On Semantics and Refinement of UML Statecharts: A Coalgebraic View *

Sun Meng, Zhang Naixiao
LMAM, School of Mathematical Science
Peking University
{sunmeng,znx}@math.pku.edu.cn

Luís S. Barbosa
Department of Informatics
Minho University
lsb@di.uminho.pt

Abstract

Statecharts was conceived as a visual formalism for the design of reactive systems. UML statecharts is an object-based variant of classical statecharts, incorporating several concepts different from the classical statecharts. This paper discusses a coalgebraic description of UML statecharts, directly derived from its operational semantics. In particular, such an approach induces suitable notions of equivalence and (behavioral) refinement for statecharts. Finally, a few refinement laws are investigated to support verifiable step-wise system development with statecharts.

Keywords:
Statechart, Semantics, Coalgebra, Refinement

1. Introduction

Statecharts, a visual language for specifying the behavior of *reactive* systems introduced originally by D. Harel in [3], has been found versatile enough to be used in the context of OO system development. The formalism is similar to *finite-state machines*, but extends them by three main ingredients: hierarchy, concurrency, and communication. Such concepts, while intuitive on their own, interact in intricate ways. As a result, a number of formal semantics proposals (and corresponding tools) has flourished, interpreting their interaction in different ways or imposing different constraints. By far the most significant development in recent years has been the emergence of widely used design notations like STATEMATE [4], ROOM [22], and UML [19], and commercial tools to support them.

The UML statecharts formalism is an object-based variant of Harel statecharts, which provides a view of the dynamic behavior of the system. It incorporates several concepts similar to those defined in the ROOM modelling language and is different from the classical Harel statecharts. The main difference results from the external context of the state machine. UML statecharts mainly specifies behavior of a *type* while the classical statecharts represent the behaviors of *processes*. There are other differences resulting from this rationale and a detailed comparison between them can be found in [19]. While the graphical syntax of the language has been formally specified, precisely defining its formal semantics proved to be extremely challenging, and more difficult than originally expected. Recently, a variety of proposals on semantics of UML statecharts has been offered in the literature [4, 11, 13, 14].

Most of the existing results on statecharts semantics are given by using labelled transition systems (LTS), which can be naturally represented as coalgebras (see e.g. [21]). This paper extends the operational semantics of statecharts given in [13] and proposes a coalgebraic account for statecharts at first hand, agreeing with their operational semantics. The difference between the operational semantics in this paper and the definition in [13] is that the transitions are considered hierarchically here and a corresponding transition rule is added. Based on the coalgebraic semantics, we investigate the equivalence of statecharts and define a notion of refinement for statecharts, which indicates whether the behavior of a system specified by a statechart is simulated by another one, so that we can replace the former by the latter. Note that although the results in this paper are for statecharts, the notion of refinement here is an instantiation of *behavioral refinement* for generic state-based components [16], which is quite independent of this formalism. So it can also be used in other contexts. Finally, a family of refinement laws for statecharts is described on the syntactic level. These laws have a well-defined underlying semantics for ensuring their correctness, which is based on the simulation relationship between statecharts.

The motivation of our work is that software systems (classes, components, etc.) can be naturally modelled as coalgebras (see e.g. [1, 6, 16, 21]), i.e., pairs $(U, \alpha : U \to \mathsf{T}U)$ where the carrier U represents the set of states of the system, and T is a functor which describes the behavior type

* This work is partially supported by the National Natural Science Foundation of China under Grant No. 60273001. The work of Luís S. Barbosa is supported by FCT, under contract POSI/ICHS/44304/2002, in the context of the PURe project.

IEEE COMPUTER SOCIETY

of the system, given as a signature of transitions and observations. One obvious advantage of the coalgebraic approach for statecharts semantics is that it induces a simple notion of behavior equivalence on statecharts, which can be characterized as a coalgebraic bisimilarity. Moreover, it provides a *uniform* treatment of statecharts and other modelling languages, e.g., class diagrams and use cases [15]. Placing data and behavior at a similar level conveys the idea that statechart models can be chosen and specified according to a given application area, in the same way that a suitable data structure is defined to meet a particular requirement, and can be used as a unifying framework to represent different modelling formalisms. Finally, UML statecharts always specify behavior of types, where the states can be represented as algebras of some signature. By commuting the underlying environment category from $\mathbf{Set}$ to a set based category enriched with some algebraic structure, like in [2], we can model the UML statecharts instead of classical statecharts naturally.

