# View Consistency in Software Development*

Martin Wirsing and Alexander Knapp

Ludwig–Maximilians–Universität München
{wirsing,knapp}@informatik.uni-muenchen.de

**Abstract.** An algebraic approach to the view consistency problem in software development is provided. A view is formalised as a sentence of a viewpoint language; a viewpoint is given by a language and its semantics. Views in possibly different viewpoints are compared over a common view for consistency by a heterogenous pull-back construction. This general notion of view consistency is illustrated by several examples from viewpoints used in object-oriented software development.

## 1 Introduction

The use of views in software development supports an often desirable "separation of concerns". Each stakeholder of a software system may express his view of the system from his own viewpoint and may employ the notation most appropriate for this viewpoint. In particular, most viewpoints taken by system stakeholders concentrate on parts of the whole system under construction which may either be rather orthogonal and separated by clean interfaces, or may overlap in intricate ways. However, the use of different views in software development poses the problem to ensure consistency, i.e., to guarantee that there is an overall integration of the views that is implementable in a software product. On the one hand, this means to integrate partial descriptions of the system; on the other hand, different notations and their semantics have to be compared.

The "system-model" solution to the view consistency problem embeds all viewpoints used for software development in a single, unifying system model and compares the embedded views over the system model's semantics. This approach has been put forward, for example, by stream-based [6], graph grammar [7] and rewrite system models [13], or the integration of different specification formalisms, like CSP and Z [8, 19] or a combination of algebraic specifications and labelled transition systems [12, 16]. However, an encompassing single system model renders reasoning on different views in a formalism suitably adapted to the view's viewpoint rather difficult. The "heterogeneous-specification" line of research concentrates on the comparison and integration of different, heterogeneous specification formalisms, retaining the formalisms most appropriate for expressing parts of the overall problem. Most prominently, institutions [9] and general logics [11] are used as a formal basis establishing a powerful framework for heterogenous specifications and heterogenous proofs [2, 15, 14]. These investigations, which concentrate on formal, logic-based software development, is complemented by set-based frameworks for view comparison and integration [3, 5].

* This research has been partially sponsored by the DFG project WI 841/6-1 "InOpSys".

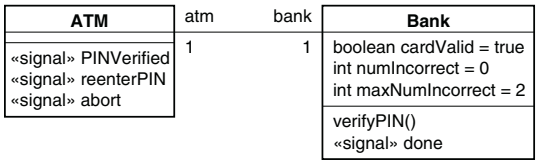
Our approach to the view consistency and integration problem in software development follows the "heterogeneous specification" approach, but applies these ideas to general software development. A viewpoint is given by a language and its semantics; a view in a viewpoint is a sentence of the viewpoint's language. We introduce a suitable notion of translations between viewpoints and view consistency for views in different viewpoints. We also define a general notion of consistency of views in possibly different viewpoints over a common view in maybe yet another viewpoint using a heterogenous pull-back construction. In particular, this construction avoids the unification of all different viewpoints into a single, formal system model. We illustrate our notion of consistency by several examples for viewpoints used in common object-oriented software development such as class diagrams and state machine diagrams.

The remainder of this paper is structured as follows: Sect. 2 motivates the use of multiple views and the consistency problem by means of examples. Sect. 3 introduces our algebraic notion of viewpoints and views. The translation of viewpoints and views is defined in Sect. 4. The algebraic notion of view consistency is presented in Sect. 5. We conclude by an outlook to further research topics. The appendices contain the formal definitions for the viewpoints used in the examples.

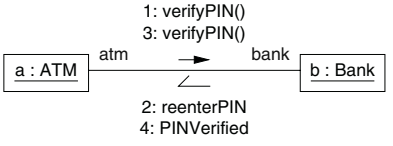
## 2 Multiple Views

In software development, multiple views and viewpoints are ubiquitous. As a simple example, consider the description of the interaction of an automatic teller machine (ATM) with a bank in Fig. 1. Using the "Unified Modeling Language" (UML [4]), the static structure of such a system may be specified by a UML class diagram as in Fig. 1(a). The dynamic behaviour of instances of the classes ATM and Bank may be given by state machine diagrams, see Fig. 1(d) for an ATM, Fig. 1(e) for a Bank. Finally, collaboration diagrams may be used for specifying desired (cf. Fig. 1(b)) or undesired behaviours (cf. Fig. 1(c)) of interaction.

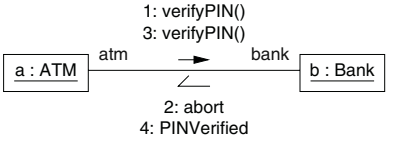
These views overlap and thus must be checked for consistency in several ways: First of all, the static structure and both of the state machines can be considered consistent if the state machines refer only to attributes, association ends, operations, and receptions that are declared in the class diagram. This syntactical notion of consistency amounts to extracting a class diagram from the state machines and checking whether this extracted class diagram is contained in the given class diagram of the static structure. In the same way, the collaboration diagrams can be checked for their class-diagram compatibility and, moreover, the same signature check must be applied to the collaborations and the state machines. Thus, when comparing the diagrams from a class-diagram or signature viewpoint, there must be translations from the views and their viewpoints under comparison into the signature viewpoint where consistency checking is signature inclusion. The same technique of consistency checking applies for showing that a collaboration is indeed realised by interacting state machines. As a collaboration specifies possible message exchanges and their order, the message exchanges of the interacting state machines have to be compared to these possible partial orders of message exchanges for inclusion. Hence, comparing diagrams from an interaction viewpoint involves translations from the views and their viewpoints under comparison into the interaction viewpoint



