An Institution for Simple UML State Machines

Alexander Knapp[1], Till Mossakowski[2], Markus Roggenbach[3], and Martin Glauer[2]

[1] Universität Augsburg, Germany
[2] Otto-von-Guericke Universität Magdeburg, Germany
[3] Swansea University, UK

Abstract. We present an institution for UML state machines without hierarchical states. The interaction with UML class diagrams is handled via institutions for guards and actions, which provide dynamic components of states (such as valuations of attributes) but abstract away from details of class diagrams. We also study a notion of interleaving product, which captures the interaction of several state machines. The interleaving product construction is the basis for a semantics of composite structure diagrams, which can be used to specify the interaction of state machines. This work is part of a larger effort to build a framework for formal software development with UML, based on a heterogeneous approach using institutions.

Keywords: UML, state machines, interleaving product, institutions.

1 Introduction

The "Unified Modeling Language" (UML [1]) 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 [2], we have outlined our research programme of "institutionalising UML". Our objective is to give, based on the theory of institutions [3], 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. In this paper, we propose a new institution for UML state machines.

Behavioural UML state machines specify the behaviour of model elements, like components, whereas *protocol* UML state machines express usage protocols, like the message exchange over a connector between components. Both variants describe dynamical system behaviour in terms of action effects and messages, where conditions are used to choose between different possibilities of the behaviour. We tackle the well-known resulting problem of integrating specifications of data (i.e., action effects and messages), logic (i.e., conditions), and processes (i.e., state machines) [4,5,6,7] by a two-step semantics: In the first step, we define *institutions of guards and actions* that capture which guards, actions, and messages can be used in order to define a state machine. In general, other UML diagrams like class diagrams or OCL constraints specify these items, i.e., define a suitable environment. In a second step, we then define institutions for behavioural and protocol state machines relative to given institutions of guards and actions. However, currently both of our institutions are restricted to "flat", non-hierarchical state machines; in fact, most of the hierarchical features can be reduced to this format [8,9]. A previous UML state machine institution by D. Calegari and N. Szasz [10] encoded all these features on a single (signature) level thus reducing integration flexibility considerably; furthermore, it only comprised behavioural state machines and captured each state machine in isolation. By contrast, we study interacting state machines and the refinement of state machines.

Our institution of behavioural state machines has the peculiarity of being a "programming language-like" institution, in the sense that each sentence essentially has one model, its canonical model. By contrast, our institution of protocol state machines is a "loose semantics" institution where generally a sentence has many models. For system development, we introduce an interleaving product of several state machines in our institution, which allows us to consider refinement for checking the correct implementation of protocols and which ideally could be integrated into the current efforts for providing precise semantics for UML composite structures [11]. Furthermore, we consider the determinism of state machines to foster code generation [12].

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 behavioural and protocol UML state machines. In Sect. 3 we define institutions for these variants of state machines. We study a notion of determinism for state machines, their interleaving, and their refinement based on the institutions in Sect. 4. Finally, in Sect. 5 we conclude with an outlook to future work.

2 Heterogeneous Institution-Based UML Semantics

The work in this paper is part of a larger effort [2] of giving an institution-based heterogeneous semantics to several UML diagrams as shown in Fig. 1. 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 [3]. 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

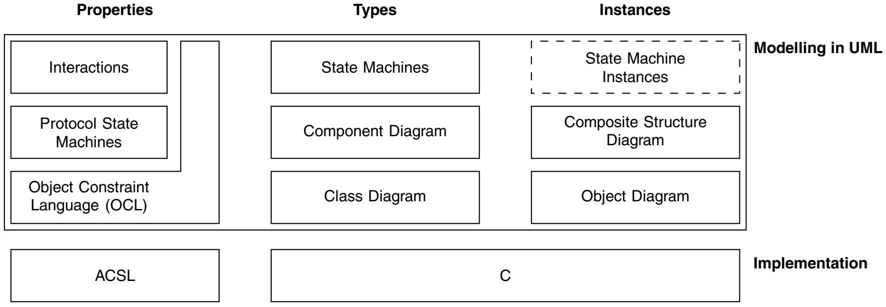


Fig. 1. Languages and diagrams to be considered

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 [14]) 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 [15,16]. 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, models and satisfaction. Signatures provide the vocabulary that may appear in sentences and that is interpreted in models. The satisfaction relation determines whether a given sentence is satisfied in a given model. The exact nature of signatures, sentences and models 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 [3], an institution $\mathscr{I} = (\mathrm{Sig}^{\mathscr{I}}, \mathit{Sen}^{\mathscr{I}}, \mathrm{Mod}^{\mathscr{I}}, \models^{\mathscr{I}})$ 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 *model functor* $\mathrm{Mod}^{\mathscr{I}} : (\mathrm{Sig}^{\mathscr{I}})^{\mathrm{op}} \to \mathrm{Class}$, where Class is the category of classes; and (iv) a family of *satisfaction relations* $\models^{\mathscr{I}}_{\Sigma} \subseteq \mathrm{Mod}^{\mathscr{I}}(\Sigma) \times \mathit{Sen}^{\mathscr{I}}(\Sigma)$ 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 for every Σ'-model $M' \in \mathrm{Mod}^{\mathscr{I}}(\Sigma')$:

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

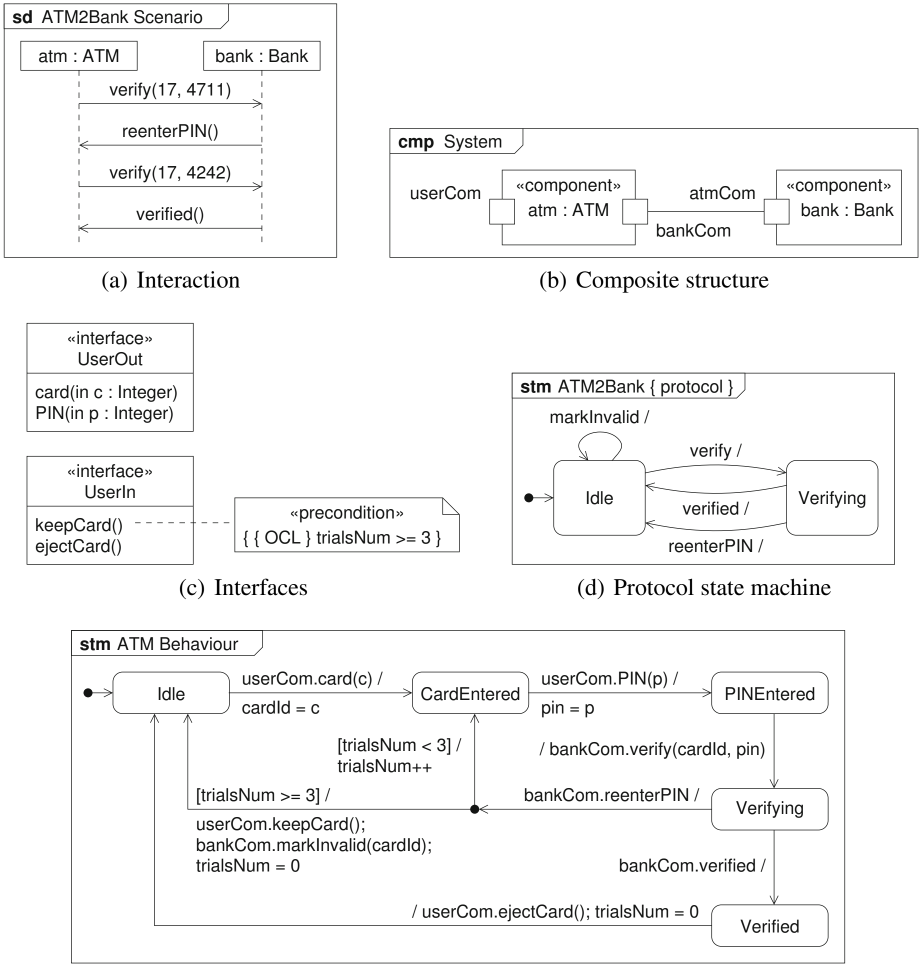
$\mathrm{Mod}^{\mathscr{I}}(\sigma)$ is called the *reduct* functor (also written $-|\sigma$), $\mathit{Sen}^{\mathscr{I}}(\sigma)$ the *translation* function (also written $\sigma(-)$).

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

An institution $\mathscr{I}$ has the *weak amalgamation property* for a pushout

$$\begin{tikzcd}
\Sigma \arrow[r] \arrow[d] & \Sigma_1 \arrow[d] \\
\Sigma_2 \arrow[r] & \Sigma_R
\end{tikzcd}$$

if any pair $(M_1, M_2) \in \mathrm{Mod}^{\mathscr{I}}(\Sigma_1) \times \mathrm{Mod}^{\mathscr{I}}(\Sigma_2)$ that is *compatible* in the sense that M_1 and M_2 reduce to the same Σ-model can be *amalgamated* to a Σ_R-model M_R (i.e., there exists a $M_R \in \mathrm{Mod}^{\mathscr{I}}(\Sigma_R)$ that reduces to M_1 and M_2, respectively). Weak amalgamation allows the computation of normal forms for specifications [17], and implies good behaviour w.r.t. conservative extensions, as well as soundness of proof systems for structured specifications [18].



(a) Interaction

(b) Composite structure

(c) Interfaces

(d) Protocol state machine

(e) State machine

Fig. 2. ATM example

2.2 ATM Example

In order to illustrate our approach to a heterogeneous institutions-based UML semantics in general and the institutions for UML state machines in particular, 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 2(a) shows a possible *interaction* between an atm and a bank, which consists out of four messages: the atm requests the bank to verify if a card and PIN number combination is valid, in the first case the bank requests to reenter the PIN, in the second case the verification is successful.

The composite structure of the ATM-bank system is specified in the *component diagram* in Fig. 2(b). 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 2(c) 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 the OCL constraint trialsNum >= 3, which refines its semantics: keepCard can only be invoked if the OCL constraints holds.

The communication protocol on this connector is captured with a *protocol state machine*, see Fig. 2(d). The protocol state machine fixes in which order the messages verify, verified, reenterPIN, and markInvalid between atm and bank may occur.

The dynamic behaviour of the atm component is specified by the *behavioural state machine* shown in Fig. 2(e). 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 the cardId in the atm object (parameter names are not shown on triggers). 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.

Questions on the model. Given the above diagrams specifying *one* system, the question arises if they actually “fit” together. Especially, one might ask if the diagrams are consistent, and if the different levels of abstraction refine each other. In our ATM example we have:

Example 1 (Consistency). The interface in Fig. 2(c) requires the operation keepCard only to be invoked when the precondition trialsNum >= 3 holds. This property holds for the state machine in Fig. 2(e) thanks to the guard trialsNum < 3.

Example 2 (Refinement). As the only trace of the interaction in Fig. 2(a) is a possible run of the state machine in Fig. 2(e), the interaction refines to the state machine.

Example 3 (Refinement). Similarly, we can consider if the protocol state machine in Fig. 2(d) refines to the product of the state machine of the atm, shown in Fig. 2(e), and of the bank; this essentially means to check for a trace inclusion w.r.t. messages observable on the interfaces, as the protocol state machine has no post conditions.

In order to study, e.g., such a refinement between a protocol state machine and its implementation by state machines, in the following we develop institutions for state machines including a notion of product.