It is the aim of this paper to present a step towards covering the gap between coalgebra theory and practice in software development. We believe that both theory and practice can benefit from this approach.

This paper is organized as follows: In Section 2, the operational semantics of statecharts is given by using extended hierarchical automata, and the coalgebraic semantics is defined. The equivalence of statecharts is presented by exploiting the coalgebraic semantics in Section 3. Section 4 discusses the notion of refinement for statecharts based on the coalgebraic semantics, and gives a family of refinement laws. The paper finishes with a discussion on concluding remarks and future work in Section 5.

2. Semantics for Statecharts

In this section we recall the notion of Extended Hierarchical Automata (EHA) defined in [17] for statecharts semantics. The operational semantics for statecharts is faithfully represented by transitions in the EHA. Then we will briefly present the coalgebraic description of statecharts semantics, where the carrier of a coalgebra represents the configurations of the system being modelled by the statechart, and the transition structure interprets the evolution of the system being triggered by the events and the optional action being performed.

Transitions in a statechart are relationships among states. One transition indicates that an object in the first state (source) will enter the second state (target) and perform specific actions when a specified event occurs provided that certain specified conditions are satisfied. A transition may be labelled by a transition string with the following general format where the options may be omitted.

event-name '(' parameter-list ')' '['
guard-condition ']' '/' action-expression

For the purpose of representing statechart diagrams, we shall use functions $src(t)$, $ev(t)$, $g(t)$, $ac(t)$ and $tgt(t)$ denote, respectively, the source state, trigger event, guard, action and target state of transition t.

2.1. From Statechart to Extended Hierarchical Automata

We first give the definition of an extended hierarchical automata, which is built by the composition of sequential automata. Then we briefly show how to represent statecharts by extended hierarchical automata. A detailed translation from statecharts to extended hierarchical automata can be found in [13].

A *sequential automata* A is a tuple $(\Sigma_A, s_A^0, L_A, \delta_A)$ where Σ_A is the set of *states* of A with $s_A^0 \in \Sigma_A$ the *initial state*, L_A is a finite set of *transition labels* of A and $\delta_A \subseteq \Sigma_A \times L_A \times \Sigma_A$ is the *transition relation*. Sequential automata can be parallely and hierarchically composed to build hierarchical automata [17].

Definition 2.1 *An* extended hierarchical automata H *is given by a triple* (F, E, ξ) *where* $F = \{A_i\}_{i=1,\cdots,n}$ *is a set of sequential automata with mutually distinct state spaces (for any* i, j *in* $\{1, \cdots, n\}$, $i \neq j$, $\Sigma_{A_i} \cap \Sigma_{A_j} = \emptyset$*),* E *is a finite set of events and* ξ *a composition function* $\xi : \bigcup_{A \in F} \Sigma_A \rightarrow \mathcal{P}(F)$ *which satisfies:*

1. $\exists\,!\,A \in F.A \notin \bigcup ran(\xi)$*, denoted by* ξ_{root}*;*
2. $\bigcup ran(\xi) = F \setminus \{\xi_{root}\}$ *and* $\forall A \in F \setminus \{\xi_{root}\}.\exists\,!\,s \in \bigcup_{A' \in F \setminus \{A\}} \Sigma_{A'}\ .\ A \in \xi(s)$*;*
3. $\forall S \subseteq \bigcup_{A \in F} \Sigma_A.((\exists s \in S.S \cap \bigcup_{A \in \xi(s)} \Sigma_A = \emptyset) \vee S = \emptyset)$.

In the sequel for $A \in F$, H will be identified with A if $A = \xi_{root}$. For $A \in F$ an automata in the extended hierarchical automata H, we use $\mathcal{A}A$, $\mathcal{S}A$ and $\mathcal{T}A$ to denote the *automata*, *states* and *transitions* under A respectively. The composition function ξ on $F = \{A_i\}_{i=1,\cdots,n}$ induces a *successor function* $\theta : \bigcup_{A \in F} \Sigma_A \rightarrow \mathcal{P}(\bigcup_{A \in F} \Sigma_A)$, which is defined by

$$\theta(s) = \{s' \mid \exists A \in F.A \in \xi(s) \wedge s' \in \Sigma_A\}$$

We use θ^+ and θ^* to denote the *irreflexive*, resp. *reflexive transitive closure* of θ. Moreover, θ induces an irreflexive partial order on states in $\bigcup_{A \in F} \Sigma_A$:

$$s' \prec s \ \text{ iff } \ s' \in \theta^+(s).$$

We say that s is the *parent* of s' iff $s' \in \theta(s)$ and s is an ancestor of s' iff $s' \in \theta^+(s)$.