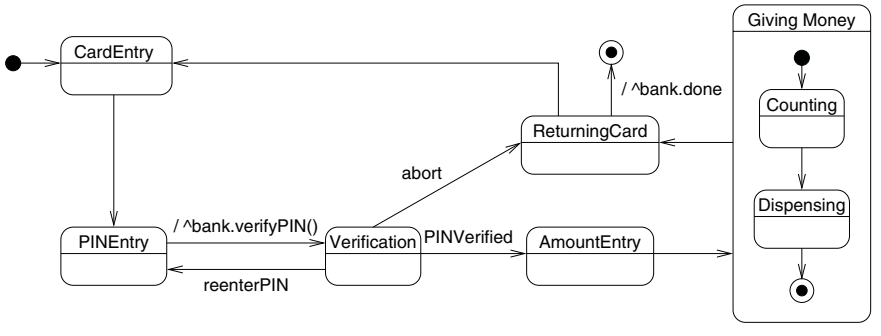

(a) Class diagram



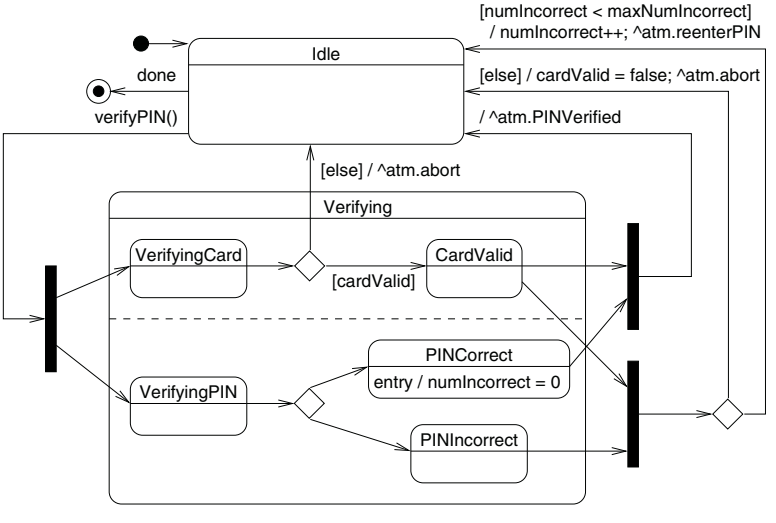

(b) Expected collaboration




(c) Erroneous collaboration

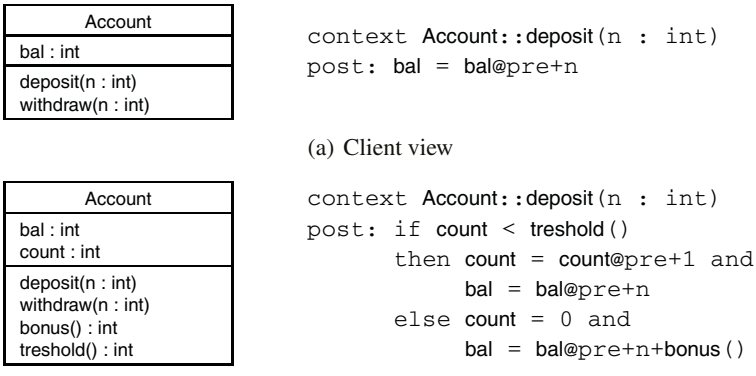



(d) State machine diagram for class `ATM`




(e) State machine diagram for class `Bank`

**Fig. 1.** Multiple views in a UML model of an ATM.




(a) Client view

(b) Management view

**Fig. 2.** Multiple views in UML models of a bank account.

where consistency checking is partial order inclusion. Similarly, the interaction viewpoint can be used to check that message exchanges given by a collaboration diagram are not included in the message exchanges of interacting state machines.

An analogous approach applies when a system is viewed from different angles by different stakeholders. Consider the (over-simplified) descriptions of a bank account depicted in Fig. 2, using UML and the "Object Constraint Language" (OCL [18]) as a pre-/postcondition language: A client may take the straightforward view of an account represented in Fig. 2(a), whereas the management of the bank may want to apply a special bonus rule for frequent customers thus taking the view in Fig. 2(b).

As in the previous example, several viewpoints are employed in both descriptions. From the structural viewpoint, UML class diagrams are used for specifying the static structure of a system, from the behavioural viewpoint the desired behaviour is expressed in terms of OCL pre-/postconditions on operations. Thus, taking the views as heterogenous views combined from views in the structural and the behavioural viewpoint, both the management and the client view can be checked for internal consistency. Furthermore, the views in the structural viewpoint may be considered to be consistent as the structural view in the management view simply extends the structural view in the client view. However, the behavioural views of the client and the management on the behaviour of the operation Account::deposit are inconsistent, as the client specification does not take into account the bonus feature of the management view.

## 3 Viewpoints and Views

Generalising from the examples above, a viewpoint of software development consists of a syntactic domain, a semantic domain, and a mapping from the syntax into the semantics. The syntactic domain captures the viewpoint notation, most conveniently in terms of an abstract syntax. A sentence of this abstract syntax conveys the information expressed in a view according to the viewpoint's notation. The semantic domain defines appropriate models for the abstract syntax.

More concretely, a *viewpoint* $\mathscr{V}$ is represented by a formal language category $L$, a semantic domain category $D$, and a semantic functor $\mathrm{Mod} : L^{\mathrm{op}} \to D$. A *view* in a viewpoint $\mathscr{V} = (L, D, \mathrm{Mod})$ is a specification object $V$ in the formal language category $L$; its semantics, i.e. its models, is given by $\mathrm{Mod}(V) \in D$. Note that we choose the opposite language category $L^{\mathrm{op}}$ for the semantics in order to express the well-known contravariance of syntax and semantics in logic and model-theory [1].

*Example 3.1.* The *structural viewpoint* $Struct$, cf. Fig. 1(a) and the class diagrams in Fig. 2(a) and 2(b) for concrete examples and App. B.1 for a detailed definition, uses the language $StructDiag$ of structure or class diagrams, comprising classes with typed attribute and method signatures and the (binary) associations between classes. As the semantic domain $StructAlg$ the class of many-sorted algebras over the many-sorted algebraic signatures induced from structure diagrams is employed. The signature of a structure diagram consists of sorts from the classes and operations from the attributes, methods, and associations. The model functor $\mathrm{Mod}_{Struct}$ maps a structure diagram to the algebras over its induced signature.

*Example 3.2.* The *behavioural viewpoint* $Beh$, cf. Fig. 2(a) and 2(b) for concrete examples and App. B.2 for a detailed definition, uses the language $BehSpec$ of pre-/ postconditions for annotating methods of classes. As the semantic domain $BehAlg$ the class of many-sorted algebras over the many-sorted algebraic signature induced from pre-/postcondition annotations is employed. The signature of pre-/postconditions consists of sorts for the classes and operations from the attributes and associations. The model functor $\mathrm{Mod}_{Beh}$ maps a pre-/postcondition specification to the algebras over its induced signature such that the methods applied to a state satisfy the pre-/postconditions.

*Example 3.3.* The *instance viewpoint* $Inst$, cf. App. B.3 for a detailed definition, uses the language $InstDiag$ of object diagrams, comprising typed objects with typed slots and their values and (binary) links between objects. As the semantic domain $InstAlg$ the class of many-sorted algebras over the many-sorted algebraic signatures induced from object diagrams is employed. The signature of an object diagram consists of sorts for the types of the objects, operations from the attributes and links, and constant operations for the objects. The model functor $\mathrm{Mod}_{Inst}$ maps an object diagram to the algebras over its induced signature such that the valuations of the slots and the links are satisfied.

*Example 3.4.* The *interaction viewpoint* $Inter$, cf. Fig. 1(b) and 1(c) for concrete examples and App. B.4 for a detailed definition, uses the language $InterDiag$ of collaboration diagrams, comprising objects and a partial order of message exchanges between these objects. As the semantic domain $InterAlg$ the class of algebras representing partial orders of messages between objects is employed. The model functor $\mathrm{Mod}_{Inter}$ maps a collaboration diagram to the class of partial orders of messages between objects that contain at least the partial order of message exchanges specified in the collaboration.

*Example 3.5.* The *machine viewpoint* $Mach$, cf. Fig. 1(d) and 1(e) for concrete examples and App. B.5 for a detailed definition, uses the language $MachDiag$ of state machine diagrams, comprising classes with their attributes and methods, and a mapping of classes to state machines. Each state machine is given by a set of states, an initial

state, and a set of transitions. Each transition is annotated with an event accepted by the transition, its effects and messages that are sent when the transition is taken. As the semantic domain $MachAlg$ the class of algebras representing the possible transitions of the state machines is employed. The model functor $\mathrm{Mod}_{Mach}$ maps a state machine diagram to the class of algebras with transitions for the machines in the diagram.

## 4 Translation

Viewpoints and thus views may be related by translations: Certain information of a view can be extracted and reformulated in another viewpoint. Translations may well induce partial loss of information, as not all viewpoints carry comparable data and not all viewpoints allow the specifier to express a software design at the same level of detail. In particular, translation may not always be possible syntactically. However, whenever information is shared between viewpoints, such as information on types in the structural, instance, interaction, and machine viewpoints, translations afford the necessary extraction mechanism.

