UML Interactions Meet State Machines — An Institutional Approach

Alexander Knapp[1] **and Till Mossakowski**[2]

1 **Universität Augsburg, Germany**
2 **Otto-von-Guericke Universität Magdeburg, Germany**

Abstract

UML allows the multi-viewpoint modelling of systems. One important question is whether an interaction as specified by a sequence diagram can be actually realised in the system. Here, the latter is specified as a combination of several state machines (one for each lifeline in the interaction) by a composite structure diagram. In order to tackle this question, we formalise the involved UML diagram types as institutions, and their relations as institution (co)morphisms.

1998 ACM Subject Classification D.2.1 Requirements/Specifications, D.2.2 Design Tools and Techniques

Keywords and phrases UML, state machines, interactions, composite structure diagrams, institutions, multi-view consistency

Digital Object Identifier 10.4230/LIPIcs.CALCO.2017.15

1 Introduction

The "Unified Modeling Language" (UML [17]) is a heterogeneous language: UML comprises a language family of 14 types of diagrams of structural and behavioural nature. These sub-languages are linked through a common meta-model, i.e., through abstract syntax; their semantics, however, is informally described mainly in isolation. In [10], we have outlined our research programme of "institutionalising UML". Our objective is to give, based on the theory of institutions [6], formal, heterogeneous semantics to UML, that — besides providing formal semantics for the individual sub-languages — ultimately allows to ask questions concerning the consistency between different diagram types and concerning refinement and implementation in a system development.

The horizontal dimension of the relationship between the different models has to ensure *consistency* of the models, i.e., that the models fit together and describe a coherent system. There are different kinds of consistency checks on the modelling level: Static checks ensuring type consistency and type correctness between types and instances. Dynamic checks include the properties and one or several cooperating instances or types. Most of the dynamic checks are theoretically undecidable, thus fully automatic tools will not be able to answer all instances. However, in many cases, useful automatic approximations are possible, while in other cases, manual effort may be involved.

In this paper, we study one such consistency problem that arises between UML state machine diagrams, UML composite structure diagrams, and UML sequence diagrams. The central question that we study is: Are the traces of an interaction diagram are realisable in a system of state machines, interlinked by a composite structure diagram? In order to answer this question, we need to define institutions for interaction diagrams and composite structure diagrams. These are also original contributions. By contrast, we can rely on a previous institution of state machines given in [11].

The remainder of this paper is structured as follows: In Sect. 2, we provide some background on our goal of heterogeneous institution-based UML semantics and introduce a small example illustrating interactions, state machines and composite structures. In Sect. 3, we recall an institution for state machines. The original contribution of this paper starts in Sect. 4, where we define an institution for interactions. Section 5 provides an institution for composite structures. Section 6 provides institution (co)morphisms for the interplay among these institutions, and discusses the verification of our main property of feasibility of an interaction. Finally, in Sect. 7 we conclude with an outlook to future work.

1.1 ATM Example

In order to illustrate our approach to a heterogeneous institutions-based UML semantics, we use as a small example the design of a traditional automatic teller machine (ATM) connected to a bank. For simplicity, we only describe the handling of entering a card and a PIN with the ATM. After entering the card, one has three trials for entering the correct PIN (which is checked by the bank). After three unsuccessful trials the card is kept.

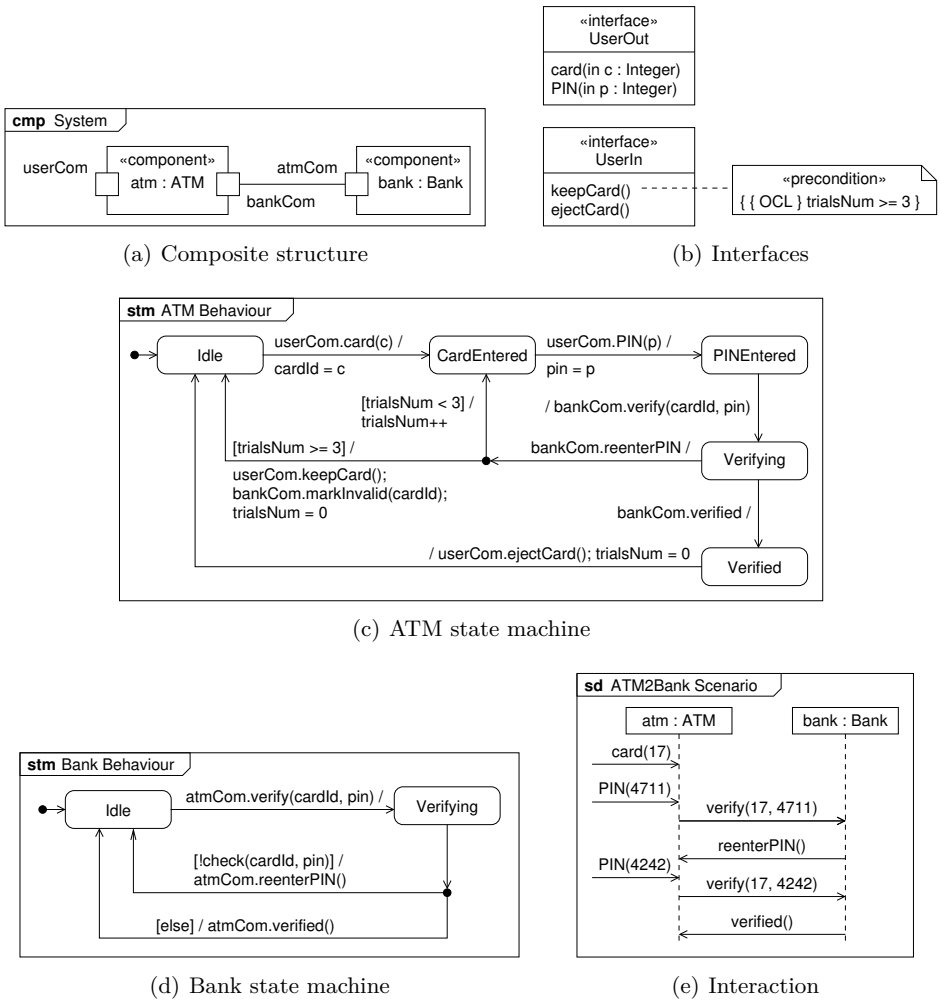
Figure 1(e) shows a possible interaction between an atm and a bank in an environment as a UML sequence diagram which consists of seven messages: after receiving a card and a PIN input from the environment, the atm requests the bank to verify if the card and PIN number combination is valid; first, the bank requests to reenter the PIN, but after receiving a second PIN the verification is successful.

The composite structure of the ATM-bank system is specified in the *component diagram* in Fig. 1(a). In order to communicate with a bank component, the atm component has a *behaviour port* called bankCom and the bank component has a behaviour port atmCom. Furthermore, atm has a port userCom to a user. Interpreted at the component instance level this *composite structure diagram* also specifies the initial configuration of the system with the component instances atm and bank for the interaction.

Figure 1(b) provides structural information in the form of the interfaces specifying what is provided at the userCom port of the atm instance (UserIn) and what is required (UserOut). An interface is a set of operations that other model elements have to implement. In our case, the interface is described in a *class diagram*. Here, the operation keepCard is enriched with a pre-condition trialsNum >= 3 expressed in the "Object Constraint Language" (OCL) which refines its semantics: keepCard can only be invoked if the constraint holds.