IEEE COMPUTER SOCIETY

A statechart can be translated to an extended hierarchical automata $H = (F, E, \xi)$ by defining F, ξ and E respectively. In the following we give a brief description of such a translation. A more detailed translation from statechart to extended hierarchical automata can be found in [13].

- Each automata $A \in F$ is defined as follows:
  1. States of the statechart are uniquely mapped to states of sequential automata.
  2. Initial state of an automata is the state corresponding to the state in the statechart marked by an initial pseudostate.
  3. A transition in a statechart is characterized by its least common ancestor (LCA) state, which is the lowest level non-concurrent state containing all the source and target states. The main source (main target) of a transition is the direct substate of its LCA that contains the sources(targets). Main sources and main targets are always transformed to states of the same automata. And each transition in the statechart is mapped to a unique transition of the extended hierarchical automata.
  4. The label of a transition t is generated by using the source(s) and target(s), while events, guard and action are inherited from t.
- The set of events E is the union of the set of trigger events and the set of events generated by actions in the statechart.
- The composition function ξ is determined by the substate relationships of composite states.

2.2. Operational Semantics

A *configuration* of an extended hierarchical automata denotes a global state of it, composed of local states of component sequential automata, together with the current environment with which the extended hierarchical automata is supposed to interact. The global state describes which states of sequential automata in an extended hierarchical automata are simultaneously active. The environment described a structure (like FIFO queues, bags, or sets, etc.) over the active events. For simplicity, we assume in the sequel such structures be modelled by sets, like in classical statecharts.

Definition 2.2 *A configuration of* $H = (F, E, \xi)$ *is pair* $\langle G, E_A \rangle$ *where*

- $G \subseteq \bigcup_{A \in F} \Sigma_A$ *is a set of states such that*
  1. *Exactly one state of* ξ_{root} *is in the configuration:* $\exists\,!\, s \in \Sigma_{\xi_{root}} \,.\, s \in G$*;*
  2. *Downward closure:* $\forall s, A \,.\, s \in G \;\wedge\; A \in \xi(s) \Rightarrow \exists\,!\, s' \in \Sigma_A \,.\, s' \in G$.
- $E_A \subseteq E$ *denotes the current environment of active events.*

The restriction of a configuration $\langle G, E_A \rangle$ *of* $H = (F, E, \xi)$ *at one of its subautomata* A' *is defined as* $\langle G|_{A'}, E_A \rangle$*, where* $G|_{A'} = G \setminus \{s \mid s' \in G \cap \Sigma_{A'} \;\wedge\; s' \prec s\}$*. The initial configuration of* H *is* $I = \langle G, E \rangle$ *where*

$$\forall s, A \,.\, A \in F \wedge s \in G \cap \Sigma_A \Rightarrow s = s_A^0$$

The operational semantics of an extended hierarchical automata is given by a transition relation $\stackrel{STEP}{-\!\!\!\longrightarrow}$, which is defined via a deduction system as follows.

A system may be either *closed* (not interacting with the environment) or *open* (interacting by receiving events from and supplying generated events to the environment). The corresponding rules are given respectively as follows.

Rule 1 (Closed systems)

$$\frac{e \in E \;\wedge\; E'' = E \setminus \{e\} \;\wedge\; b(G) \qquad (G, \{e\}) \stackrel{e[b]}{-\!\!\!\longrightarrow} (G', E')}{(G, E) \stackrel{STEP}{-\!\!\!\longrightarrow} (G', E'' \cup E')}$$

For open systems, the environment is allowed to add events after each step.

Rule 2 (Open systems)

$$\frac{e \in E \;\wedge\; E'' = E \setminus \{e\} \;\wedge\; b(G) \qquad (G, \{e\}) \stackrel{e[b]}{-\!\!\!\longrightarrow} (G', E') \qquad E' \subseteq E'''}{(G, E) \stackrel{STEP}{-\!\!\!\longrightarrow} (G', E'' \cup E''')}$$

In the above rules an auxiliary relation $\stackrel{e[b]}{-\!\!\!\longrightarrow}$ is used for modelling transitions of the extended hierarchical automata A, where e is the event triggering the corresponding transition and b is the optional guard condition which should be true to fire the transition. Before defining the deduction system for $\stackrel{e[b]}{-\!\!\!\longrightarrow}$ we first give the following auxiliary definition.