A translation transfers information from one viewpoint $\mathscr{V}_1 = (L_1, D_1, \mathrm{Mod}_1)$ into another viewpoint $\mathscr{V}_2 = (L_2, D_2, \mathrm{Mod}_2)$. A *syntactic* viewpoint translation $\tau$ from $\mathscr{V}_1$ to $\mathscr{V}_2$ uses viewpoint notations: $\tau : L_1 \to L_2$. Corresponding to the contravariant behaviour of the model functors, a *semantic* viewpoint translation $\mu$ from $\mathscr{V}_1 \to \mathscr{V}_2$ operates contravariantly on the semantic domains of the viewpoints: $\mu : D_2 \to D_1$. A viewpoint translation from $\mathscr{V}_1$ to $\mathscr{V}_2$ consists of a pair of syntactic and semantic viewpoint translations $(\tau, \mu) : \mathscr{V}_1 \to \mathscr{V}_2$.

*Example 4.1.* We consider the translation of the instance viewpoint $Inst$ into the structural viewpoint $Struct$. Syntactically, a class diagram can be induced from an object diagram by turning type names, slots, links, and links ends into classes, attributes, associations, and association ends. Thus we have a syntactic viewpoint translation $\tau : InstDiag \to StructDiag$. As an algebra in $StructAlg$ always can be interpreted as an algebra in $InstAlg$ by forgetting the methods, we also have a semantic viewpoint translation $\mu : StructAlg \to InstAlg$.

*Example 4.2.* There is also a syntactical translation of the structural viewpoint $Struct$ into the instance viewpoint $Inst$: $\tau : StructDiag \to InstDiag$ can be trivially defined by translating each structure diagram into the empty object diagram. However, there is a more natural semantic translation from the structural viewpoint into the instance viewpoint $\mu : InstAlg \to StructAlg$ by forgetting the additional object constants.

*Example 4.3.* In a similar way to translating the instance viewpoint into the structural viewpoint, there is a syntactical translation $\tau : InstDiag \to InterDiag$ of the instance viewpoint $Inst$ into the interaction viewpoint $Inter$: The objects are kept, but the interaction diagram will contain no messages. Conversely, the interaction viewpoint can be translated syntactically into the instance viewpoint, $\tau : InterDiag \to InstDiag$, by keeping the objects and adding links between objects which exchange messages.

*Example 4.4.* The examples 4.1 and 4.3 can be combined into a syntactic translation $\tau : Inter \to Struct$ which infers the classes and associations from the interaction

diagram. However, there is also a direct syntactic translation from $Inter$ to $Struct$ that also keeps the messages.

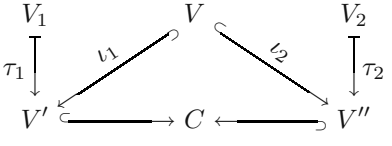
*Example 4.5.* A syntactic translation of the interaction viewpoint into the machine viewpoint can be achieved as follows: Each class of an object in the interaction is associated with a machine that accepts the messages incoming to the object. An incoming message is answered by all following outgoing messages [10]. This syntactic translation is complemented by a a semantic translation of the interaction viewpoint $Inter$ into the machine viewpoint $Mach$, i.e. $\mu : MachAlg \to InterAlg$. The machines in a machine algebra are run concurrently from their initial states. Each machine that fires a transition sends messages to the other machines. The receiving machine reacts to the incoming event by subsequently firing a transition. Thus, every run of the machines induces a partial order of exchanged messages.

## 5 Consistency

Given the notions of viewpoints, views, and viewpoint translations introduced above, the view consistency problem in software development amounts to relating views in different viewpoints by syntactical or semantical translations, and checking whether the overlapping parts of the views are acceptable to the software engineers. In particular, views may be consistent from certain viewpoints, but deemed to be inconsistent from others. Therefore it seems advisable to introduce a point of comparison between two views. This point of comparison can be chosen as a view in the viewpoint from one of the views to be related, or may be of yet another viewpoint. Moreover, the point of comparison view can specify the minimal requirements for the compared views or can embrace all the information, relative to the chosen viewpoint, that is expected to be available in the views to be compared. Finally, views can be compared syntactically involving syntactic translations, or semantically using semantic translations.

### 5.1 Syntactic Consistency

The syntactic consistency check for two views over a common point of comparison view necessitates the syntactic translation of the views under comparison into the viewpoint of the point of comparison. We call two views *consistent* over a common view if there are embeddings of the common view in the translated versions of the views under comparison. More precisely, when comparing the views $V_1$ and $V_2$ in viewpoints $\mathscr{V}_1 = (L_1, D_1, \mathrm{Mod}_1)$ and $\mathscr{V}_2 = (L_2, D_2, \mathrm{Mod}_2)$, respectively, over a point of comparison view $V$ in a viewpoint $\mathscr{V} = (L, D, \mathrm{Mod})$ we require that there are syntactic viewpoint translations $\tau_1 : \mathscr{V} \to \mathscr{V}_1$, $\tau_2 : \mathscr{V} \to \mathscr{V}_2$ from the viewpoint $\mathscr{V}$. Given these translations there must be embeddings $\iota_1 : V \to \tau_1(V_1)$ and $\iota_2 : V \to \tau_2(V_2)$. If $D$ admits push-outs, we obtain the following diagram:

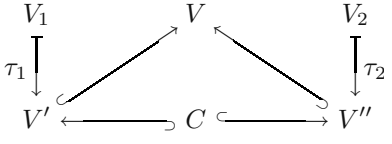


Here, the point of comparison view $V$ is *shared* between the (translated) views $V_1$ and $V_2$, that is, $V$ contains symbols that should have the same meaning in $V_1$ and $V_2$.

*Example 5.1.* Considering the structure diagram $S$ from Fig. 2(b) and the behavioural specifications $B_1$ and $B_2$ in Fig. 2(a) and Fig. 2(b), syntactic consistency of $B_1$ and $B_2$ over $S$ can be seen from the translations of behavioural specifications into structure diagrams: The additional pre-/postconditions are forgotten, thus the push-out in $StructDiag$ is isomorphic to $S$. Note that a comparison of $B_1$ and $B_2$ over the structure diagram from Fig. 2(a) would result in an enrich structure diagram also containing bonus and threshold.

In general two behavioural specifications $B_1$ and $B_2$ are consistent over a structure diagram $S$ if, and only if all classes, attribute signatures, operation signatures, and association end signatures of $S$ are contained in both behavioural specifications. The push-out constructs the union of all features of $B_1$ and $B_2$ separately renaming the features in $B_1$ and $B_2$ that are not present in $S$.

As the example illustrates it may sometimes be desirable to require that the point of comparison already contains all information that can be deduced from the views under comparison by translating these views into the common viewpoint. Such a point of comparison view is called *embracing* and leeds to a similar consistency diagram as for the shared case, but with arrows reversed such that $C$ now becomes a pull-back in $D$:

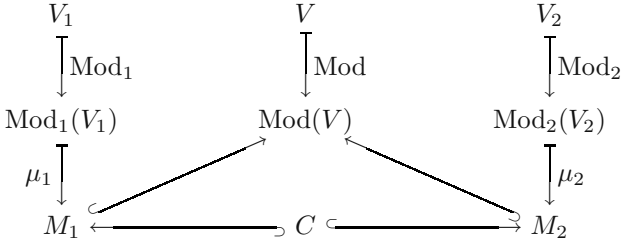



*Example 5.2.* We consider the comparison of the collaboration views $I_1$ and $I_2$ in Fig. 1(b) and Fig. 1(c) over the embracing point of comparison view $S$ in Fig. 1(a). Translating $I_1$ and $I_2$ into structure diagrams, there are embeddings of these translated interaction diagrams into $S$. Therefore $I_1$ and $I_2$ are syntactically consistent with respect to $S$. The pull back contains only the classes ATM and Bank with their respective methods PINVerified and verifyPIN.