3 Institutions for Simple UML State Machines

We now detail a possible formalisation of a simplified version of UML state machines as institutions. In particular, we omit hierarchical states. Our construction is generic in w.r.t. an institution of *guards*. Then, we give an institutions for the *actions* of a state machine. These fix the conditions which can be used in guards of transitions, the actions for the effects of transitions, and also the messages that can be sent from a state machine. The source of this information typically is a class or a component diagram: The conditions and actions involve the properties available in the classes or components, the messages are derived from the available signals and operations. The sentences of the action institution form a simple dynamic logic (inspired by OCL) which can express that if a guard holds as pre-condition, when executing an action, a certain set of messages is sent out, and another guard holds as post-condition. We then build a family of institutions for *state machines* over the institutions for guards and actions. A state machine adds the events and states that are used. The events comprise the signals and operations that can be accepted by the machine; some of these will, in general, coincide with the messages from the environment. Additionally, the machine may react to completion events, i.e., internal events that are generated when a state of the machine has been entered and which trigger those transitions that do not show an explicit event as their trigger in the diagrammatic representation (we use the states as the names of these events). The initial state as well as the transitions of the machine are represented as sentences in the institution.[1] In a next step, we combine the family of state machine institutions parameterised over actions into a single institution.

3.1 Institution of Guards

We assume that there is an institution of guards. Typically, guards are formulas in some language like OCL. More formally, an *institution of guards* is an institution where signatures are sets, and signature morphisms are functions. We will call the elements of these sets *variables*, but one can think of attributes or properties. Models of a signature V are valuations $\omega : V \to \mathrm{Val}$ into a fixed set of values Val[2]. Model reduct is just composition, that is, given a signature morphism $\upsilon : V \to V'$ and a model $\omega' : V' \to \mathrm{Val}$, its υ-reduct is $\omega' \circ \upsilon$. The nature of sentences $G(V)$ and their translation $G(\upsilon) : G(V) \to G(V')$ is left unspecified, as well as the satisfaction relation — we only require the satisfaction condition, which amounts to

$$\omega' \models G(\upsilon)(g) \quad \text{iff} \quad \omega' \circ \upsilon \models g\,.$$

Example 4. Consider the UML component ATM. A guard signature for ATM would contain the variable trialsNum, leading to sentences such as true, trialsNum < n, and trialsNum == n for $n \in \mathbb{N}$.

[1] For simplicity, final states are left implicit here. For hierarchical states, they need to be made explicit.

[2] In UML, variables and values would be typed, and variable valuations have to respect the typing. For simplicity, we disregard this here. Moreover, UML queries, could be covered by valuations assigning values in some function space. However, a more elaborate institution would be preferable for OCL.

3.2 Institution of Actions

An object of the category of action *signatures* $\mathrm{Sig}^{\mathsf{Act}}$ is a triple of sets

$$H = (A_H, M_H, V_H)$$

of actions, messages and variables; and a morphism $H \rightarrow H'$ of $\mathrm{Sig}^{\mathsf{Act}}$ is a triple of functions $\eta : (\eta_A : A_H \rightarrow A_{H'}, \eta_M : M_H \rightarrow M_{H'}, \eta_V : V_H \rightarrow V_{H'})$. The class of action *structures* $\mathrm{Mod}^{\mathsf{Act}}(H)$ for an action signature H consists of transition relations

$$\Omega \subseteq |\Omega| \times (A_H \times \wp(M_H)) \times |\Omega|\ ,$$

where $|\Omega| = (V_H \rightarrow \mathrm{Val})$ represents the possible configurations of data states, and

$$(\omega, a, \overline{m}, \omega') \in \Omega \quad \text{(also written } \omega \xrightarrow[\Omega]{a,\overline{m}} \omega'\text{)}$$

expresses that action a leads from state $\omega \in (V_H \rightarrow \mathrm{Val})$ to state $\omega' \in (V_H \rightarrow \mathrm{Val})$ producing the set of messages $\overline{m} \subseteq M_H$.

The *reduct* $\Omega'|\eta$ of an H'-action structure Ω' along the morphism $\eta : H \rightarrow H'$ is given by all transitions

$$\omega_1|\eta_V \xrightarrow[\Omega'|\eta]{a,\eta_M^{-1}(\overline{m})} \omega_2|\eta_V \quad \text{for which} \quad \omega_1 \xrightarrow[\Omega']{\eta_A(a),\overline{m}} \omega_2\ .$$

An action a is called *deterministic* if $\omega_1 \xrightarrow[\Omega]{a,\overline{m}} \omega_2$ and $\omega_1 \xrightarrow[\Omega]{a,\overline{m}'} \omega_2'$ imply $\overline{m} = \overline{m}'$ and $\omega_2 = \omega_2'$. An action relation Ω is called deterministic if all its actions are deterministic, that is, it is a partial function of type $|\Omega| \times A_H \rightharpoonup \wp(M_H) \times |\Omega|$.

Note that reducts can introduce non-determinism. Given an action signature $(A, M, \{x, y\})$ suppose that a deterministic action a leads to a change of state expressed by the assignment $x := x + y$. Now take the reduct to the signature $(A, M, \{x\})$, i.e., the variable y has been removed. Then a performs a non-deterministic assignment $x := x + y$ where the value for y is non-deterministically guessed.

The set of action *sentences* $\mathit{Sen}^{\mathsf{Act}}(H)$ for an action signature H comprises the expressions

$$g_{\mathrm{pre}} \rightarrow [a]\overline{m} \rhd g_{\mathrm{post}}$$

with $g_{\mathrm{pre}}, g_{\mathrm{post}} \in G(V_H)$ guard sentences over V_H, $a \in A_H$, and $\overline{m} \subseteq M_H$, intuitively meaning (like an OCL constraint) that if the pre-condition g_{pre} currently holds, then, after executing a, the messages $\overline{m}$ are produced and the post-condition g_{post} holds. The *translation* $\eta(g_{\mathrm{pre}} \rightarrow [a]\overline{m} \rhd g_{\mathrm{post}})$ of a sentence $g_{\mathrm{pre}} \rightarrow [a]\overline{m} \rhd g_{\mathrm{post}}$ along the signature morphism $\eta : H \rightarrow H'$ is given by $G(\eta_V)(g_{\mathrm{pre}}) \rightarrow [\eta_A(a)]\eta_M(\overline{m}) \rhd G(\eta_V)(g_{\mathrm{post}})$. Finally, the satisfaction relation $\Omega \models_H^{\mathsf{Act}} g_{\mathrm{pre}} \rightarrow [a]\overline{m} \rhd g_{\mathrm{post}}$ holds if, and only if, for all $\omega \in (V_H \rightarrow \mathrm{Val})$, if $\omega \models g_{\mathrm{pre}}$ and $\omega \xrightarrow[\Omega]{a,\overline{m}'} \omega'$, then $\omega' \models g_{\mathrm{post}}$ and $\overline{m} \subseteq \overline{m}'$. Then the *satisfaction condition* follows.