The dynamic behaviour of the atm component is specified by the *behavioural state machine* shown in Fig. 1(c). The machine consists of five states including Idle, CardEntered, etc. Beginning in the initial Idle state, the user can *trigger* a state change by entering the card. This has the *effect* that the parameter c from the card event is assigned to cardId in the atm. Entering a PIN triggers another transition to PINEntered. Then the ATM requests verification from the bank using its bankCom port. The transition to Verifying uses a *completion event*: No explicit trigger is declared and the machine autonomously creates such an event whenever a state is completed, i.e., all internal activities of the state are finished (in our example there are no such activities). If the interaction with the bank results in reenterPIN, and the *guard* trialsNum < 3 is true, the user can again enter a PIN. In general, a state machine proceeds in *run-to-completion steps*: In a state, an event is fetched from the machine's event pool, where completion events are preferred; a transition outgoing from the state, triggered by the event, and with its guard satisfied is chosen; and the chosen transition is fired, leaving the source state, executing the transition's effects, and entering the target state. Message reception is recorded by an (external) event in the state machine's event pool.

Another behavioural state machine specifies the behaviour of the bank component, see



(a) Composite structure (b) Interfaces

(c) ATM state machine

(d) Bank state machine (e) Interaction

Figure 1 ATM example

Fig. 1(d). For the sake of simplicity, we have omitted almost all details here. The machine waits in the starting Idle state until a verify event received via the atmCom port triggers the transition to the Verifying state. The machine then internally calls a check operation on the cardId and pin transmitted with the verify event. If this check fails, a reenterPIN event is sent to the ATM, otherwise, verified is sent. In reality, the machine and its check operation will be more involved. However, since these internals will not interfere with the ATM machine, they can be omitted here.

While this is a toy example, our envisaged interplay of interactions, state machines and composite structure diagrams is of industrial use: we cooperate with Fraunhofer IFF on applications of this very interplay to modelling parts of the smart electricity grid, e.g., adaptive grid protection devices and inter-station communication for voltage regulation.

2 Heterogeneous Institution-based UML Semantics

We have analysed in [9] that existing approaches to multi-view consistency in UML (i.e., addressing the question whether a family of UML diagrams is consistent) all have shortcomings

and drawbacks. Indeed, there are only few approaches that cover more than three different diagram types, and very few that cover composite structure diagrams. Moreover, even those that do cover the latter do not provide means to answer our central question, whether the interaction expressed by the sequence diagram can be realised by the interplay of several state machines that are interconnected through a composite structure diagram.

This situation motivated us to start a larger effort [10] of giving an institution-based heterogeneous semantics to several UML diagrams. The specific choice of an institution-based approach is motivated in greater detail in [9]. The work in this paper is part of this effort. The vision is to provide semantic foundations for model-based specification and design using a heterogeneous framework based on Goguen's and Burstall's theory of institutions [6]. We handle the complexity of giving a coherent semantics to UML by providing several institutions formalising different diagrams of UML, and several institution translations (formalised as so-called institution morphisms and comorphisms) describing their interaction and information flow. The central advantage of this approach over previous approaches to formal semantics for UML (e.g., [13]) is that each UML diagram type can stay "as-is", without the immediate need of a coding using graph grammars (as in [3]) or some logic (as in [13]). Such coding can be done at verification time — this keeps full flexibility in the choice of verification mechanisms. The formalisation of UML diagrams as institutions has the additional benefit that a notion of refinement comes for free, see [2] and Sect. 6 below. Furthermore, the framework is flexible enough to support various development paradigms as well as different resolutions of UML's semantic variation points. This is the crucial advantage of the proposed approach to the semantics of UML, compared to existing approaches in the literature which map UML to a specific global semantic domain in a fixed way.

2.1 Institutions

Institutions are an abstract formalisation of the notion of logical systems. Informally, institutions provide four different logical notions: signatures, sentences, structures[1], and satisfaction. Signatures provide the vocabulary that may appear in sentences and that is interpreted in structures (= realisations). The satisfaction relation determines whether a given sentence is satisfied in a given structure. The exact nature of signatures, sentences, and structures is left unspecified, which leads to a great flexibility. This is crucial for the possibility to model UML diagrams (which in the first place are not "logics") as institutions.

More formally [6], an institution $\mathscr{I} = (\mathrm{Sig}^{\mathscr{I}}, \mathit{Sen}^{\mathscr{I}}, \mathrm{Str}^{\mathscr{I}}, \models^{\mathscr{I}})$ consists of (i) a category of *signatures* $\mathrm{Sig}^{\mathscr{I}}$; (ii) a *sentence functor* $\mathit{Sen}^{\mathscr{I}} : \mathrm{Sig}^{\mathscr{I}} \to \mathrm{Set}$, where Set is the category of sets; (iii) a contra-variant *structure functor* $\mathrm{Str}^{\mathscr{I}} : (\mathrm{Sig}^{\mathscr{I}})^{\mathrm{op}} \to \mathrm{Class}$, where Class is the category of classes; and (iv) a family of *satisfaction relations* $\models^{\mathscr{I}}_{\Sigma} \subseteq \mathrm{Str}^{\mathscr{I}}(\Sigma) \times \mathit{Sen}^{\mathscr{I}}(\Sigma)$ indexed over $\Sigma \in |\mathrm{Sig}^{\mathscr{I}}|$, such that the following *satisfaction condition* holds for every signature morphism $\sigma : \Sigma \to \Sigma'$ in $\mathrm{Sig}^{\mathscr{I}}$, every sentence $\varphi \in \mathit{Sen}^{\mathscr{I}}(\Sigma)$, and every Σ'-structure $M' \in \mathrm{Str}^{\mathscr{I}}(\Sigma')$:

$$\mathrm{Str}^{\mathscr{I}}(\sigma)(M') \models^{\mathscr{I}}_{\Sigma} \varphi \;\Leftrightarrow\; M' \models^{\mathscr{I}}_{\Sigma'} \mathit{Sen}^{\mathscr{I}}(\sigma)(\varphi) \;.$$

$\mathrm{Str}^{\mathscr{I}}(\sigma)$ is called the *reduct* functor (also written $-|\sigma$), $\mathit{Sen}^{\mathscr{I}}(\sigma)$ the *translation* function (also written $\sigma(-)$).

A *theory* T in an institution consists of a signature Σ, written $\mathit{sig}(T)$, and a set of Σ-sentences; its structure class is the class of all Σ-structures satisfying the sentences.

[1] Structures are called *models* in [6]. We use the term *structure* (and, interchangeably, *realisation*) here in order to avoid confusion with the term *model* in the sense of model-driven engineering.

In the next sections, we formalise different UML diagram types as institutions. We will also need to relate institutions via so-called comorphisms. They formalise the intuition of translating or encoding an institution into another one.

An *institution comorphism* [7] $\rho : \mathscr{I} \to \mathscr{J}$ consists of a functor $\Phi : \mathrm{Sig}^{\mathscr{I}} \to \mathrm{Sig}^{\mathscr{J}}$, a natural transformation $\alpha : Sen^{\mathscr{I}} \dot{\to} \Phi\,;Sen^{\mathscr{J}}$, and a natural transformation $\beta : \Phi^{\mathrm{op}}\,;\mathrm{Str}^{\mathscr{I}} \dot{\to} \mathrm{Str}^{\mathscr{J}}$, such that for any $\Sigma \in |\mathrm{Sig}^{\mathscr{I}}|$, for any $\varphi \in Sen^{\mathscr{I}}(\Sigma)$, and any $M' \in \mathrm{Str}^{\mathscr{J}}(\Phi(\Sigma))$ the following satisfaction condition holds:

$$M' \models^{\mathscr{J}}_{\Phi(\Sigma)} \alpha_\Sigma(\varphi) \;\Leftrightarrow\; \beta_\Sigma(M') \models^{\mathscr{I}}_{\Sigma} \varphi\ .$$

By contrast, institution morphisms formalise the intuition of mapping a "richer" institution onto a "poorer" or more abstract one. Both morphisms and comorphisms also come in a "semi" variant (i.e., semi-morphisms and semi-comorphisms) [7]. These omit both the sentence translation and the satisfaction condition. Semi-(co-)morphisms can provide a model-theoretic link between institutions that are too different to permit a sentence translation, e.g., interactions and state machines. Here, we only need semi-morphisms.