### 5.2 Semantic Consistency

The semantic consistency check for two views over a comparison view involves the construction of a common model such that the comparison view is extended by the compared views consistently. The construction is dual to syntactic consistency. However, the existence of embeddings is sufficient to assess whether views are consistent or not, as embeddings may also exist for semantically inconsistent views. We therefore have to inspect the pull-back, i.e., roughly speaking, the intersection, of the model classes of the views. If the pull-back is empty, then the views are inconsistent; if it is not empty, the views are formally consistent, but the feature interaction of the views may still result in undesired behaviour which can be revealed by inspecting the models of the pull-back.

More specifically, let $V_1$ and $V_2$ be views in the viewpoints $\mathscr{V}_1 = (L_1, D_1, \mathrm{Mod}_1)$ and $\mathscr{V}_2 = (L_2, D_2, \mathrm{Mod}_2)$, respectively, and let $V$ be a (shared) point of comparison view in viewpoint $\mathscr{V} = (L, D, \mathrm{Mod})$. We say that $V_1$ and $V_2$ are *semantically consistent* with respect to $V$, if there are semantic viewpoint translations $\mu_1 : \mathscr{V}_1 \to \mathscr{V}$, $\mu_2 : \mathscr{V}_2 \to \mathscr{V}$ from the viewpoint $\mathscr{V}$ and there exists a pull-back $C$ in $D$ such that the following diagram commutes:




Note that again the common view provides the viewpoint of the integrating model. The commutation of the diagram above implies that $V$ is a shared view between $V_1$ and $V_2$. The opposite case, that $V$ embraces all information that can be extracted by viewpoint translations from $V_1$ and $V_2$ leads to a diagram where the embedding arrows between $V_1'$ and $V$, and $V_2'$ and $V$ have to be reversed.

*Example 5.3.* The semantical consistency of the behavioural specifications $B_1$ and $B_2$ in Fig. 2(a) and Fig. 2(b) over the structure diagram $S$ from Fig. 2(b) can be checked as follows: The models of $B_1$ are $\mathrm{Sig}_{Beh}(B_1) = \langle\{\mathsf{Account}, \mathsf{int}, \mathrm{State}\}, \{\mathsf{bal} : \mathrm{State} \times \mathsf{Account} \to \mathsf{int}, \ldots\}\rangle$-algebras subject to the behavioural specification `post: bal = bal@pre+n`, the models of $B_2$ are $\mathrm{Sig}_{Beh}(B_1)$-algebras subject to the behavioural specification `post: if count < threshold() ... endif`. But the models of $B_1$ and $B_2$ in the behavioural viewpoint $Beh$ can be translated into models in the structural viewpoint $Struct$ forgetting the additional sorts and operations. These translated models can be trivially embedded in $\mathrm{Mod}(S) = \mathrm{Sig}_{Struct}(S)$-$Alg$. The pull-back in $StructAlg$ turns out to be the category of algebras over the amalgamated sum of the signatures of $B_1$ and $B_2$ over the signature of $S$ where all symbols from $S$ are shared. In particular, the pull-back is not empty and thus $B_1$ and $B_2$ are semantically consistent over $S$. However, this formal consistency may be misleading as $\mathsf{bonus}^A$ is identically 0 for $A \in |C|$ or $\mathsf{count}^A(s,a) <^A \mathsf{threshold}^A(s,a)$ for all $s \in \mathrm{State}^A$, $a \in \mathsf{Account}^A$; this may not be desirable. Moreover, if a software designer requires bonus to be greater than zero, the views become inconsistent.

More generally, in the case of behavioural specifications the semantic consistency, i.e. the calculation of pull-backs, can be reduced to logical consistency by computing the conjunction of the theories of the views under consideration (modulo renamings required by the shared view).

*Example 5.4.* A comparison of the machine view $A$ with the state machines in Fig. 1(d) and 1(e) with the interaction view $I$ of the collaboration in Fig. 1(b) leaves several possiblities for choosing a common point of comparison view. We employ the empty interaction view $\emptyset$ as the shared view. The translation of $\mathrm{Mod}_{Mach}(A)$ into the interaction viewpoint amounts to all possible interaction sequences of the state machines. The pull-

back $C$ now contains all partial orders of message exchanges that are part of the partial orders in $\mathrm{Mod}_{Inter}(I)$ and the partial orders from the translation of $\mathrm{Mod}_{Mach}(A)$.

In general, an interaction diagram is consistent with a machine diagram w. r. t. the empty interaction diagram if there is a run of the collaboration diagram that is also a run of the state machines. In the case of finite state systems, like the ATM example, this property can be checked efficiently by model checking [17].

## 6 Conclusions

We have presented an algebraic framework for view consistency in software development. The framework is inspired by the institution-based approaches to heterogeneous specifications. Viewpoints are formalised as a pair of a syntactic and a semantic category linked by a model functor. Views are objects in the syntactic category. Consistency of views is defined by a heterogeneous pullback construction.

However, view consistency is merely a stepping stone to the successful employment of different views in software development. Views on a system will evolve over the software life-cycle, some may extend over all software development phases, some may be replaced, refined, or be combined during construction. Taking views and viewpoints serious hence in particular means to provide further support for separation of concerns by view maintainance allowing the different stakeholders to keep their viewpoints of the system. Correct view development, replacement, and refinement remain a challenge.

## References


1. Jon Barwise, editor. *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, 1977.
2. Gilles Bernot, Sophie Coudert, and Pascale Le Gall. Towards Heterogenous Formal Specifications. In Martin Wirsing and Maurice Nivat, editors, *Proc. 5th Int. Conf. Algebraic Methodology and Software Technology*, volume 1101 of *Lect. Notes Comp. Sci.*, pages 458–472. Springer, Berlin, 1996.
3. Eerke A. Boiten, Howard Bowman, John Derrick, Peter F. Linington, and Maarten Steen. Viewpoint consistency in ODP. *Computer Networks*, 34(3):503–537, 2000.
4. Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison–Wesley, Reading, Mass., &c., 1998.
5. Howard Bowman, Maarten W. A. Steen, Eerke A. Boiten, and John Derrick. A Formal Framework for Viewpoint Consistency. *Formal Methods in System Design*, 21:111–166, 2002.
6. Manfred Broy and Ketil Stølen. *Specification and Development of Interactive Systems:* FOCUS *on Streams, Interfaces, and Refinement*. Springer, Berlin, 2001.
7. Gregor Engels, Reiko Heckel, Gabi Taentzer, and Hartmut Ehrig. A Combined Reference Model- and View-Based Approach to System Specification. *Int. J. Softw. Knowl. Eng.*, 7(4):457–477, 1997.
8. Clemens Fischer. CSP-OZ: How to Combine Z with a Process Algebra. In Howard Bowman and John Derrick, editors, *Proc. 2nd Int. Conf. Formal Methods for Open Object-Based Distributed Systems*, volume 2, pages 423–438. Chapman & Hall, Boston, 1997.
9. Joseph A. Goguen and Rod M. Burstall. Institutions: Abstract Model Theory for Specification and Programming. *J. ACM*, 39(1):95–146, 1992.