Example 5. Consider the UML component ATM with its properties cardId, pin, and trialsNum, its ports userCom and bankCom, and its outgoing operations ejectCard() and keepCard() to userCom, and verify() and markInvalid() to bankCom. An action signature for ATM is derived by forming actions and messages over this information, such that it will contain the actions userCom.ejectCard(); trialsNum = 0 and trialsNum++, as well as the messages userCom.ejectCard() and bankCom.markInvalid(cardId). Action sentences over such an action signature could be

$$\mathsf{true} \to [\mathsf{userCom.ejectCard();\ trialsNum = 0}]\{\mathsf{userCom.ejectCard()}\} \rhd \mathsf{trialsNum == 0}$$
$$\mathsf{trialsNum ==}\ n \to [\mathsf{trialsNum{+}{+}}]\emptyset \rhd \mathsf{trialsNum ==}\ n\mathsf{+1}\ .$$

3.3 Behavioural State Machine Institution

The institution of state machines is now built over the action institution. Let H be an action signature and Ω an action structure over H. An object of the category of state machine *signatures* $\mathrm{Sig}^{\mathsf{SM}(H,\,\Omega)}$ over H and Ω is given by a triple

$$\Sigma = (E_\Sigma, F_\Sigma, S_\Sigma)$$

of (external) events E_Σ, completion events F_Σ, and states S_Σ with $E_\Sigma \cap F_\Sigma = \emptyset$ and $E_\Sigma \cap S_\Sigma = \emptyset$; and a morphism $\sigma : \Sigma \to \Sigma'$ of $\mathrm{Sig}^{\mathsf{SM}(H,\,\Omega)}$ is a triple of injective functions $\sigma = (\sigma_E : E_\Sigma \to E_{\Sigma'}, \sigma_F : F_\Sigma \to F_{\Sigma'}, \sigma_S : S_\Sigma \to S_{\Sigma'})$, such that $E_\Sigma \cap M_H = E_{\Sigma'} \cap M_H$ (preservation of internal messages). The class of state machine *structures* $\mathrm{Mod}^{\mathsf{SM}(H,\,\Omega)}(\Sigma)$ for a state machine signature $\Sigma = (E_\Sigma, F_\Sigma, S_\Sigma)$ over H and Ω consists of the pairs

$$\Theta = (I_\Theta, \Delta_\Theta)$$

where $I_\Theta \in \wp(V_H \to \mathrm{Val}) \times S_\Sigma$ represents the initial configurations, fixing the initial control state; and $\Delta_\Theta \subseteq C_\Sigma \times \wp(M_H) \times C_\Sigma$ with $C_\Sigma = (V_H \to \mathrm{Val}) \times \wp(E_\Sigma \cup F_\Sigma) \times S_\Sigma$ represents a transition relation from a configuration, consisting of an action state, an event pool, and a control state, to a configuration, emitting a set of messages. The event pool may contain both types of events from the signature: external events from signals and operations, and completion events (which are typically represented by states).

Example 6. Consider the state machine of Fig. 2(e) defining the behaviour of ATM. It works over the action signature sketched in the previous example, and its signature is $(E_{\mathsf{ATM}}, F_{\mathsf{ATM}}, S_{\mathsf{ATM}})$ with

$$E_{\mathsf{ATM}} = \{\mathsf{card, PIN, reenterPIN, verified}\}\ ,$$
$$F_{\mathsf{ATM}} = \{\mathsf{PINEntered, Verified}\}\ ,$$
$$S_{\mathsf{ATM}} = \{\mathsf{Idle, CardEntered, PINEntered, Verifying, Verified}\}\ .$$

In particular, the completion events consist of those states from which a completion transition originates.

The *reduct* $\Theta'|\sigma$ of a state machine structure Θ' along the morphism $\sigma : \Sigma \to \Sigma'$ is given by the structure

$$\begin{aligned}
&(\{(\omega, s) \mid (\omega, \sigma_S(s)) \in I'\}, \Delta) \quad \text{with} \\
&\Delta = \{(\omega_1, \sigma_P^{-1}(\overline{p_1}), s_1) \xrightarrow{\overline{m}} (\omega_2, \sigma_P^{-1}(\overline{p_2}), s_2) \mid \\
&\qquad\qquad (\omega_1, \overline{p_1}, \sigma_S(s_1)) \xrightarrow[\Delta_{\Theta'}]{\overline{m}} (\omega_2, \overline{p_2}, \sigma_S(s_2))\}\,,
\end{aligned}$$

where $\sigma_P(p) = \sigma_E(p)$ if $p \in E_\Sigma$ and $\sigma_P(p) = \sigma_F(p)$ if $p \in F_\Sigma$. Here, σ_P^{-1} deletes those events from the event pool that are not present in the pre-image.