An *institution semi-morphism* $\mu : \mathscr{I} \to \mathscr{J}$ consists of a functor $\Phi : \mathrm{Sig}^{\mathscr{I}} \to \mathrm{Sig}^{\mathscr{J}}$ and a natural transformation $\beta : \mathrm{Str}^{\mathscr{I}} \dot{\to} \Phi^{\mathrm{op}}\,;\mathrm{Str}^{\mathscr{J}}$.

3 An Institution for Simple UML State Machines

We recall our institution for UML simple (non-hierarchical) state machines from [11]; an institution also covering hierarchical states has been developed in [5]. We only need the so-called flat state machine institution. *Signatures* Σ consist of a set of actions $A(\Sigma)$, a set of messages $M(\Sigma)$, a set of variables $V(\Sigma)$, a set of (external) events $E(\Sigma)$, a set of completion events $F(\Sigma)$, and a set of states $S(\Sigma)$, such that $E(\Sigma) \cap F(\Sigma) = \emptyset = E(\Sigma) \cap S(\Sigma)$. *Signature morphisms* map signatures component-wise, where the maps for $E(\Sigma)$, $F(\Sigma)$ and $S(\Sigma)$ need to be injective.

▶ **Example 1.** Considering the UML state machine ATM in Fig. 1(c), its signature *ATMSig* will contain the actions userCom.ejectCard(); trialsNum = 0 and trialsNum++, as well as the messages userCom.ejectCard() and bankCom.markInvalid(cardId). Variables will include trialsNum. There are no internal events. States (and completion events) will include Idle, CardEntered, PINEntered, Verifying and Verified. ◀

Sentences are labelled transition systems (LTS) with states in $S(\Sigma)$ and labels in $(E(\Sigma) \cup F(\Sigma)) \times G(V(\Sigma)) \times A(\Sigma) \times \wp(F(\Sigma))$, where $G(V(\Sigma))$ is a set of guard sentences over $V(\Sigma)$. For example, the graphs of the state machines in Figs. 1(c) and 1(d) are such LTS.

▶ **Example 2.** Continuing the previous example for the state machine of Fig. 1(c) defining the behaviour of ATM, this state machine can be represented as the following sentence over this signature:

$$\begin{aligned}(\mathsf{Idle}, \{&\mathsf{Idle} \xrightarrow[T]{\mathsf{card(c)[true]/cardId\,=\,c},\emptyset} \mathsf{CardEntered},\\ &\mathsf{CardEntered} \xrightarrow[T]{\mathsf{PIN(p)[true]/pin\,=\,p},\mathsf{PINEntered}} \mathsf{PINEntered},\\ &\mathsf{PINEntered} \xrightarrow[T]{\mathsf{PINEntered[true]/bank.verify(cardId,\,pin)},\emptyset} \mathsf{Verifying},\\ &\mathsf{Verifying} \xrightarrow[T]{\mathsf{reenterPIN[trialsNum}\,<\,3\mathsf{]/trialsNum{+}{+}},\emptyset} \mathsf{CardEntered}, \ldots\})\ .\end{aligned}$$

In particular, PINEntered occurs both as a state and as a completion event to which the third transition reacts. The junction pseudostate for making the decision whether trialsNum $<$ 3 or trialsNum $>=$ 3 has been resolved by combining the transitions. ◀

Structures involve a set of configurations $Conf(\Sigma)$. A configuration consists of a valuation over variables in $V(\Sigma)$ in some value domain, an event pool consisting of a sequence events over $E(\Sigma) \cup F(\Sigma)$, and a state $s \in S(\Sigma)$. Structures are pairs $\Theta = (I_\Theta, \Delta_\Theta)$ where $I_\Theta \subseteq Conf(\Sigma)$ is the set of initial configurations, and $\Delta_\Theta \subseteq Conf(\Sigma) \times Lbl(\Sigma) \times Conf(\Sigma)$ is a labelled transition relation, for a suitable set of labels Lbl. In [11], we use sets of messages from $M(\Sigma)$ as labels, i.e., $Lbl(\Sigma) = \wp(M(\Sigma))$. The drawback is that such a modelling of state machines does not cater for external events being consumed by the state machine. As a consequence, the interleaving product construction in [11] has to deeply look into state machine configurations. By contrast, we here want to treat configurations as black boxes. We therefore choose the set of labels as

$$Lbl(\Sigma) = \{out(M') \mid M' \subseteq M(\Sigma)\} \cup \{in(M') \mid M' \subseteq M(\Sigma)\} \ .$$

That is, a transition can either emit a set of messages M', in the same way as in [11]. Alternatively, a transition can also consume a set of messages M', which is then just added to the event pool of the machine's configuration (state and variable valuation are not affected).

Satisfaction of a sentence (which essentially is a syntactic LTS) in a structure (which essentially is a semantic LTS, i.e. enriches states with variable valuations and event pools) means that the semantic LTS makes moves as prescribed by the syntactic LTS, if the respective guard evaluates to true, while simultaneously updating the variable valuation and the event pool. For details, see [11]. Consumed messages $in(M')$ lead to a slight change w.r.t. [11], the formalisation of which is obvious.

Altogether, this gives us a flat state machine institution **SM**. Flatness here refers to the staged construction of the state machine institution in [11], starting with institutions of actions and guards. The flat construction in [11] directly incorporates these institutions into the state machine institution.

4 An Institution for Simple UML Interactions

UML sequence diagrams are one variant of UML interactions that specify the communication between several components (state machines and also users) of a system. We here formalise a simplified version of sequence diagrams as an institution.

Signatures Σ consist of a set $L(\Sigma)$ of *lifelines* and a set $M(\Sigma)$ of *messages*. For simplicity, we do not consider any typing of lifelines or messages. If Σ and Σ' are signatures, then a *signature morphism* $\sigma : \Sigma \to \Sigma'$ consists of an injective function $L(\sigma) : L(\Sigma) \to L(\Sigma')$ and a function $M(\sigma) : M(\Sigma) \to M(\Sigma')$.

Sentences over the signature Σ are given by the following grammar:

$$\begin{aligned} F ::= {} & \mathsf{skip} \mid \mathsf{snd}(s, r, m) \mid \mathsf{snd}(s, m) \mid \mathsf{rcv}(s, r, m) \mid \mathsf{rcv}(r, m) \mid \\ & \mathsf{strict}(F_1, F_2) \mid \mathsf{seq}(F_1, F_2) \mid \mathsf{par}(F_1, F_2) \mid \mathsf{alt}(F_1, F_2) \end{aligned}$$

skip is the empty interaction. $\mathsf{snd}(s, r, m)$ denotes the event of lifeline $s \in L(\Sigma)$ sending message $m \in M(\Sigma)$ to lifeline $r \in L(\Sigma)$, whereas $\mathsf{snd}(s, m)$ denotes the sending from s to the environment. Conversely, $\mathsf{rcv}(s, r, m)$ is the event of r receiving message m from s, and $\mathsf{rcv}(r, m)$ the reception of m from the environment. $\mathsf{strict}(F_1, F_2)$ is strict sequencing of interactions, i.e., all events in F_1 must occur before those in F_2. $\mathsf{seq}(F_1, F_2)$ is weak

sequencing, only imposing the restriction that events keep their lifeline-wise order. $\mathsf{par}(F_1, F_2)$ allows for any parallel interleaving of F_1 and F_2. $\mathsf{alt}(F_1, F_2)$ chooses either F_1 or F_2. In this interaction sub-language we omit, in particular, guards, hiding by ignore and consider, and so-called negative behaviour as specified with neg and assert; a more comprehensive account is provided in [20]. Translation of a sentence by a signature morphism σ is the straightforward lifting of σ to the operators.