10. Ingolf Krüger, Radu Grosu, Peter Scholz, and Manfred Broy. From MSCs to Statecharts. In Franz J. Rammig, editor, *Distributed and Parallel Embedded Systems*, pages 61–71. Kluwer Academic, Boston–Dordrecht, 1999.
11. José Meseguer. General Logics. In Heinz-Dieter Ebbinghaus, José Fernández-Prida, Manuel Garrido, Daniel Lascar, and Mario Rodríguez Artalejo, editors, *Proc. Logic Colloquium '87*, volume 129 of *Studies in Logic and the Foundatons of Mathematics*, pages 275–329. Elsevier, North Holland, Amsterdam, 1989.
12. José Meseguer. Conditional Rewriting Logic as a Unified Model of Concurrency. *Theo. Comp. Sci.*, 96:73–155, 1992.
13. José Meseguer. Formal Interoperability. In *Proc. 5th Int. Symp. Mathematics and Artificial Intelligence*, Fort Laurerdale, Florida, 1998. 9 pages.
14. Till Mossakowski. Heterogenous Development Graphs and Heterogeneous Borrowing. In Mogens Nielsen and Uffe 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.
15. Till Mossakowski, Andrzej Tarlecki, and Wiesław Pawłowski. Combining and Representing Logical Systems Using Model-Theoretic Parchments. In Francesco Parisi Presicce, editor, *Sel. Papers 12th Int. Wsh. Algebraic Development Techniques*, volume 1376 of *Lect. Notes Comp. Sci.*, pages 349–364. Springer, Berlin, 1998.
16. Gianna Reggio and Luigi Repetto. CASL-CHART: A Combination of Statecharts and the Algebraic Specification Language CASL. In Teodor Rus, editor, *Proc. 8th Int. Conf. Algebraic Methodology and Software Technology*, volume 1816 of *Lect. Notes Comp. Sci.*, pages 243–272. Springer, Berlin, 2000.
17. Timm Schäfer, Alexander Knapp, and Stephan Merz. Model Checking UML State Machines and Collaborations. In Scott Stoller and Willem Visser, editors, *Proc. Wsh. Software Model Checking*, volume 55(3) of *Elect. Notes Theo. Comp. Sci.*, Paris, 2001. 13 pages.
18. Jos Warmer and Anneke Kleppe. *The Object Constraint Language*. Addison–Wesley, Reading, Mass., &c., 1999.
19. Heike Wehrheim. *Behavioural Subtyping in Object-Oriented Specification Formalisms*. Habilitationsschrift, Carl-von-Ossietzky Universität Oldenburg, 2002.

## A Many-Sorted Algebras

*Signatures.* A (many-sorted) *signature* $\langle S, F\rangle$ consists of a set of *sort* symbols $S$ and a set of *operations* $F$ of the form $f : (s_i)_{1\leq i\leq k} \to s_0$ ($k \in \mathbb{N}$) with $f$ an operation symbol and $s_0, s_1, \ldots, s_k \in S$. We require all sort symbols and all operation symbols to be distinct.

A *signature morphism* $\sigma : \langle S, F\rangle \to \langle S', F'\rangle$ is given by a map $\sigma$ from the symbols of $\langle S, F\rangle$ to the symbols of $\langle S', F'\rangle$ such that a sort symbol is mapped to a sort symbol and an operation symbol is mapped to an operation symbol; and the following condition holds: Given an operation $f : (s_i)_{1\leq i\leq k} \to s_0 \in F$ the image $\sigma(f) : (\sigma(s_i))_{1\leq i\leq k} \to \sigma(s_0)$ is in $F'$.

The category *Sig* has as objects: signatures, and as morphisms: signature morphisms. The composition of signature morphisms is defined as function composition; the identity morphism on a signature is given by the identity on the signature's symbols.

*Algebras.* Given a (many-sorted) signature $\Sigma = \langle S, F\rangle$, a (many-sorted) *$\Sigma$-algebra* $A$ consists of a family of *universes* $(s^A)_{s\in S}$ and a family of functions $(f^A)_{f\in F}$ such that $f^A : (s_i^A)_{1\leq i\leq n} \to s_0^A$ for $f : (s_i)_{1\leq i\leq n} \to s_0 \in F$.

A $\Sigma$-*algebra morphism* $\alpha : A \to A'$ for a signature $\Sigma = \langle S, F\rangle$ and algebras $A$ and $A'$ is given by a family of functions $(\alpha_s : s^A \to s^{A'})_{s\in S}$ such that $\alpha_{s_0}(f^A(u_1, \ldots u_n)) = f^{A'}(\alpha_{s_1}(u_1), \ldots, \alpha_{s_n}(u_n))$ for $f : (s_i)_{1\leq i\leq n} \to s_0 \in F$.

The category $\Sigma$-$Alg$ has as objects: $\Sigma$-algebras, and as morphisms: $\Sigma$-algebra morphisms. The composition of $\Sigma$-algebra morphisms is defined as function composition; the identity morphism on a $\Sigma$-algebra is given by the family of identity functions on its universes.

A signature morphism $\sigma : \Sigma \to \Sigma'$ for signatures $\Sigma = \langle S, F\rangle$ and $\Sigma' = \langle S', F'\rangle$ induces a functor $\restriction_\sigma : \Sigma'$-$Alg \to \Sigma$-$Alg$ such that, for an $A' \in |\Sigma'\text{-}Alg|$, $s^{A'\restriction_\sigma} = \sigma(s)^{A'}$ for $s \in S$ and $f^{A'\restriction_\sigma} = \sigma(f)^{A'}$ for $f \in F$; and $(\alpha' : A' \to B')\restriction_\sigma = \alpha : A'\restriction_\sigma \to B'\restriction_\sigma$ where $(\alpha_s)_{s\in S} = (\alpha'_{\sigma(s)})_{s\in S}$.

The category $Alg$ has as objects: the categories $\Sigma$-$Alg$ for signatures $\Sigma$, and as morphisms: for every signature morphism $\sigma : \Sigma \to \Sigma'$ the reduct functor $\restriction_\sigma$. The composition of $Alg$-morphisms is defined as functor composition; the identity morphism on the category $\Sigma$-$Alg$ is given by the reduct functor from the identity signature morphism $\mathrm{id} : \Sigma \to \Sigma$.

# B Viewpoints

## B.1 Structural Viewpoint

*Syntax.* A *structure diagram* $\langle C, A\rangle$ consists of a set of classes $C$ and a set of associations $A$. Each *class* $\gamma \in C$ is given by its (unique) name; a set of *attributes* of the form $a : \tau$ with $a$ a (unique) name and $\tau$ the name of a class in $C$; and a set of *methods* of the form $m : (\tau_i)_{1\leq i\leq k} \to \tau_0$ ($k \in \mathbb{N}$) with $m$ a (unique) name and $\tau_0, \tau_1, \ldots, \tau_k$ names of classes in $C$. Each *association* $\alpha \in A$ is given by a pair of *association ends* of the form $\langle a_1 : \tau_1, a_2 : \tau_2\rangle$ with $a_1, a_2$ (unique) names and $\tau_1, \tau_2$ names of classes in $C$.

A *structure diagram morphism* $\sigma : \langle C, A\rangle \to \langle C', A'\rangle$ is given by a map $\sigma$ from the names of $\langle C, A\rangle$ to the names of $\langle C', A'\rangle$ such that a class name is mapped to a class name, an attribute name is mapped to an attribute name, &c; and the following conditions hold: (1) Given a class $\gamma \in C$ with name $n$, attributes $a_1 : \tau_1, \ldots, a_k : \tau_k$, and methods $m_1 : (\tau_{1i})_{1\leq i\leq k_1} \to \tau_{10}, \ldots, m_l : (\tau_{li})_{1\leq i\leq k_l} \to \tau_{l0}$ the class $\gamma' \in C'$ denoted by $\sigma(n)$ has at least the attributes $\sigma(a_1) : \sigma(\tau_1), \ldots, \sigma(a_k) : \sigma(\tau_k)$ and at least the methods $\sigma(m_1) : (\sigma(\tau_{1i}))_{1\leq i\leq k_1} \to \sigma(\tau_{10}), \ldots, \sigma(m_l) : (\sigma(\tau_{li}))_{1\leq i\leq k_l} \to \sigma(\tau_{l0})$; (2) given an association $\langle a_1 : \tau_1, a_2 : \tau_2\rangle \in A$ there is an association $\langle \sigma(a_1) : \sigma(\tau_1), \sigma(a_2) : \sigma(\tau_2)\rangle \in A'$.