Definition 2.3 *For* $A \in F$*, set of states* G *and environment* E*, the set of all the enabled local transitions of* A *in* (G, E)*,* $LEn_A(G, E)$ *is defined as follows:*

$$LEn_A(G, E) = \{t \in \delta_A \mid src(t) \in G \wedge ev(t) \in E \wedge (G, E) \models g(t)\}$$

The set of all enabled transitions of A *in* (G, E) *includes all the* $LEn_{A'}(G, E)$ *where* A' *is a descendent of* A*, which is defined as follows:*

$$En_A(G, E) = \bigcup_{A' \in \mathcal{A}A} LEn_{A'}(G, E)$$

Rule 3 (Progress)

$$\frac{\begin{array}{c} t \in LEn_A(G,E) \\ \nexists\, t' \in En_A(G,E)\ .\ src(t') \prec src(t) \\ G' = G \setminus (\{src(t)\} \cup \{s \mid src(t) \prec s\}) \\ G'' = G' \cup (\{tgt(t)\} \cup \{s \mid tgt(t) \prec s\}) \\ E' = (E \setminus \{ev(t)\}) \cup ac(t) \end{array}}{(G,E) \overset{ev(t)[g(t)]}{\longrightarrow} (G'',E')}$$

In this rule, the event $ev(t)$ being used to trigger the transition t is consumed and will no longer exist. This mechanism looks intuitive and reasonable and can help to prevent incorrect looping. Once a transition t is taken, a new configuration is entered and proper actions are performed. The condition $\nexists\, t' \in En_A(G,E)\ .\ src(t') \prec src(t)$ specifies the priority condition: there is no transition having higher priority than t at the configuration (G,E).

Transitions in statecharts are considered hierarchically in this paper, and this provides an extension to the semantics definition in [13]. If no local transitions of A are enabled, an enabled transition for the active substate of A may be performed instead. This consideration is carried out in the following rule.

Rule 4 (Hierarchy)

$$\frac{\begin{array}{c} LEn_A(G,E) = \emptyset \\ A' \in \mathcal{A}A \\ t \in LEn_{A'}(G,E) \\ (G,E) \overset{ev(t)[g(t)]}{\longrightarrow} (G',E') \\ G'' = G' \cup \{s \mid tgt(t) \prec s\} \end{array}}{(G \cup \{s \mid src(t) \prec s\}, E) \overset{ev(t)[g(t)]}{\longrightarrow} (G'',E')}$$

For a parallel statechart, it is natural and intuitive that several transitions allocated in orthogonal components may be executed simultaneously. That means, they can be performed in a truly concurrent way. This situation is captured by the following rule:

Rule 5 (Composition)

$$\frac{\begin{array}{c} \{s\} = G \cap \Sigma_A \\ \xi(s) = \{A_1, A_2, \cdots, A_n\} \neq \emptyset \\ (G|_{A_i}, E) \overset{ev(t_i)[g(t_i)]}{\longrightarrow} (G_i, E_i)\ for\ 1 \le i \le m \\ En_{A_j}(G|_{A_j}, E) = \emptyset\ for\ m < j \le n \\ G' = G \setminus \bigcup_{1\le i\le m} (\{src(t_i)\} \cup \{s \mid src(t_i) \prec s\}) \\ G'' = G' \cup \bigcup_{1\le i\le m} (\{tgt(t_i)\} \cup \{s \mid tgt(t_i) \prec s\}) \\ E' = (E \setminus (\bigcup_{1\le i\le m} ev(t_i))) \cup \bigcup_{1\le i\le m} ac(t_i) \end{array}}{(G,E) \overset{\bigcup_{1\le i\le m} ev(t_i)[\bigwedge_{1\le i\le m} g(t_i)]}{\longrightarrow} (G'',E')}$$

If no transition in an extended hierarchical automata A is enabled, and no subautomata exist to which the transition may be delegated, then A has to "stutter".

Rule 6 (Stuttering)

$$\frac{En_A(G,E) = \emptyset}{(G,E) \overset{\emptyset}{\longrightarrow} (G,E)}$$

2.3. Semantics of Statecharts: Coalgebraically

A coalgebraic semantics of statecharts may be directly induced by its operational semantics. This leads to a definition of statecharts as coalgebras, whose carriers represent configurations and the dynamics is determined by their operational semantics. The coalgebraic structure captures the evolution of the system modelled by the statechart.