▶ **Example 3.** Consider the interaction in Fig. 1(e). Its signature *ScenarioSig* consists of two lifelines atm and bank, as well as the seven messages card(17), PIN(4711), verify(17, 4711), etc. The interaction is then directly expressed by the sentence

$$\begin{aligned}&\mathsf{seq}\big(\mathsf{rcv}(\mathsf{atm}, \mathsf{card}(17)),\\ &\quad \mathsf{seq}\big(\mathsf{rcv}(\mathsf{atm}, \mathsf{PIN}(4711)),\\ &\qquad \mathsf{seq}\big(\mathsf{strict}(\mathsf{snd}(\mathsf{atm}, \mathsf{bank}, \mathsf{verify}(17,\ 4711)), \mathsf{rcv}(\mathsf{atm}, \mathsf{bank}, \mathsf{verify}(17,\ 4711))),\\ &\qquad\quad \mathsf{seq}\big(\mathsf{strict}(\mathsf{snd}(\mathsf{bank}, \mathsf{atm}, \mathsf{reenterPIN}()), \mathsf{rcv}(\mathsf{bank}, \mathsf{atm}, \mathsf{reenterPIN}())),\\ &\qquad\qquad \mathsf{seq}\big(\mathsf{rcv}(\mathsf{atm}, \mathsf{PIN}(4242)), \dots\big)\big)\big)\big)\big)\end{aligned}$$

where UML's default composition mechanism is weak sequencing, and strict sequencing is used to express UML's requirement that a message must be sent before it can be received. ◀

Structures over a signature Σ involve traces of events. *Events* $\mathscr{E}(\Sigma)$ are either of the form $snd(s, r, m)$ ("object $s \in L(\Sigma)$ sends invocation $m \in M(\Sigma)$ to object $r \in L(\Sigma)$"), $snd(s, m)$ ("s sends m to the environment"), $rcv(s, r, m)$ ("r receives m from s"), or $rcv(r, m)$ ("r receives m from the environment"). An *event trace* $t \in \mathscr{E}(\Sigma)^*$ is a sequence of events, where $\langle\rangle$ denotes the empty sequence and $e :: t$ event concatenation. A Σ-*structure* is then a set of event traces $T \subseteq \mathscr{E}(\Sigma)^*$. A signature morphism $\sigma : \Sigma \to \Sigma'$ is applied to events by $\sigma(snd(s, r, m)) = snd(L(\sigma)(s), L(\sigma)(r), M(\sigma)(m))$ and similarly for $\sigma(snd(s, m))$, $\sigma(rcv(s, r, m))$, and $\sigma(rcv(r, m))$; and also to event traces by $\sigma(\langle\rangle) = \langle\rangle$ and $\sigma(e :: t) = \sigma(e) :: \sigma(t)$. The *reduct* of $T' \subseteq \mathscr{E}(\Sigma')^*$ along $\sigma : \Sigma \to \Sigma'$ is defined by taking the preimage of σ on event traces:

$$T'|\sigma = \{t \mid \sigma(t) \in T'\}\ .$$

In order to define the semantics of sentences, we need some auxiliary notions on events and traces: The set of lifelines $\alpha(e)$ *active* in an event e is defined as $\alpha(snd(s, r, m)) = \{s\} = \alpha(snd(s, m))$ and $\alpha(rcv(s, r, m)) = \{r\} = \alpha(rcv(r, m))$. Two events e_1 and e_2 are *in conflict*, written as $e_1 \asymp e_2$, if they share active lifelines, i.e., $\alpha(e_1) \cap \alpha(e_2) \neq \emptyset$. We use the following binary operations on traces yielding sets of traces, which we will also apply to sets of traces defining $T_1 \diamond T_2 = \bigcup\{t_1 \diamond t_2 \mid t_1 \in T_1,\ t_2 \in T_2\}$:

Strict sequencing In $t_1\,;\,t_2$, all events of t_1 must occur before all events of t_2:

$$\langle\rangle\,;\,t_2 = \{t_2\}\ ,\quad (e :: t_1)\,;\,t_2 = \{e :: t \mid t \in t_1\,;\,t_2\}$$

Weak sequencing In $t_1 \mathbin{;_{\not\asymp}} t_2$, events of t_1 may also occur after events of t_2. However, the ordering on lifelines must be preserved; this is ensured by the conflict condition $\neg(e_1 \asymp e_2)$:

$$\begin{aligned}&\langle\rangle \mathbin{;_{\not\asymp}} t_2 = \{t_2\}\ ,\qquad t_1 \mathbin{;_{\not\asymp}} \langle\rangle = \{t_1\}\ ,\\ &(e_1 :: t_1) \mathbin{;_{\not\asymp}} (e_2 :: t_2) = \{e_1 :: t \mid t \in t_1 \mathbin{;_{\not\asymp}} (e_2 :: t_2)\} \cup\\ &\qquad\qquad \{e_2 :: t \mid t \in (e_1 :: t_1) \mathbin{;_{\not\asymp}} t_2,\ \neg(e_1 \asymp e_2)\}\end{aligned}$$

Interleaving $t_1 \parallel t_2$ implements parallel interleaving of traces t_1 and t_2:

$$\begin{aligned}&\langle\rangle \parallel t_2 = \{t_2\}\ ,\qquad t_1 \parallel \langle\rangle = \{t_1\}\ ,\\ &(e_1 :: t_1) \parallel (e_2 :: t_2) = \{e_1 :: t \mid t \in t_1 \parallel (e_2 :: t_2)\} \cup \{e_2 :: t \mid t \in (e_1 :: t_1) \parallel t_2\}\end{aligned}$$

With these preliminaries, we are now ready to define the set of traces of a sentence:

$$\begin{aligned}
&\mathscr{P}(\mathsf{skip}) = \{\langle\rangle\} && \mathscr{P}(\mathsf{strict}(F_1, F_2)) = \mathscr{P}(F_1)\,;\,\mathscr{P}(F_2)\\
&\mathscr{P}(\mathsf{snd}(s, r, m)) = \{\langle snd(s, r, m)\rangle\} && \mathscr{P}(\mathsf{seq}(F_1, F_2)) = \mathscr{P}(F_1)\,;_{\gtrdot}\,\mathscr{P}(F_2)\\
&\mathscr{P}(\mathsf{snd}(s, m)) = \{\langle snd(s, m)\rangle\} && \mathscr{P}(\mathsf{par}(F_1, F_2)) = \mathscr{P}(F_1) \parallel \mathscr{P}(F_2)\\
&\mathscr{P}(\mathsf{rcv}(s, r, m)) = \{\langle rcv(s, r, m)\rangle\} && \mathscr{P}(\mathsf{alt}(F_1, F_2)) = \mathscr{P}(F_1) \cup \mathscr{P}(F_2)\\
&\mathscr{P}(\mathsf{rcv}(r, m)) = \{\langle rcv(r, m)\rangle\}
\end{aligned}$$

The *satisfaction relation* requires that a sentence is satisfied in a structure iff some of the sentence's traces occur in the structure:

$$T \models_\Sigma F \;\Leftrightarrow\; \mathscr{P}(F) \cap T \neq \emptyset\ .$$

In fact, injectivity of $L(\sigma)$ in signature morphisms is needed for ensuring the satisfaction condition: Consider Σ and Σ' with $L(\Sigma) = \{l_1, l_2, l_3, l_4\}$, $M(\Sigma) = \{m\} = M(\Sigma')$, $L(\Sigma') = \{l, l_3, l_4\}$, and $\sigma : \Sigma \to \Sigma'$ mapping l_1 and l_2 to l and the rest identically. Let $T' = \{\langle snd(l, l_4, m), snd(l, l_3, m)\rangle\}$ and $F = \mathsf{seq}(\mathsf{snd}(l_1, l_3, m), \mathsf{snd}(l_2, l_4, m))$. Then $T'|\sigma \models_\Sigma F$ (witnessed by the trace $\langle snd(l_2, l_4, m), snd(l_1, l_3, m)\rangle$), but $T' \not\models_{\Sigma'} \sigma(F)$ as $snd(l, l_4, m) \gtrdot snd(l, l_3, m)$ prohibiting the event order to be changed in the sequential composition.