The category $StructDiag$ has as objects: structure diagrams, and as morphisms: structure diagram morphisms. The composition of structure diagram morphisms is defined as function composition; the identity morphism on a structure diagram is given by the identity on the structure diagram's names.

*Signature.* The *signature* of a structure diagram $\mathrm{Sig}(\langle C, A\rangle) = \langle S, F\rangle$ is defined as follows: (1) $S$ contains a distinguished sort $\mathrm{State}$. (2) Every class name in $C$ gives rise to a sort symbol in $S$. (3) An attribute $a : \tau$ of a class with name $\gamma$ gives rise to an operation $a : \mathrm{State} \times \gamma \to \tau$ in $F$. (4) A method $m : (\tau_i)_{1\leq i\leq k} \to \tau_0$ of a class with name $\gamma$ gives rise to operations $m : \mathrm{State} \times \gamma \times (\tau_i)_{1\leq i\leq k} \to \tau_0$ and

$m\text{state} : \text{State} \times \gamma \times (\tau_i)_{1 \leq i \leq k} \to \text{State}$ in $F$. (5) An association $\langle a_1 : \tau_1, a_2 : \tau_2 \rangle$ in $A$ gives rise to the operations $a_1 : \text{State} \times \tau_2 \to \tau_1$ and $a_2 : \text{State} \times \tau_1 \to \tau_2$ in $F$. (6) $S$ and $F$ are the least such sets under inclusion.

As each structure diagram morphism naturally induces a signature morphism, the signature mapping on objects of *StructDiag* can be extended to a functor $\text{Sig} : \mathit{StructDiag} \to \mathit{Sig}$.

*Semantics.* The *semantics* $\text{Mod}(\langle C, A \rangle)$ of a structure diagram $\langle C, A \rangle$ is given by $\text{Sig}(\langle C, A \rangle)$-*Alg*. The full sub-category of *Alg* induced by the image of Mod is called *StructAlg*. This semantics on objects of *StructDiag* is lifted to a functor $\text{Mod} : \mathit{StructDiag}^{\text{op}} \to \mathit{StructAlg}$ by setting $\text{Mod}(\sigma^{\text{op}} : \langle C', A' \rangle \to \langle C, A \rangle) = \restriction_{\text{Sig}(\sigma)} : \text{Mod}(\langle C', A' \rangle) \to \text{Mod}(\langle C, A \rangle)$.

### B.2 Behavioural Viewpoint

*Syntax.* A *behaviour specification* $\langle C, \text{pre}, \text{post} \rangle$ consists of a set of classes $C$ and mappings pre and post from the methods of classes from $C$ to pre- and postcondition specifications. Each *class* $\gamma \in C$ is given by its (unique) name; a set of *attributes* of the form $a : \tau$ with $a$ a (unique) name and $\tau$ the name of a class in $C$; and a set of *methods* of the form $m : (x_i : \tau_i)_{1 \leq i \leq k} \to \tau_0$ ($k \in \mathbb{N}$) with $m$ a (unique) name, $x_1, \ldots, x_n$ (unique) parameter names, and $\tau_0, \tau_1, \ldots, \tau_k$ names of classes in $C$. A *pre-condition* of a method $m$ of class $\gamma \in C$ is given by a boolean term involving the names of attributes of $\gamma$ and the parameter names of $m$. A *postcondition* of a method $m$ of class $\gamma \in C$ is given by boolean term involving the names of attributes of $\gamma$, references to the pre-state of attributes using the special notation `@pre`, and the special constant `result`.

A *behaviour specification morphism* $\beta : \langle C, \text{pre}, \text{post} \langle$ is given by a map $\beta$ from the names of $\langle C, \text{pre}, \text{post} \rangle$ to the names of $\langle C', \text{pre}', \text{post}' \rangle$ such that a class name is mapped to a class name, an attribute name is mapped to an attribute name, &c; and the following conditions hold: (1) Given a class $\gamma \in C$ with name $n$, attributes $a_1 : \tau_1, \ldots, a_k : \tau_k$, and methods $m_1 : (x_{1i} : \tau_{1i})_{1 \leq i \leq k_1} \to \tau_{10}, \ldots, m_l : (x_{li} : \tau_{li})_{1 \leq i \leq k_l} \to \tau_{l0}$ the class $\gamma' \in C'$ denoted by $\sigma(n)$ has at least the attributes $\sigma(a_1) : \sigma(\tau_1), \ldots, \sigma(a_k) : \sigma(\tau_k)$ and at least the methods $\sigma(m_1) : (\sigma(x_{1i}) : \sigma(\tau_{1i}))_{1 \leq i \leq k_1} \to \sigma(\tau_{10}), \ldots, \sigma(m_l) : (\sigma(x_{li} : \sigma(\tau_{li}))_{1 \leq i \leq k_l} \to \sigma(\tau_{l0})$. (2) The homomorphic images of the pre- and postconditions of a method $m$ of class $\gamma$ w. r. t. $\beta$ is a pre- and postcondition of $\beta(m)$, resp.

The category *BehSpec* has as objects: behaviour specifications, and as morphisms: behaviour specification morphisms. The composition of behaviour specification morphisms is defined as function composition; the identity morphism on a behaviour specification is given by the identity on the behaviour specification's names.

*Signature.* The *signature* of a behaviour specification $\text{Sig}(\langle C, \text{pre}, \text{post} \rangle) = \langle S, F \rangle$ is defined as follows: (1) $S$ contains a distinguished sort State. (2) Every class name in $C$ gives rise to a sort symbol in $S$. (3) An attribute $a : \tau$ of a class with name $\gamma$ gives rise to an operation $a : \text{State} \times \gamma \to \tau$ in $F$. (4) A method $m : (\tau_i)_{1 \leq i \leq k} \to \tau_0$ of a class with name $\gamma$ gives rise to operations $m : \text{State} \times \gamma \times (\tau_i)_{1 \leq i \leq k} \to \tau_0$ and

$m\text{state} : \text{State} \times \gamma \times (\tau_i)_{1 \leq i \leq k} \rightarrow \text{State}$ in $F$. (5) $S$ and $F$ are the least such sets under inclusion.

As each behaviour specification morphism naturally induces a signature morphism, the signature mapping on objects of *BehSpec* can be extended to a functor $\text{Sig} : \mathit{BehSpec} \rightarrow \mathit{Sig}$.

*Semantics.* The *semantics* $\text{Mod}(\langle C, \text{pre}, \text{post}\rangle)$ of a structure diagram $\langle C, \text{pre}, \text{post}\rangle$ is given by the full sub-category of $\text{Sig}(\langle C, \text{pre}, \text{post}\rangle)$-*Alg*-algebras $A$ satisfying the following condition: The interpretation of a pre-condition of a method $m : (x_i : \tau_i)_{1 \leq i \leq k} \rightarrow \tau_0$ over an $s \in \text{State}^A$ and parameters $x_1 = v_1, \ldots, x_k = v_k$ implies the interpretation of the postcondition of $m$ over $s$ and $m\text{state}^A(s, v_1, \ldots, v_n)$ with $x_1 = v_1, \ldots, x_k = v_k$ and $\text{result}^A = m(s, v_1, \ldots, v_n)$. The full sub-category of $\mathit{Alg}$ induced by the image of Mod is called $\mathit{BehAlg}$. This semantics on objects of *BehSpec* is lifted to a functor $\text{Mod} : \mathit{BehSpec}^{\text{op}} \rightarrow \mathit{BehAlg}$ by setting $\text{Mod}(\sigma^{\text{op}} : \langle C', \text{pre}', \text{post}'\rangle \rightarrow \langle C, \text{pre}, \text{post}\rangle) = \restriction_{\text{Sig}(\sigma)} : \text{Mod}(\langle C', \text{pre}', \text{post}'\rangle) \rightarrow \text{Mod}(\langle C, \text{pre}, \text{post}\rangle)$.