The set of state machine *sentences* $Sen^{\mathsf{SM}(H,\,\Omega)}(\Sigma)$ for a state machine signature Σ over H and Ω consists of the pairs

$$\varphi = (s_0 \in S_\Sigma, T \subseteq S_\Sigma \times (E_\Sigma \cup F_\Sigma) \times (G(V_H) \times A_H \times \wp(F_\Sigma)) \times S_\Sigma)$$

where s_0 means an initial state and the transition set T represents the transitions from a state s with a triggering event p (either a declared event or a completion event), a guard g, an action a, and a set of completion events $\overline{f}$ to another state s'. We also write $s \xrightarrow[T]{p[g]/a,\overline{f}} s'$ for such a transition. The translation $\sigma(s_0, T)$ of a sentence (s_0, T) along the signature morphism $\sigma : \Sigma \to \Sigma'$ is given by $(\sigma_S(s_0), \{\sigma_S(s_1) \xrightarrow{\sigma_P(p)[g]/a,\wp\sigma_F(\overline{f})} \sigma_S(s_2) \mid s_1 \xrightarrow[T]{p[g]/a,\overline{f}} s_2\})$. Finally, the *satisfaction relation* $\Theta \models_{\Sigma}^{\mathsf{SM}(H,\,\Omega)} (s_0, T)$ holds if, and only if $\pi_2(I_\Theta) = s_0$ and Δ_Θ is the least transition relation satisfying[3]

$$\begin{aligned}
&(\omega, p :: \overline{p}, s) \xrightarrow[\Delta_\Theta]{\overline{m} \setminus E_\Sigma} (\omega', \overline{p} \lhd ((\overline{m} \cap E_\Sigma) \cup \overline{f}), s') \quad \text{if} \\
&\qquad \exists s \xrightarrow[T]{p[g]/a,\overline{f}} s' \,.\, \omega \models g \wedge \omega \xrightarrow[\Omega]{a,\overline{m}} \omega' \\
&(\omega, p :: \overline{p}, s) \xrightarrow[\Delta_\Theta]{\emptyset} (\omega, \overline{p}, s) \quad \text{if} \\
&\qquad \forall s \xrightarrow[T]{p'[g]/a,\overline{f}} s' \,.\, p \neq p' \vee \omega \not\models g
\end{aligned}$$

where $p :: \overline{p}$ expresses that some element p from the pool $\overline{p}$ is extracted, and $\overline{p} \lhd \overline{p}'$ adds the events in $\overline{p}'$ to the pool $\overline{p}$ with respect to some extraction and selection schemes (where completion events are prioritised). The messages on a transition in the structure Θ are only those that are not accepted by the machine itself, i.e., not in E_Σ. The accepted events in E_Σ as well as the completion events are added to the event pool of the target configuration. When no transition is triggered by the current event, the event is discarded (this will happen, in particular, to all superfluously generated completion events). Checking the satisfaction condition

$$\Theta'|\sigma \models_{\Sigma}^{\mathsf{SM}(H,\,\Omega)} (s_0, T) \;\Leftrightarrow\; \Theta \models_{\Sigma'}^{\mathsf{SM}(H,\,\Omega)} \sigma(s_0, T)$$

for a state machine signature morphism $\sigma : \Sigma \to \Sigma'$ is straightforward.

[3] Usually, the two cases do not overlap, so the two cases are complete characterisations (iff).

Example 7. Continuing the previous example for the state machine of Fig. 2(e) defining the behaviour of ATM, this state machine can be represented as the following sentence over this signature:

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

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.

3.4 Protocol State Machine Institution

Protocol state machines differ from behavioural state machines by not mandating a specific behaviour but just monitoring behaviour: They do not show guards and effects, but a pre- and a post-condition for the trigger of a transition. Moreover, protocol state machines do not just discard an event that currently does not fire a transition; it is an error when such an event occurs.

For adapting the state machine institution to protocol state machines we thus change the *sentences* to

$$\varphi = (s_0, e \in S_\Sigma, T \subseteq S_\Sigma \times (G(V_H) \times E_\Sigma \times G(V_H) \times \wp(M_H) \times \wp(F_\Sigma)) \times S_\Sigma)$$

where s_0 is the start state and e a dedicated error state, the two occurrences of $G(V_H)$ represent the pre- and the post-conditions, and $\wp(M_H)$ represents the messages that have to be sent out in executing the triggering event (protocol state machines typically do not show completion events). The *satisfaction relation* now requires that when an event e is chosen from the event pool the pre-condition of some transition holds in the source configuration, its post-condition holds in the target configuration, and that all messages have been sent out. Instead of the second clause of Δ_Θ, discarding an event, the error state is targeted when no transition is enabled.

3.5 Flat State Machine Institution

Given an institution of guards, we now flatten the institutions $\mathsf{SM}(H, \Omega)$ for each action signature H and each action structure Ω over H into a single institution SM.[4] The signatures $\langle H, \Sigma\rangle$ consist of an action signature H and a state machine signature Σ, similarly for signature morphisms as well as for structures $\langle \Omega, \Theta\rangle$. As $\langle H, \Sigma\rangle$-sentences we now have both dynamic logic formulas (over H), as well as control transition relations

[4] This is an instance of a general construction, namely the Grothendieck institution [19].

(over H and Σ). Also satisfaction is inherited. Only the definition of reducts is new, because they need to reduce state machine structures along more complex signature morphisms: $\langle \Omega', \Theta' \rangle|(\eta, \sigma) = \langle \Omega'|\eta, \Theta'|\sigma|\eta \rangle$ where $\Theta''|\eta = (I_{\Theta''}, \{c_1'', \eta_M^{-1}(\overline{m}''), c_2'') \mid (c_1'', \overline{m}'', c_2'') \in \Delta_{\Theta''}\})$.