To show the satisfaction condition for the injective case, we need a lemma:

▶ **Lemma 4.** *Let $\sigma : \Sigma \to \Sigma'$ be a signature morphism.*

1. $\mathscr{P}(\sigma(F)) \subseteq \sigma(\mathscr{E}(\Sigma)^*)$.
2. $\mathscr{P}(\sigma(F)) = \sigma(\mathscr{P}(F))$.

Proof. (1) By a straightforward induction on the structure of F.
(2) By induction on the structure of F. We only show the cases of skip, snd, and seq:

$$\begin{aligned}
&\mathscr{P}(\sigma(\mathsf{skip})) = \mathscr{P}(\mathsf{skip}) = \{\langle\rangle\} = \sigma(\{\langle\rangle\}) = \sigma(\mathscr{P}(\mathsf{skip}))\ ;\\
&\mathscr{P}(\sigma(\mathsf{snd}(l_s, l_r, m))) = \mathscr{P}(\mathsf{snd}(L(\sigma)(l_s), L(\sigma)(l_r), M(\sigma)(m))) =\\
&\qquad \{\langle snd(L(\sigma)(l_s), L(\sigma)(l_r), M(\sigma)(m))\rangle\} = \sigma(\mathscr{P}(\mathsf{snd}(l_s, l_r, m)))\ ;\\
&\mathscr{P}(\sigma(\mathsf{seq}(F_1, F_2))) = \mathscr{P}(\mathsf{seq}(\sigma(F_1), \sigma(F_2))) = \mathscr{P}(\sigma(F_1))\,;_{\gtrdot}\,\mathscr{P}(\sigma(F_2)) =\\
&\qquad \sigma(\mathscr{P}(F_1))\,;_{\gtrdot}\,\sigma(\mathscr{P}(F_2)) = \sigma(\mathscr{P}(F_1)\,;_{\gtrdot}\,\mathscr{P}(F_2)) = \sigma(\mathscr{P}(\mathsf{seq}(F_1, F_2)))\ ,
\end{aligned}$$

where injectivity of $L(\sigma)$ is needed for $\sigma(\mathscr{P}(F_1))\,;_{\gtrdot}\,\sigma(\mathscr{P}(F_2)) = \sigma(\mathscr{P}(F_1)\,;_{\gtrdot}\,\mathscr{P}(F_2))$. ◀

▶ **Proposition 5** (Satisfaction condition)**.** *Let $\sigma : \Sigma \to \Sigma'$ be a signature morphism and T' a structure over Σ'. Then $T'|\sigma \models_\Sigma F \Leftrightarrow T' \models_{\Sigma'} \sigma(F)$.*

Proof. We have $T'|\sigma \models_\Sigma F$ iff $\mathscr{P}(F) \cap \sigma^{-1}(T') \neq \emptyset$ iff (*) $\sigma(\mathscr{P}(F)) \cap \sigma(\sigma^{-1}(T')) \neq \emptyset$ iff (by Lem. 4(2)) $\mathscr{P}(\sigma(F)) \cap \sigma(\sigma^{-1}(T')) \neq \emptyset$ iff (**) $\mathscr{P}(\sigma(F)) \cap T' \neq \emptyset$ iff $T' \models_{\Sigma'} \sigma(F)$. Step (**) from left to right follows since $\sigma(\sigma^{-1}(T')) \subseteq T'$. For the converse direction, if $\mathscr{P}(\sigma(F)) \cap T' \neq \emptyset$, by Lem. 4(1), $\mathscr{P}(\sigma(F)) \cap \sigma(\mathscr{E}(\Sigma)^*) \cap T' \neq \emptyset$, hence $\mathscr{P}(\sigma(F)) \cap \sigma(\sigma^{-1}(T')) \neq \emptyset$. Step (*) from left to right is clear. From right to left, if $t_1 \in \mathscr{P}(F)$ and $t_2 \in \sigma^{-1}(T')$ with $\sigma(t_1) = \sigma(t_2)$, then $t_1 \in \mathscr{P}(F) \cap \sigma^{-1}(T')$. ◀

Altogether, this gives us an institution **SD** of sequence diagrams.

If one wants to drop the restriction of injectivity for signature morphisms, one could use so-called extended structures [21]. This would mean to start with signature morphisms without

injectivity restriction, which only leads to a so-called pre-institution (i.e., an institution without satisfaction condition). From a pre-institution, one can build an institution of extended structures. An extended structure over a signature Σ is a pair (σ, M') where $\sigma : \Sigma \to \Sigma'$ is a (pre-institution, here: possibly non-injective) signature morphism and M' is a Σ'-structure. The institution of extended structures always enjoys the satisfaction condition.

5 An Institution for Simple UML Composite Structures

A UML composite structure diagram specifies the linkage of component instances communicating through ports over connectors interlinking ports. We formalise a simplified (non-hierarchical) variant of composite structures here. All connectors are binary and each component instance is equipped with a state machine for describing its behaviour.

A composite structure *signature* Σ consists of a set of *components* $C(\Sigma)$; a *state machine assignment* $S(\Sigma) : C(\Sigma) \to |\mathbf{SM}\text{-Sig}|$, where **SM**-Sig is the category of signatures of the flat state machine institution (see Sect. 3); a set $P(\Sigma)$ of *ports* p each showing a component $c(p) \in C(\Sigma)$, a port name $n(p)$, and a set of messages $M(p)$; and a symmetric binary relation $\Gamma(\Sigma) \subseteq P(\Sigma) \times P(\Sigma)$ of *connectors* that connect ports, such that

- for each port $p \in P(\Sigma)$ and each message $m \in M(p)$, the prefixed message $n(p).m$ is a message of component c's state machine, i.e., $n(p).m \in M(S(\Sigma)(c))$, and
- for each connector $(p_1, p_2) \in \Gamma(\Sigma)$, $M(p_1) = M(p_2)$, i.e., messages are fully compatible.

We say that port $p \in P(\Sigma)$ is *open* in $\Gamma(\Sigma)$ if there is no $p' \in P(\Sigma)$ such that $(p, p') \in \Gamma(\Sigma)$; otherwise p is *connected*.

Note that we consider the message components $M(S(\Sigma)(c))$ of the state machine signatures only. The event components are not relevant here, because they refer to internal events, and these cannot be sent over a connection.

▶ **Example 6.** Consider the composite structure in Fig. 1(a). Its signature *SystemSig* features two components atm and bank. The state machine signature $S(\mathit{SystemSig})(\mathsf{atm})$ is that of Ex. 1. The set of ports is $P(\mathit{SystemSig}) = \{\mathit{atmCom}, \mathit{bankCom}\}$ with $c(\mathit{bankCom}) = \mathsf{atm}$, $c(\mathit{atmCom}) = \mathsf{bank}$, $n(\mathit{atmCom}) = \mathsf{atmCom}$, $n(\mathit{bankCom}) = \mathsf{bankCom}$, and $M(\mathit{atmCom}) = \{\mathsf{verify}(-,-), \mathsf{reenterPIN}(), \mathsf{verified}(), \mathsf{markInvalid}(-)\} = M(\mathit{bankCom})$. The placeholders $-$ denote possible arguments, leading to an infinite set of possible messages. The set of connectors is $\Gamma(\mathit{SystemSig}) = \{(\mathit{atmCom}, \mathit{bankCom}), (\mathit{bankCom}, \mathit{atmCom})\}$.