### B.3 Instance Viewpoint

*Syntax.* An *object diagram* $\langle O, L\rangle$ is given by a set of objects $O$ and a set of links $L$. An *object* $o \in O$ is given by a (unique) name; a type (name); and a set of *slots* of the form $a = v$ with $a$ a (unique) name and $v$ an object in $O$. Each *link* $\lambda \in L$ is given by a pair of *link ends* of the form $\langle a_1 = v_1, a_2 = v_2\rangle$ with $a_1, a_2$ (unique) names and $v_1, v_2$ objects in $O$. An object diagram has to be *well-typed*: If $o$ and $o'$ both show type $\tau$ then: (1) if $a = v$ is a slot of $o$ with $v$ an object with type $\tau$ there is a slot $a = v'$ of $o'$ with $v'$ an object with type $\tau$; (2) if there is a link $\langle a_1 = o, a_2 = v_2\rangle$ then there is a link $\langle a_1 = o', a_2 = v_2'\rangle$; (3) if there is a link $\langle a_1 = v_1, a_2 = o\rangle$ then there is a link $\langle a_1 = v_1', a_2 = o'\rangle$.

An *object diagram morphism* $\iota : \langle O, L\rangle \rightarrow \langle O', L'\rangle$ is given by a map $\iota$ from $O$ to $O'$ such that the following conditions hold: (1) Given an object $o \in O$ with name $n$, type $\tau$, and slots $a_1 = v_1, \ldots, a_k = v_k$, the object $\iota(o) \in O'$ has type $\tau$ and at least the slots $a_1 = \iota(v_1), \ldots, \iota(a_k) = \sigma(v_k)$; (2) given a link $\langle a_1 = v_1, a_2 = v_2\rangle \in L$ there is a link $\langle a_1 = \iota(v_1), a_2 : \iota(v_2)\rangle \in L'$.

The category *ObjDiag* has as objects: object diagrams, and as morphisms: object diagram morphisms. The composition of object diagram morphisms is defined as function composition; the identity morphism on an object diagram is given by the identity on the object diagram's objects.

*Signature.* The *signature* $\text{Sig}(\langle O, L\rangle) = \langle S, F\rangle$ of an object diagram $\langle O, L\rangle$ is defined as follows: (1) Every type name of an object in $O$ gives rise to a sort symbol in $S$. (2) Every object $o$ in $O$ with type $\tau$ gives rise to a constant operation $o : \rightarrow \tau$ in $F$. (3) A slot $a = v$ of an object with type $\tau$ and $v$ an object with type $\tau'$ gives rise to an operation $a : \tau \rightarrow \tau'$ in $F$. (4) A link $\langle a_1 = v_1, a_2 = v_2\rangle$ in $L$ with $v_1$, $v_2$ objects with types $\tau_1$, $\tau_2$, resp., gives rise to the operations $a_1 : \tau_2 \rightarrow \tau_1$, $a_2 : \tau_1 \rightarrow \tau_2$ in $F$. (5) $S$ and $F$ are the least such sets under inclusion.

As each object diagram morphism naturally induces a signature morphism, the signature mapping on objects of *ObjDiag* can be extended to a functor $\text{Sig} : \mathit{ObjDiag} \rightarrow \mathit{Sig}$.

*Semantics.* The *semantics* $\mathrm{Mod}(\langle O, L\rangle)$ of an object diagram $\langle O, L\rangle$ is given by the full sub-category of $\mathrm{Sig}(\langle O, L\rangle)$-algebras $A$ satisfying the following conditions: If $a = v$ is a slot in $\langle O, L\rangle$ then $a^A(o^A) = v^A$; if $\langle a_1 = v_1, a_2 = v_2\rangle$ is a link in $\langle O, L\rangle$ then $a_1^A(v_2^A) = v_1^A$ and $a_2^A(v_1^A) = v_2^A$. The full sub-category of $Alg$ induced by the image of Mod is called $InstAlg$. This semantics on objects of $ObjDiag$ is lifted to a functor $\mathrm{Mod} : ObjDiag^{\mathrm{op}} \to InstAlg$ by setting $\mathrm{Mod}(\sigma^{\mathrm{op}} : \langle O', L'\rangle \to \langle O, L\rangle) = \restriction_{\mathrm{Sig}(\sigma)} : \mathrm{Mod}(\langle O', L'\rangle) \to \mathrm{Mod}(\langle O, L\rangle)$.

### B.4 Interaction Viewpoint

*Syntax.* An *interaction diagram* $\langle O, M, \leq\rangle$ is given by a set of objects $O$, a set of messages $M$, and a partial order $\leq \subseteq M \times M$. Each object $o \in O$ is given by a (unique) name and a type (name). Each *message* $m \in M$ is given by a (unique) name; a *sender* object in $O$, a *receiver* object in $O$, and a message *label* name.

An *interaction diagram morphism* $\mu : \langle O, M, \leq\rangle \to \langle O', M', \leq'\rangle$ is given by a map $\mu$ from the names of $\langle O, M, \leq\rangle$ to the names of $\langle O', M', \leq'\rangle$ such that object names are mapped to object names, type names are mapped to type names, &c., and the following conditions hold: (1) A message $m \in M$ with sender $s \in O$, receiver $r \in O$, and label $l$ is mapped to message $\mu(m) \in M'$ with sender $\mu(s) \in O'$, receiver $\mu(r) \in O'$, and label $\mu(l)$. (2) If $m \leq n$ for messages $m, n \in M$ then $\mu(m) \leq' \mu(n)$.

The category $InterDiag$ has as objects: interaction diagrams, and as morphisms: interaction diagram morphisms. The composition of interaction diagram morphisms is defined as function composition; the identity morphism on an interaction diagram is given by the identity on the interaction diagram's messages.

*Signature.* The *signature* $\mathrm{Sig}(\langle O, M, \leq\rangle) = \langle S, F\rangle$ of an interaction diagram $\langle O, M, \leq\rangle$ is defined as follows: (1) $S$ contains distinguished sorts $\mathrm{Obj}$, $\mathrm{Label}$, $\mathrm{Msg}$, and $\mathrm{Boolean}$. (2) Every type name $\tau$ of an object in $O$ gives rise to a sort symbol $\tau$ in $S$. (3) $F$ contains distinguished operation symbols $\mathrm{sender} : \mathrm{Msg} \to \mathrm{Obj}$, $\mathrm{receiver} : \mathrm{Msg} \to \mathrm{Obj}$, $\mathrm{label} : \mathrm{Msg} \to \mathrm{Label}$, and $\leq : \mathrm{Msg} \times \mathrm{Msg} \to \mathrm{Boolean}$. (4) Every object $o \in O$ with type name $\tau$ gives rise to a constant operation symbol $o : \to \tau$. (5) Every message label $l$ in $M$ gives rise to a constant operation $l : \to \mathrm{Label}$. (6) Every message with identifier $m$ gives rise to a constant operation symbol $m : \to \mathrm{Msg}$. (7) $S$ and $F$ are the least such sets under inclusion.

