering in the Future (RISSEF'02). Revised Papers*, volume 2941 of *Lect. Notes Comp. Sci.*, pages 341–357. Springer, 2004.
 50. M. Wirsing, R. D. Nicola, S. Gilmore, M. M. Hölzl, R. Lucchi, M. Tribastone, and G. Zavattaro. Sensoria process calculi for service-oriented computing. In U. Montanari, D. Sannella, and R. Bruni, editors, *Trustworthy Global Computing, Second Symposium, TGC 2006, Lucca, Italy, November 7-9, 2006, Revised Selected Papers*, volume 4661 of *Lecture Notes in Computer Science*, pages 30–50. Springer, 2007.