The first condition on signatures means that the involved state machines must be able to communicate all port messages by prefixing with the port name. For example, the bank state machine in Fig. 1(d) can emit a message atmCom.verified, and the ATM state machine in Fig. 1(c) can receive a message bankCom.verified as trigger. Both messages are prefixed with the port name. Inside the port, the message is just verified in both cases. The second condition on signatures ensures that all such shared messages can be sent over the connector. ◀

A *signature morphism* $\sigma : \Sigma \to \Sigma'$ consists of a function $C(\sigma) : C(\Sigma) \to C(\Sigma')$ mapping components; for each $c \in C(\Sigma)$, a signature morphism $S(\sigma)(c) : S(\Sigma)(c) \to S(\Sigma')(C(\sigma)(c)) \in \mathbf{SM}\text{-Sig}$ interfacing the state machine signatures; a function $P(\sigma) : P(\Sigma) \to P(\Sigma')$ mapping ports; and a function $M(\sigma) : \bigcup\{M(p) \mid p \in P(\Sigma)\} \to \bigcup\{M(p') \mid p' \in P(\Sigma')\}$ mapping messages, such that

- for each port $p \in P(\Sigma)$, $M(\sigma)(M(p)) \subseteq M(P(\sigma)(p))$, that is, $M(\sigma)$ restricts to the port appropriately, and
- $(P(\sigma) \times P(\sigma))(\Gamma(\Sigma)) \subseteq \Gamma(\Sigma')$, i.e., connectors are preserved.

Sentences are pairs (c, φ) with $c \in C$ and $\varphi \in \mathbf{SM}\text{-Sen}(S(\Sigma)(c))$. Thus, on any component, we can impose state machine sentences over the signature of that component. Sentence translation is defined by $\sigma(c, \varphi) = (C(\sigma)(c), S(\sigma)(c)(\varphi))$.

Structures are families $R = (R(c) \in \mathbf{SM}\text{-Str}(S(\Sigma)(c)))_{c \in C(\Sigma)}$ of state machine structures. Given a signature morphism $\sigma : \Sigma \to \Sigma'$ and a Σ'-structure $R' = (R'(c') \in \mathbf{SM}\text{-Str}(S(\Sigma')(c')))_{c' \in C(\Sigma')}$, its *$\sigma$-reduct* is

$$R'|\sigma = (R'(C(\sigma)(c))|S(\sigma)(c))_{c \in C(\Sigma)} \ .$$

Satisfaction is defined by selecting the appropriate component:

$$R \models_\Sigma (c, \varphi) \ \Leftrightarrow \ R(c) \models^{\mathbf{SM}}_{S(\Sigma)(c)} \varphi \ .$$

Also, the satisfaction condition holds:

▶ **Proposition 7.** *Let $\sigma : \Sigma \to \Sigma'$ be a signature morphism and R' a structure over Σ'. Then $R'|\sigma \models_\Sigma (c, \varphi) \Leftrightarrow R' \models_{\Sigma'} \sigma(c, \varphi)$.*

Proof. $R'|\sigma \models_\Sigma (c, \varphi)$ iff $R'(C(\sigma)(c))|S(\sigma)(c) \models^{\mathbf{SM}}_{S(\Sigma)(c)} \varphi$ iff (by the satisfaction condition in **SM**) $R'(C(\sigma)(c)) \models^{\mathbf{SM}}_{S(\Sigma')(C(\sigma)(c))} S(\sigma)(c)(\varphi)$ iff $R' \models_{\Sigma'} \sigma(c, \varphi)$. ◀

Altogether, this gives us an institution **CMP** of composite structure diagrams.

Note that ports and connectors do not play a role for the satisfaction relation. They will play a role in the interaction with other institutions, the topic of the next section.

6 Putting It All Together

Our ultimate goal is the formalisation of the question whether the traces of an interaction diagram are realisable in a system of state machines, interlinked by a composite structure diagram. We will now introduce the institution translations necessary for reaching this goal.

6.1 Comorphism SM → CMP

We introduce a comorphism **SM** → **CMP** that can be used to construe a single state machine as a (singleton) composite structure. It is a trivial embedding and acts as follows:

Signatures $\Sigma_{\mathbf{SM}} \mapsto \Sigma_{\mathbf{CMP}}$ with $C(\Sigma_{\mathbf{CMP}}) = \{cid\}$, $S(\Sigma_{\mathbf{CMP}})(cid) = \Sigma_{\mathbf{SM}}$, $P(\Sigma_{\mathbf{CMP}}) = \emptyset$, and $\Gamma(\Sigma_{\mathbf{CMP}}) = \emptyset$
Sentences $\varphi_{\mathbf{SM}} \mapsto (cid, \varphi_{\mathbf{SM}})$
Realisations $R_{\mathbf{CMP}} \mapsto R_{\mathbf{CMP}}(cid)$

The satisfaction condition follows trivially.

Note that the resulting composite structure will name the single component *cid*. This can be changed with a renaming along a signature morphism.

6.2 Semi-morphism CMP → SD

Since interactions and composite structures have very different sentences, all we can hope for is to relate them via an institution semi-morphism **CMP** → **SD**. On signatures, it construes components as lifelines, and it assembles all possible messages that can be sent via the connectors. These are messages that are available in the state machines of the components, but prefixed with the respective port name. For example, the bank state machine in Fig. 1(d)

can emit a message atmCom.verified, and the ATM state machine in Fig. 1(c) can receive a message bankCom.verified as trigger. In the interaction of Fig. 1(e), there is a message verified that corresponds to the message atmCom.verified sent over the connector and being received as bankCom.verified. Hence, the message verified will appear in the resulting interaction signature.

More formally, a composite structure signature $\Sigma_{\mathbf{CMP}}$ is mapped to the interaction signature $\Phi(\Sigma_{\mathbf{CMP}})$ with $L(\Phi(\Sigma_{\mathbf{CMP}})) = C$ and

$$M(\Phi(\Sigma_{\mathbf{CMP}})) = \bigcup\{M(p) \mid p \in P(\Sigma_{\mathbf{CMP}})\}$$

A signature morphism $\sigma_{\mathbf{CMP}} : \Sigma_{\mathbf{CMP}} \to \Sigma'_{\mathbf{CMP}}$ is mapped to $\Phi(\sigma_{\mathbf{CMP}}) : \Phi(\Sigma_{\mathbf{CMP}}) \to \Phi(\Sigma'_{\mathbf{CMP}})$ with $L(\Phi(\sigma_{\mathbf{CMP}})) = C(\sigma_{\mathbf{CMP}})$ and $M(\Phi(\sigma_{\mathbf{CMP}})) = M(\sigma_{\mathbf{CMP}})$.

Concerning structures, note that a structure in **CMP** is a family of state machine structures: Form the interleaving product of these, and take the traces over the interleaved product. This is defined as a set of the message sequences of all possible runs in the labelled transition system, starting with an initial state. This gives an interaction structure.

We will now make this construction more precise. Note that we do not use the interleaved product construction from [11], because it is not abstract enough. That construction assumes that variable sets are shared (i.e., "shared memory"), and it moreover looks into the configurations of the state machines, e.g., in order to inject events into their event pools. By contrast, we here take a black-box view on state machines, and only use labels on transitions for communication among state machines. As a side effect, we also overcome the restriction made in [11] that state sets of the involved state machines must be disjoint, as well as their event sets. Moreover, a major difference is that in [11] we have used shared message names for communication, while here, communication happens only via explicit connectors.