As each interaction diagram morphism naturally induces a signature morphism, the signature mapping on objects of $InterDiag$ can be extended to a functor $\mathrm{Sig} : InterDiag \to Sig$.

*Semantics.* The *semantics* $\mathrm{Mod}(\langle O, M, \leq\rangle)$ of an interaction diagram $\langle O, M, \leq\rangle$ is given by the full sub-category of $\mathrm{Sig}(\langle O, M, \leq\rangle)$-algebras $A$ satisfying the following conditions: (1) The sort symbol $\mathrm{Boolean}$ is interpreted as the standard booleans, i.e. $\mathrm{Boolean}^A = \mathbb{B}$. (2) $\mathrm{Obj}$ is interpreted as the union of the interpretion of all sort symbols $\tau$ for object type names. (3) If $m$ is a message in $M$ with sender $s$, receiver $r$, and message label $l$, then $\mathrm{sender}^A(m^A) = s^A$ and $\mathrm{receiver}^A(m^A) = r^A$ and $\mathrm{label}^A(m^A) = l^A$. (4) If $m$ and $n$ are messages in $M$ with $m \leq n$ then $\leq^A(m^A, n^A) = true$. The

full sub-category of $Alg$ induced by the image of $\mathrm{Mod}$ is called $InterAlg$. This semantics on objects of $InterDiag$ is lifted to a functor $\mathrm{Mod} : InterDiag^{\mathrm{op}} \to InterAlg$ by setting $\mathrm{Mod}(\sigma^{\mathrm{op}} : \langle O', M', \leq' \rangle \to \langle O, M, \leq \rangle) = \restriction_{\mathrm{Sig}(\sigma)} : \mathrm{Mod}(\langle O', M', \leq' \rangle) \to \mathrm{Mod}(\langle O, M, \leq \rangle)$.

### B.5 State Machine Viewpoint

*Syntax.* A *state machine diagram* $\langle C, \mu \rangle$ is given by a set of classes $C$ and a mapping $\mu$ of classes to machines. Each *class* $\gamma \in C$ is given by its (unique) name; a set of *attributes* of the form $a : \tau$ with $a$ a (unique) name and $\tau$ the name of a class in $C$; and a set of *methods* of the form $m : (\tau_i)_{1 \leq i \leq k} \to \tau_0$ ($k \in \mathbb{N}$) with $m$ a (unique) name and $\tau_0, \tau_1, \ldots, \tau_k$ names of classes in $C$. A *machine* of class $\gamma \in C$ is given by a set $S$ of (unique) *state* names, a set $T$ of transitions, and an *initial state* $i \in S$. A *transition* of a machine of class $\gamma$ is given by a source state $s_1 \in S$, an incoming event $e$ from the method names of $C$, a guard as a boolean expression of the names of attributes of $\gamma$ and the names of parameters of the event $e$, an effect consisting of a statement over the names of attributes of $\gamma$ and the names of parameters of the event $e$ and a subset of outgoing messages, and a target state $s_2 \in S$. An outgoing message of a machine of class $\gamma$ is given by the name of an attribute of $\gamma$ and the name of a method name of the class of this attribute and list of expressions over the names of attributes of $\gamma$ and the parameters of $e$.

A *state machine diagram morphism* $\beta : \langle C, \mu \rangle \to \langle C', \mu' \rangle$ is given by a map $\beta$ from the names in $\langle C, \mu \rangle$ to the names in $\langle C', \mu' \rangle$ such that class names are mapped to class names, attribute names are mapped to attribute names, &c., and the following condition holds: $\mu'(\beta(C)) = \beta(\mu(C))$ where $\beta(\mu(C))$ is the homomorphic extension of $\beta$ to guards, statements, and expressions.

The category $MachDiag$ has as objects: state machine diagrams, and as morphisms: state machine diagram morphisms. The composition of state machine diagram morphisms is defined as function composition; the identity morphism on a state machine diagram is given by the identity on the state machine diagram's classes, attributes, and method names.

*Signature.* The *signature* $\mathrm{Sig}(\langle C, \mu \rangle) = \langle S, F \rangle$ of a state machine diagram $\langle C, \mu \rangle$ is defined as follows: (1) $S$ contains distinguished sorts $\mathrm{Obj}$, $\mathrm{Set}(\mathrm{Obj})$, $\mathrm{Env}$, $\mathrm{State}$, $\mathrm{Msg}$, and $\mathrm{Set}(\mathrm{Msg})$. (2) Every class name $\gamma \in C$ gives rise to a sort symbol $\gamma$ in $S$. (3) $F$ contains a distinguished operation symbol $\mathrm{obj} : \mathrm{Env} \to \mathrm{Set}(\mathrm{Obj})$. (4) $F$ contains a distinguished operation symbol $\mathrm{initial} : \to \mathrm{State}$. (5) $F$ contains distinguished operation symbols $\mathrm{state} : \mathrm{Obj} \times \mathrm{State} \times \mathrm{Env} \to \mathrm{State}$, $\mathrm{env} : \mathrm{Obj} \times \mathrm{State} \times \mathrm{Env} \to \mathrm{Env}$, $\mathrm{msg} : \mathrm{Obj} \times \mathrm{State} \times \mathrm{Env} \to \mathrm{Msg}$. (6) An attribute $a : \tau$ of a class with name $\gamma$ gives rise to an operation $a : \mathrm{Env} \times \gamma \to \tau$. (7) A method $(\tau_i)_{1 \leq i \leq k} \to \tau_0$ of a class with name $\gamma$ gives rise to an operation $m : \gamma \times (\tau_i)_{1 \leq i \leq k} \to \mathrm{Msg}$. (7) $S$ and $F$ are the least such sets under inclusion.

As each state machine diagram morphism naturally induces a signature morphism, the signature mapping on objects of $MachDiag$ can be extended to a functor $\mathrm{Sig} : MachDiag \to Sig$.

*Semantics.* The *semantics* $\mathrm{Mod}(\langle C, \mu\rangle)$ of a state machine diagram $\langle C, \mu\rangle$ is given by the full sub-category of $\mathrm{Sig}(\langle C, \mu\rangle)$-algebras $A$ satisfying the following conditions: (1) The sort symbol $\mathrm{Obj}$ is interpreted as the union of the interpretions of the sort symbols $\tau$ for classes. (2) The sort symbol $\mathrm{Set}(\mathrm{Obj})$ is interpreted as the standard subsets of the interpretion of $\mathrm{Obj}$; the sort symbol $\mathrm{Set}(\mathrm{Msg})$ is interpreted as the standard subsets of the interpretion of $\mathrm{Msg}$. (3) The operaton symbol $\mathrm{env}$ is interpreted as the function that given an object in the interpretion of sort $\tau$, a state $s_1$ of the machine of class $\tau$, and an environment $e$ yields the environment resulting from executing the machine for class $\tau$ in one step. The operation symbols $\mathrm{state}$ and $\mathrm{msg}$ are to be interpreted similarly, but resulting in the state and the outgoing messages, respectively. The full sub-category of $Alg$ induced by the image of $\mathrm{Mod}$ is called $MachAlg$. This semantics on objects of $MachDiag$ is lifted to a functor $\mathrm{Mod} : MachDiag^{\mathrm{op}} \to MachAlg$ by setting $\mathrm{Mod}(\sigma^{\mathrm{op}} : \langle C', \mu'\rangle \to \langle C, \mu\rangle) = \restriction_{\mathrm{Sig}(\sigma)} : \mathrm{Mod}(\langle C', \mu'\rangle) \to \mathrm{Mod}(\langle C, \mu\rangle)$.