4 Determinism, Interleaving, and Refinement

4.1 Deterministic State Machines

The transition and action relations are not required to be functions. Thus a transition may have multiple choices for the same configuration of states, variables and events. But when moving towards the implementation, deterministic behaviour is desirable.

i-a) A transition set T is called *syntactically deterministic* if it is a partial function of type $S_\Sigma \times (E_\Sigma \cup F_\Sigma) \rightharpoonup G_H \times A_H \times \wp(F_\Sigma) \times S_\Sigma$.

i-b) A transition set T is called *semantically deterministic* if for any two distinct transitions $s \xrightarrow[T]{p[g_1]/a_1,\overline{f}_1} s_1$ and $s \xrightarrow[T]{p[g_2]/a_2,\overline{f}_2} s_2$ sharing the same pre-state s and trigger event p, their guards must be disjoint, that is, there is no $\omega : V_H \to \mathrm{Val}$ satisfying both g_1 and g_2.

ii) A transition relation Δ_Θ is called *deterministic* if and only if it is a partial function of type $C_\Sigma \rightharpoonup \wp(M_H) \times C_\Sigma$.[5]

iii) A state machine structure (Ω, Θ) of SM is called *deterministic* if and only if the corresponding action relation and transition relation are deterministic.

The transition relation Δ_Θ is defined by Ω and T. So it is justified to expect some inheritance of determinism between those.

Theorem 1. *If T is syntactically or semantically deterministic and Ω is deterministic, then Δ_Θ is also deterministic.*

Proof. Consider a configuration $(\omega, p :: \overline{p}, s)$. If there is any transition, then the new state s' and executed action a are determined by $T(s, p) = (g, a, \overline{f}, s')$ (if defined) in the syntactic case. The sent message $\overline{m}$ and the new configuration of the variables ω' result from $\Omega(\omega, a) = (\overline{m}, \omega')$. In the semantic case, at most one guard can be enabled, hence at most one transition in T can fire. □

4.2 Interleaving Product of State Machines

Inside the flat state machine institution SM we can consider the composition of state machines over different action signatures. The composition captures the interplay between different state machines and their communication. The different action signatures represent the local views of the state machines. This composition interleaves the behaviours

[5] Note that this function is total if Δ_Θ satisfies some sentence. This originates from the discarding of events that can not be processed in the current state and configuration, which is again a transition.

of the UML state machines. The communication is performed by the exchange of messages which are turned into events, not by synchronisation over shared events.

Given two state machine signatures $\langle H_1, \Sigma_1 \rangle$ and $\langle H_2, \Sigma_2 \rangle$ of SM with $E_{\Sigma_1} \cap E_{\Sigma_2} = \emptyset$ and $S_{\Sigma_1} \cap S_{\Sigma_2} = \emptyset$, we combine these into a single signature $\langle \hat{H}, \hat{\Sigma} \rangle$ of SM by taking the component-wise union for the guard, actions, messages, and variables, the union of events and states for the events, and the product of the state sets for the states. Now consider two state machine structures (Ω_1, Θ_1) over $\langle H_1, \Sigma_1 \rangle$ and (Ω_2, Θ_2) over $\langle H_2, \Sigma_2 \rangle$, respectively. Their *interleaving product* is given by

$$\langle \Omega_1, \Theta_1 \rangle \parallel \langle \Omega_2, \Theta_2 \rangle = (\Omega_1 \parallel \Omega_2, \Theta_1 \parallel \Theta_2) \qquad \text{where}$$

- $\Omega_1 \parallel \Omega_2$ is given by $\omega \xrightarrow[\Omega_1 \parallel \Omega_2]{a, \overline{m}} \omega'$ if for some $i \in \{1, 2\}$: $a \in A_{H_i}$ and $\omega|V_{H_i} \xrightarrow[\Omega_i]{a, \overline{m}} \omega'|V_{H_i}$ and for $i \neq j \in \{1, 2\} : \omega|(V_{H_j} \setminus V_{H_i}) = \omega'|(V_{H_j} \setminus V_{H_i})$
- $\Theta_1 \parallel \Theta_2 = (I_{\Theta_1} \parallel I_{\Theta_2}, \Delta_{\Theta_1} \parallel \Delta_{\Theta_2})$ with, letting $I_{\Theta_i} = (\Gamma_i, s_i)$,

$$I_{\Theta_1} \parallel I_{\Theta_2} = (\{\omega : V_{H_1} \cup V_{H_2} \to \mathrm{Val} \mid \forall j \in \{1, 2\} \,.\, \omega|V_{H_j} \in \Gamma_j\}, (s_1, s_2)) \quad \text{and}$$

$$(\omega, p :: (\overline{p}_1 \cup \overline{p}_2), (s_1, s_2)) \xrightarrow[\Delta_{\Theta_1} \parallel \Delta_{\Theta_2}]{\overline{m} \setminus E_{\hat{\Sigma}}} (\omega', (\overline{p}_1 \cup \overline{p}_2) \lhd ((\overline{p}' \cup \overline{m}) \cap E_{\hat{\Sigma}}), (s'_1, s'_2))$$

$$\text{iff } \exists i \in \{1, 2\} \,.\, (\omega|V_{H_i}, p :: \overline{p}_i, s_i) \xrightarrow[\Delta_{\Theta_i}]{\overline{m}} (\omega'|V_{H_i}, \overline{p}_i \lhd \overline{p}', s'_i) \land$$
$$\forall j \in \{1, 2\} \setminus \{i\} \,.\, (\omega|V_{H_j}, \overline{p}_j, s_j) = (\omega'|V_{H_j}, \overline{p}_j, s'_j) \,.$$

There is also a syntactic version of the interleaving product: given sentences (s_0^1, T_1) and (s_0^2, T_2), their interleaving $(s_0^1, T_1) \parallel (s_0^2, T_2)$ is given by $((s_0^1, s_0^2), T)$ with

$$(s_1, s_2) \xrightarrow[T]{p[g]/a, \overline{f}} (s'_1, s'_2) \text{ iff } \exists i \in \{1, 2\} \,.\, s_i \xrightarrow[T_i]{p[g]/a, \overline{f}} s'_i \land \forall j \in \{1, 2\} \setminus \{i\} \,.\, s_j = s'_j \,.$$

The syntactic version is compatible with the semantic interleaving product:

Theorem 2. *If* $\langle \Omega_i, \Theta_i \rangle \models (s_0^i, T_i)$ *for* $i \in \{1, 2\}$, *then* $\langle \Omega_1, \Theta_1 \rangle \parallel \langle \Omega_2, \Theta_2 \rangle \models (s_0^1, T_1) \parallel (s_0^2, T_2)$. □

Example 8. Consider the composite structure diagram in Fig. 2(b), 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 which internalises those events that are part of the common signature. On the other hand, messages to the outside, i.e., through the userCom port are still visible.