Given a $\Sigma_{\mathbf{CMP}}$-structure R in **CMP**, we construct an LTS $\Pi(R)$ representing the interleaved product as follows: Let us abbreviate $C(\Sigma_{\mathbf{CMP}})$ by $C_{\mathbf{CMP}}$, $S(\Sigma_{\mathbf{CMP}})$ by $S_{\mathbf{CMP}}$, and $\Gamma(\Sigma_{\mathbf{CMP}})$ by $\Gamma_{\mathbf{CMP}}$. Let $R(c) = (I_{R_c}, \Delta_{R_c})$ for each $c \in C_{\mathbf{CMP}}$. The state space of $\Pi(R)$ is given by $\prod_{c \in C_{\mathbf{CMP}}} Conf(S_{\mathbf{CMP}}(c))$. Its set of initial states is given by $\prod_{c \in C_{\mathbf{CMP}}} I_{R_c}$. The transitions are given by

$$(s_c)_{c \in C_{\mathbf{CMP}}} \xrightarrow[\Pi(R)]{E} (s'_c)_{c \in C_{\mathbf{CMP}}}$$

if there is a component $\hat{c} \in C_{\mathbf{CMP}}$ such that

- either $s_{\hat{c}} \xrightarrow[\Delta_{R_{\hat{c}}}]{in(M_{\hat{c}})} s'_{\hat{c}}$ with all ports in $\{p \mid c(p) = \hat{c},\ n(p).m \in M_{\hat{c}}\}$ open in $\Gamma_{\mathbf{CMP}}$, $E = \{rcv(\hat{c}, m) \mid n.m \in M_{\hat{c}}\}$, and $s_c \xrightarrow[\Delta_{R_c}]{out(\emptyset)} s'_c$ or $s_c = s'_c$ for all $c \in C_{\mathbf{CMP}} \setminus \{\hat{c}\}$; i.e., component $\hat{c}$ receives input from the environment through its open ports and all other components make only internal steps;
- or $s_{\hat{c}} \xrightarrow[\Delta_{R_{\hat{c}}}]{out(M'_{\hat{c}})} s'_{\hat{c}}$ with $\{p \mid c(p) = \hat{c},\ n(p).m \in M_{\hat{c}}\} = P_0 \cup P_1$ such that all ports in P_0 are open and all ports in P_1 connected in $\Gamma_{\mathbf{CMP}}$, $E = \{snd(\hat{c}, m) \mid p_0 \in P_0,\ n(p_0).m \in M_{\hat{c}}\} \cup \{snd(\hat{c}, c(p_2), m), rcv(\hat{c}, c(p_2), m) \mid p_1 \in P_1,\ (p_1, p_2) \in \Gamma_{\mathbf{CMP}},\ n(p_1).m \in M'_{\hat{c}}\}$, $s_{c(p_2)} \xrightarrow[\Delta_{R_{c(p_2)}}]{in(M_{c(p_2)})} s'_{c(p_2)}$ with $M_{c(p_2)} = \{n(p_2).m \mid n(p_1).m \in M_{\hat{c}}\}$ for each $(p_1, p_2) \in \Gamma_{\mathbf{CMP}}$ and $p_1 \in P_1$, and $s_c \xrightarrow[\Delta_{R_c}]{out(\emptyset)} s'_c$ or $s_c = s'_c$ for $c \in C_{\mathbf{CMP}} \setminus (\{\hat{c}\} \cup \{c(p_2) \mid p_1 \in P_1,\ (p_1, p_2) \in \Gamma_{\mathbf{CMP}}\})$; i.e., component $\hat{c}$ sends messages which are received either by connected partners or the environment, and all other components not participating in the communication make only internal steps.

Based on this, the set of traces $Tr(\Pi(R))$ of $\Pi(R)$ is defined as follows: A *finite run* in $\Pi(R)$ is an alternating sequence $s_0\, E_1\, s_1\, \ldots\, E_n\, s_n$ with $s_0 \in I$ and $s_{i-1} \xrightarrow[\Pi(R)]{E_i} s_i$ for $i = 1, \ldots, n$. In this case, any $\overline{E}_1\, \ldots\, \overline{E}_n$ is a *trace* of $\Pi(R)$, where for $i = 1, \ldots, n$, $\overline{E}_i$ is a sequence with the same elements as the set E_i and all *snd*-events occur before all *rcv*-events.

$Tr(\Pi(R))$ is the translation of the structure R by the semi-morphism.

▶ **Example 8.** Consider the composite structure diagram in Fig. 1(a), showing instances atm and bank of the ATM and Bank components, respectively, that are connected through their bankCom and atmCom ports. In execution, atm and bank will exchange messages, as prescribed by their state machines, and this exchange is reflected by the interleaving product. Messages atmCom.verified (sent by the bank state machine) and bankCom.verified (received by the ATM state machine) will be unified to message verified in the interleaving product LTS and in its traces. ◀

6.3 Are Interactions Realisable?

Our initial question was whether the traces of an interaction diagram realisable in a system of state machines, interlinked by a composite structure diagram. This question can now be formalised as follows: The system of state machines is formalised as a theory in the institution **CMP**, using the comorphism $\mathbf{SM} \to \mathbf{CMP}$ to inject individual state machines into the system. This theory typically has a unique structure (because state machine theories have a unique structure). The structure of this theory in **CMP** can be translated along the semi-morphism $\mathbf{CMP} \to \mathbf{SD}$, resulting in a set of traces. If this intersects with the traces of the interaction, the interaction is realisable in the system of state machines.

This verification condition can also be expressed in the Distributed Ontology, Model and Specification Language (DOL) [18, 14][2], which recently has been adopted as a standard by the Object Management Group (OMG). UML diagrams can be referenced in DOL as-is using the standard interchange format XMI without the need of an encoding into some other language. The system of two state machines linked by a composite structure diagram can be expressed as follows, using the institution comorphism `sm2cmp` : $\mathbf{SM} \to \mathbf{CMP}$ introduced in Sect. 6.1 above:

```
model System =
  ATM_behaviour with translation sm2cmp with cid |-> atm
and
  Bank_behaviour with translation sm2cmp with cid |-> bank
then
  cmp
end
```

Recall that `sm2cmp` provides exactly one component with name `cid`. Using renaming in DOL, we can create two different components of the composite structure diagram, which are joined by a DOL union. The part `cmp` contains the specification of ports and connectors from the UML diagram.

We now express that this system can realise the interactions expressed in sequence diagram `ATM2Bank_Scenario` as a refinement in DOL:

```
refinement r2 =
  ATM2Bank_Scenario refined to { System hide along cmp2sd }
end
```

[2] See also `http://www.omg.org/spec/DOL/` and `http://dol-omg.org`.

Here, `cmp2sd` : **CMP** → **SD** is the institution semi-morphism introduced in Sect. 6.2. The semantics of the DOL refinement is just structure class inclusion. This means that each structure of `System hide along cmp2sd` must be a structure of `ATM2Bank_Scenario`. Now a structure of `System` is the (typically unique) family of semantic transition systems for the involved state machines (here `atm` and `bank`). `hide along cmp2sd` invokes translation along the institution semi-morphism `cmp2sd`. The result is the set of traces in the interleaved product. The requirement is now that this is a structure of `ATM2Bank_Scenario`, which means that at least one trace of the interleaved product must be a trace of `ATM2Bank_Scenario`. Hence, this exactly captures the condition that the interaction is realisable in the system of state machines as specified by the composite structure diagram.

▶ **Example 9.** The interaction in Fig. 1(e) can be realised by the composite structure in Fig. 1(a) that connects the state machines in Fig. 1(d) and Fig. 1(c). ◀

During a typical development process, UML diagrams are refined and more details are added. We have

▶ **Proposition 10.** *Realisability of interactions is preserved when the composite structure diagram (with its state machines) is refined.*

Proof. By the above discussion, realisability of interactions can be expressed as a refinement. Moreover, refinements compose. ◀

Note that realisability of interactions is in general *not* preserved when the interaction itself is refined.

7 Conclusions

We have presented an important step in our program of formalising families of UML diagrams using institutions. This formalisation makes UML diagrams ready for use with the OMG-standardised Distributed Ontology, Model and Specification Language (DOL), which for example can express refinements. Our initial question whether interaction is realisable in the system of state machines as specified by the composite structure diagram can be indeed expressed as a DOL refinement. Such meta relations between UML models are normally expressed in an ad-hoc way. With our approach, meta relations can be expressed in DOL, a formal language with a formal, institution-based semantics.

Related approaches to semantics of UML composite structures are [19, 1, 16]. While [1] only consider static semantics, [19, 16] also consider behaviour. However, [19] do cover neither interaction diagrams nor composite structures explicitly (they use some ad-hoc sequence of actions as well as parallel composition in Object-Z), and while [16] treat composite structures in great detail (also with some operational semantics, which however is not explicated in the paper), they do not consider interactions.

Related approaches formalising connectors such as [4] use a more category-theoretic notation and approach. While this is mathematically more elegant, our approach has the advantage of supporting UML diagrams as they are, without any need of encoding. Moreover, since we support signature morphisms, DOL's modularity constructs (which include constructs for networks of models[3] and their colimit) can be fully used also for UML diagrams.

[3] Technically, a network of (UML) models is a diagram (in the sense of category theory) of logical theories that possibly live in different institutions.

Note that in the sense of [8], our approach is one of asynchronously communicating components. This is because each UML state machine is equipped with its own event pool, which acts as a communication buffer. It should be possible to use the results of [8] for studying (weak) asynchronous compatibility of UML composite structures. This notion ensures that all messages that are sent by some state machine are indeed accepted by another one. Note however that UML has a very simple mechanism to avoid deadlocks of communicating state machines: messages that cannot be processed by a state machine are simply dropped from the event buffer (this is also built-in in our state machine institution). Still, it may be interesting to know whether such situations actually happen or not.

Another important piece of future work is the provision of tool support. The tool Hugo/RT allows checking realisability of interactions for a set of communicating UML state machines, see our previous work [12]. There, a restricted subset of UML interactions is translated into a kind of Büchi automata. While UML composite structures are not considered there, it should be possible to realise our interleaved product construction in Hugo/RT, such that the verification conditions studied in the present paper can be checked as well. The Heterogeneous Tool Set (Hets [15]) provides analysis and proof support for multi-logic specifications in DOL, based on a strong semantic (institution-based) backbone. With Hets, also refinements can be specified and checked. And indeed, Hets can already read in and process XMI files, OMG's interchange format for UML diagrams. Our ultimate goal is to provide a complete integration of a family of UML diagrams into the DOL/Hets ecosystem, and thus to provide tools for UML that are currently not available (see the extensive discussion in [9]): namely comprehensive semantically-grounded support for checking consistency and verification conditions of multi-view UML diagrams.

References

1 Umesh Bellur and V. Vallieswaran. On OO Design Consistency in Iterative Development. In *Proc. 3rd Intl. Conf. Information Technology: New Generations (ITNG'06)*, pages 46–51. IEEE, 2006.

2 Mihai Codescu, Till Mossakowski, Don Sannella, and Andrzej Tarlecki. Specification Refinements: Calculi, Tools, and Applications. *Sci. Comp. Prog.*, 2017. To appear.

3 Gregor Engels, Reiko Heckel, and Jochen Malte Küster. The Consistency Workbench: A Tool for Consistency Management in UML-Based Development. In Perdita Stevens, Jon Whittle, and Grady Booch, editors, *Proc. 6th Intl. Conf. Unified Modeling Language (UML'03)*, volume 2863 of *Lect Notes Comp. Sci.* Springer, 2003.

4 José Luiz Fiadeiro. *Categories for Software Engineering.* Springer, 2005.

5 Martin Glauer. *Institution for Hierarchical UML State Machines.* Master thesis, Otto-von-Guericke-Universität Magdeburg, 2015.

6 Joseph A. Goguen and Rod M. Burstall. Institutions: Abstract Model Theory for Specification and Programming. *J. ACM*, 39:95–146, 1992.

7 Joseph A. Goguen and Grigore Roşu. Institution Morphisms. *Formal Asp. Comp.*, 13:274–307, 2002.

8 Rolf Hennicker, Michel Bidoit, and Thanh-Son Dang. On Synchronous and Asynchronous Compatibility of Communicating Components. In Alberto Lluch-Lafuente and José Proença, editors, *Proc. 18th IFIP WG 6.1 Intl. Conf. Coordination Models and Languages (COORDINATION'16)*, volume 9686 of *Lect. Notes Comp. Sci.*, pages 138–156. Springer, 2016.

9 Alexander Knapp and Till Mossakowski. Multi-view Consistency in UML, 2016. URL: `https://arxiv.org/abs/1610.03960`.

10 Alexander Knapp, Till Mossakowski, and Markus Roggenbach. An Institutional Framework for Heterogeneous Formal Development in UML. A Position Paper. In Rocco De Nicola and Rolf Hennicker, editors, *Software, Services, and Systems. Essays Dedicated to Martin Wirsing on the Occasion of His Retirement from the Chair of Programming and Software Engineering*, volume 8950 of *Lect. Notes Comp. Sci.*, pages 215–230. Springer, 2015.

11 Alexander Knapp, Till Mossakowski, Markus Roggenbach, and Martin Glauer. An Institution for Simple UML State Machines. In Alexander Egyed and Ina Schaefer, editors, *Proc. 18th Intl. Conf. Fundamental Approaches to Software Engineering (FASE'15)*, volume 9033 of *Lect. Notes Comp. Sci.*, pages 3–18. Springer, 2015.

12 Alexander Knapp and Jochen Wuttke. Model Checking of UML 2.0 Interactions. In Thomas Kühne, editor, *Reports Rev. Sel. Papers Ws.s Symp.s MoDELS 2006*, volume 4364 of *Lect. Notes Comp. Sci.*, pages 42–51. Springer, 2007.

13 Kevin Lano, editor. *UML 2 — Semantics and Applications*. Wiley, 2009.

14 Till Mossakowski, Mihai Codescu, Fabian Neuhaus, and Oliver Kutz. The Distributed Ontology, Modelling and Specification Language — DOL. In Arnold Koslow and Arthur Buchsbaum, editors, *The Road to Universal Logic — Festschrift for the 50th Birthday of Jean-Yves Beziau, vol. II*, Studies in Universal Logic. Birkhäuser, 2015.

15 Till Mossakowski, Christian Maeder, and Klaus Lüttich. The Heterogeneous Tool Set. In Orna Grumberg and Michael Huth, editors, *Proc. 13th Intl. 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.

16 Iulian Ober and Iulia Dragomir. Unambiguous UML Composite Structures: The OMEGA2 Experience. In Ivana Cerná, Tibor Gyimóthy, Juraj Hromkovic, Keith G. Jeffery, Rastislav Královic, Marko Vukolic, and Stefan Wolf, editors, *Proc. 37th Conf. Current Trends in Theory and Practice of Computer Science (SOFSEM'11)*, volume 6543 of *Lect. Notes Comp. Sci.*, pages 418–430. Springer, 2011.

17 Object Management Group. Unified Modeling Language 2.5. Standard formal/2015-03-01, OMG, 2015. URL: `http://www.omg.org/spec/UML/2.5`.

18 Object Management Group. Distributed Ontology, Modeling, and Specification Language (DOL) 1.0 - Beta. Standard ptc/2016-02-37, OMG, 2016. URL: `http://www.omg.org/spec/DOL/`.

19 Holger Rasch and Heike Wehrheim. Checking Consistency in UML Diagrams: Classes and State Machines. In Elie Najm, Uwe Nestmann, and Perdita Stevens, editors, *Proc. 6th IFIP WG 6.1 Intl. Conf. Formal Methods for Open Object-Based Distributed Systems (FMOODS'03)*, volume 2884 of *Lect. Notes Comp. Sci.*, pages 229–243. Springer, 2003.

20 Tobias Rosenberger. *Relating UML State Machines and Interactions in an Institutional Framework*. Master thesis, Universität Augsburg, 2017.

21 Lutz Schröder, Till Mossakowski, and Christoph Lüth. Type Class Polymorphism in an Institutional Framework. In José Luiz Fiadeiro, editor, *Rev. Sel. Papers 17th Intl. Ws. Recent Trends in Algebraic Development Techniques (WADT'04)*, volume 3423 of *Lect. Notes Comp. Sci.*, pages 234–248. Springer, 2005.