A system resulting from an interleaving product $\langle \Omega_1, \Theta_1 \rangle \parallel \langle \Omega_2, \Theta_2 \rangle$ represents a state machine in our notation. Thus it can be again part of an interleaving product

$$\langle (\langle \Omega_1, \Theta_1 \rangle \parallel \langle \Omega_2, \Theta_2 \rangle) \parallel \langle \Omega_3, \Theta_3 \rangle \rangle \,.$$

The interleaving product meets the intuitive algebraic properties. Due to the disjoint event sets each event can only trigger at most one machine. Messages are stripped off

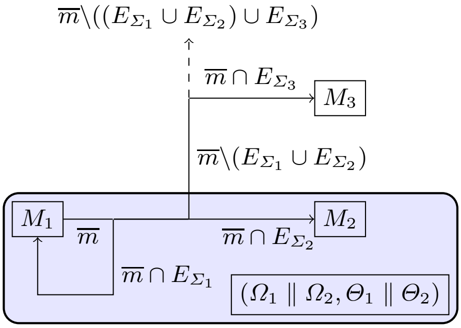

$\overline{m}\backslash((E_{\Sigma_1} \cup E_{\Sigma_2}) \cup E_{\Sigma_3})$
$\overline{m} \cap E_{\Sigma_3}$
M_3
$\overline{m}\backslash(E_{\Sigma_1} \cup E_{\Sigma_2})$
M_1
M_2
$\overline{m}$
$\overline{m} \cap E_{\Sigma_2}$
$\overline{m} \cap E_{\Sigma_1}$
$(\Omega_1 \parallel \Omega_2, \Theta_1 \parallel \Theta_2)$

Fig. 3. Messages sent between three machines on transition in machine M_1

the events which can be processed by either of the inner machines, and remaining messages are sent to the third machine, which also extracts its corresponding events as illustrated in Fig. 3. Hence it is impossible that the inner machines consume an event that can also be processed by the third machine. The same behaviour occurs, if the first machine sends a message to a system of the two remaining machines. Thus the distinction in “inner” and “outer” machines becomes obsolete and the interleaving product is associative. Since each machine extracts the events present in its event set, it is not required to consider the order of the machines, and hence the interleaving product is commutative. Finally, we can regard a state machine with no events, no messages, and only one state. In an interleaving product this nearly empty machine would have no effect on the behaviour of the other machine, and thus behaves as a neutral element.

Theorem 3. *The set of state machine structures (over all signatures) with interleaving product $\parallel$ forms a discrete symmetric monoidal category, which is a “commutative monoid up to isomorphism”.* □

It is desirable that the interleaving product of two deterministic machines preserves this determinism. The new action function is determined by the two old ones in such a way, that the new configuration is taken from the configuration of the triggered submachine, which is deterministic, and the missing variable configuration remains untouched. Thus the new action relation is deterministic. The same goes for the transition relation. The sent messages and configuration are determined by the (as argued above) deterministic action relation and the new state and events result from the triggered submachine. However, we need the following prerequisite: Two action relations Ω_1, Ω_2 are called *compatible* if $\Omega_1|(H_1 \cap H_2) = \Omega_2|(H_1 \cap H_2)$.

Theorem 4. *Let (Ω_1, Θ_1) and (Ω_2, Θ_2) be deterministic state machines with both action relations compatible. Then $(\Omega_1 \parallel \Omega_2, \Theta_1 \parallel \Theta_2)$ is also deterministic.* □

Using a slightly modified version of the interleaving product construction where messages of shared actions leading to compatible states are united, instead of generating two separate transitions, we can prove:

Theorem 5. *The action institution admits weak amalgamation for pushout squares with injective message mappings.* □

4.3 Institutional Refinement of State Machines

We have defined an institution capturing both behavioural and protocol state machines via different sentences. With the machinery developed so far, we can now apply the institution independent notion of refinement to our institution of state machines. The simplest such notion is just model class inclusion, that is, a theory T_1 refines to T_2, written $T_1 \rightsquigarrow T_2$, if $\mathrm{Mod}^{\mathsf{SM}}(T_2) \subseteq \mathrm{Mod}^{\mathsf{SM}}(T_1)$. (Note that state machines are theories consisting typically of one sentence only.)

However, this is too simple to cover the phenomenon of *state abstraction*, where several states (like Idle, CardEntered, PinEntered and Verified in Fig. 2(e)) in a more concrete state machine can be abstracted to one state (like Idle in Fig. 2(d)) in a more abstract state machine. This situation can be modelled using the institution independent notion of *translation* of a theory T along a signature morphism $\sigma : \mathit{sig}(T) \to \Sigma$, resulting in a structured theory $\sigma(T)$ which has signature Σ, while the model class is $\{M \in \mathrm{Mod}^{\mathsf{SM}}(\Sigma) \mid M|\sigma \in \mathrm{Mod}^{\mathsf{SM}}(T)\}$, i.e., models are those Σ-models that reduce (via σ) to a T-model. Moreover, sometimes we want to *drop events* (like card in Fig. 2(e)) when moving to a more abstract state machine. This can be modelled by a notion dual to translation, namely *hiding*. Given a theory T and a signature morphism $\theta : \Sigma \to \mathit{sig}(T)$, the structured theory $\theta^{-1}(T)$ has signature Σ, while the model class is $\{M|\theta \in \mathrm{Mod}^{\mathsf{SM}}(\Sigma) \mid M \in \mathrm{Mod}^{\mathsf{SM}}(T)\}$, i.e., models are all θ-reducts of T-models. Altogether, we arrive at

Definition 1. *An "abstract" (behavioural or protocol) state machine* T_1 refines into *a "concrete" state machine* T_2 *via signature morphisms* $\theta : \mathit{sig}(T_1) \to \Sigma$ *and* $\sigma : \mathit{sig}(T_2) \to \Sigma$ *into some "mediating signature"* Σ*, if*

$$T_1 \rightsquigarrow \theta^{-1}(\sigma(T_2))$$

in other words, for all Σ*-models* M

$$M|\sigma \in \mathrm{Mod}^{\mathsf{SM}}(T_2) \Rightarrow M|\theta \in \mathrm{Mod}^{\mathsf{SM}}(T_1)\,.$$

Concerning our original refinement question stated in Ex. 3, we now can argue: As the state machine of the atm, shown in Fig. 2(e) is a refinement of the protocol state machine in Fig. 2(d), using a suitable signature morphism, the interleaving product of the atm and bank state machine, in the syntactic version, will be so as well. As furthermore the protocol state machine has no post conditions, we have established a refinement.

5 Conclusions

We have presented institutions for behavioural and protocol UML state machines and have studied an interleaving product and a notion of determinism. We furthermore presented first steps of how to study refinement in such a context. Our institutions provide the necessary prerequisites for including UML state machines into a heterogeneous institution-based UML semantics and to develop their relationship to other UML sub-languages and diagram types.

An important future extension for the state machine institutions is to add hierarchical states, and to consider refinements from hierarchical to flat state machines. For an integration into the software development process, the study of correct code generation is indispensable. The Heterogeneous Tool Set (Hets [18,20]) provides analysis and proof support for multi-logic specifications, based on a strong semantic (institution-based) backbone. Implementation of proof support for UML state machines (and other kinds of UML diagrams) is under way.

References

1. Object Management Group: Unified Modeling Language. Standard formal/2011-08-06, OMG (2011)
2. Knapp, A., Mossakowski, T., Roggenbach, M.: Towards an Institutional Framework for Heterogeneous Formal Development in UML - A Position Paper. In: De Nicola, R., Hennicker, R. (eds.) Wirsing Festschrift. LNCS, vol. 8950, pp. 215–230. Springer, Heidelberg (2015)
3. Goguen, J.A., Burstall, R.M.: Institutions: Abstract model theory for specification and programming. J. ACM 39, 95–146 (1992)
4. Große-Rhode, M.: Semantic Integration of Heterogeneous Software Specifications. Monographs in Theoretical Computer Science. Springer (2004)
5. Roggenbach, M.: CSP-CASL: A New Integration of Process Algebra and Algebraic Specification. Theo. Comp. Sci. 354, 42–71 (2006)
6. Mossakowski, T., Roggenbach, M.: Structured CSP – A Process Algebra as an Institution. In: Fiadeiro, J.L., Schobbens, P.-Y. (eds.) WADT 2006. LNCS, vol. 4409, pp. 92–110. Springer, Heidelberg (2007)
7. O'Reilly, L., Mossakowski, T., Roggenbach, M.: Compositional Modelling and Reasoning in an Institution for Processes and Data. In: Mossakowski, T., Kreowski, H.-J. (eds.) WADT 2010. LNCS, vol. 7137, pp. 251–269. Springer, Heidelberg (2012)
8. Schattkowsky, T., Müller, W.: Transformation of UML State Machines for Direct Execution. In: VL/HCC 2005, pp. 117–124. IEEE (2005)
9. Fecher, H., Schönborn, J.: UML 2.0 State Machines: Complete Formal Semantics Via core state machine. In: Brim, L., Haverkort, B.R., Leucker, M., van de Pol, J. (eds.) FMICS and PDMC 2006. LNCS, vol. 4346, pp. 244–260. Springer, Heidelberg (2007)
10. Calegari, D., Szasz, N.: Institutionalising UML 2.0 State Machines. Innov. Syst. Softw. Eng. 7, 315–323 (2011)
11. Object Management Group: Precise Semantics of UML Composite Structures. Beta Specification ptc/14-06-15, OMG (2014)
12. Dereziska, A., Szczykulski, M.: Interpretation Problems in Code Generation from UML State Machines — A Comparative Study. In: Kwater, T. (ed.) Computing in Science and Technology 2011: Monographs in Applied Informatics, pp. 36–50. Warsaw University (2012)
13. Lano, K. (ed.): UML 2 — Semantics and Applications. Wiley (2009)
14. Engels, G., Heckel, R., Küster, J.M.: The Consistency Workbench: A Tool for Consistency Management in UML-Based Development. In: Stevens, P., Whittle, J., Booch, G. (eds.) UML 2003. LNCS, vol. 2863, pp. 356–359. Springer, Heidelberg (2003)
15. Mossakowski, T., Sannella, D., Tarlecki, A.: A Simple Refinement Language for CASL. In: Fiadeiro, J.L., Mosses, P.D., Orejas, F. (eds.) WADT 2004. LNCS, vol. 3423, pp. 162–185. Springer, Heidelberg (2005)
16. Codescu, M., Mossakowski, T., Sannella, D., Tarlecki, A.: Specification Refinements: Calculi, Tools, and Applications (2014) (submitted)

17. Borzyszkowski, T.: Logical Systems for Structured Specifications. Theor. Comput. Sci. 286, 197–245 (2002)
18. Mossakowski, T., Autexier, S., Hutter, D.: Development Graphs — Proof Management for Structured Specifications. J. Log. Alg. Program. 67, 114–145 (2006)
19. Diaconescu, R.: Grothendieck Institutions. Applied Cat. Struct. 10, 383–402 (2002)
20. Mossakowski, T., Maeder, C., Lüttich, K.: The Heterogeneous Tool Set, HETS. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 519–522. Springer, Heidelberg (2007)