Because of the hierarchical structure of statecharts, we will endow the set of configurations with a coalgebraic structure induced by the operational semantics rather than simply construct the coalgebraic structure over the set of states. Let $CF_A \subseteq \mathcal{P}(\Sigma_A)$ be the set of possible configurations for a given statechart A. Define functor T as follows:

$$\mathsf{T}(X) = \mathsf{B}(X \times \mathcal{P}E)^E$$

where E denotes the set of all events (Here events can carry parameters rather than being primitive signals) and B is a strong monad[1] for specifying the behavior pattern of the statechart. This is a novelty of the approach in the sense that by parameterizing T by a strong monad B we can abstract away from a particular behavioral model (like determinism or non-determinism) underlying the statechart definition. This means that the computation of a statechart transition will not simply produce an action and a continuation configuration, but a B-structure of such pairs.

Several possibilities can be considered, on a pragmatic basis, in the definition of B. The simplest case is, obviously, the *identity* monad, Id. Systems specified by such statecharts would then behave in a totally deterministic way. More interesting possibilities, capturing more complex behavioral features, include:

- *Partiality*, i.e., the possibility of deadlock or failure, captured by the usual maybe monad $\mathsf{B} = \mathsf{Id} + 1$.
- *Nondeterminism*, introduced by the (finite) powerset monad, $\mathsf{B} = \mathcal{P}$. If the target of a transition in a statechart is a composite state, i.e., the transition may have more than one configurations as its target, then the effect of such a transition would be captured by the powerset monad.

1 A *strong monad* is a monad $\langle \mathsf{B}, \eta, \mu \rangle$ where B is a strong functor and both η and μ are strong natural transformations [12]. B being strong means there exist natural transformations $\tau_r{}^{\mathsf{B}} : \mathsf{B} \times - \Longrightarrow \mathsf{B}(\mathsf{Id} \times -)$ and $\tau_l{}^{\mathsf{B}} : - \times \mathsf{B} \Longrightarrow \mathsf{B}(- \times \mathsf{Id})$, called the right and left strength, respectively, subject to certain conditions. Their effect is to *distribute* the free variable values in the context "$-$" along functor B.

IEEE COMPUTER SOCIETY

- *Markov stamping*, with $\mathsf{B} = \mathsf{Id} \times M$, $M = [0,1]$. And it should hold

$$\forall s \in CF \;.\; \sum_{e \in E} \pi_2(\alpha(s,e)) = 1$$

where π_2 is the projection which returns m for $\alpha(s,e) = ((s',e'),m)$. Such a monad specifies the probability of executing every transition in a statechart.

For the simplest case, where B is the identity functor, the T-coalgebra corresponding to a particular statechart is defined as (CF, α), where CF is the set of configurations and

$$\alpha(G,E) = e \mapsto \begin{cases} ((G',E'), E' \setminus E) & if\ b(s) = true \wedge \\ & (G,E) \xrightarrow{e[b]} (G',E') \\ (G,E) & if\ b(s) = false \end{cases}$$

Then the behavior of SC is given by

$$[\![SC]\!] : \lambda s : CF \;.\; \alpha(s)$$

The overall behavior of the statechart is retrieved, of course, as the coinductive extension of α.

3. Equivalence of Statecharts

One major concept for the semantics of statecharts, is that of their equivalence. Given two different statecharts, can they be considered to be two different ways of modelling the same system? This question actually consists of two parts:

1. What is the criterion to affirm that two statecharts are equivalent?
2. How is the equivalence relation between statecharts defined formally?

Intuitively, two statecharts are equivalent to each other if they have the same semantics, i.e., from the syntax, we can derive the same behavior relevant information. Here the syntax of statecharts is given as EHA clearly, and the semantics is given by (final) coalgebras. Therefore, the definition of an equivalence relation both presupposes and implicitly induces the semantics: the relevant information about the behavior being specified by the statechart is precisely that which is common to all equivalent statecharts. So the semantics of a statechart is defined as its equivalence class, i.e., the final coalgebra.

For the formal definition of the equivalence relation between statecharts, the semantics of statecharts plays an important role and formal definitions of equivalence should be consistent with the semantics. Therefore, we take bisimulation into consideration, which is one basic notion in coalgebra theory [21], and without question the simplest way to define the behavioral equivalence of two statecharts.

Let $(CF_1, \alpha : CF_1 \to \mathsf{T}(CF_1))$ and $(CF_2, \beta : CF_2 \to \mathsf{T}(CF_2))$ be two coalgebras corresponding to statecharts SC_1 and SC_2, in the following we depict the bisimulation relationship between the two configuration spaces CF_1 and CF_2 which specifies their equivalent property.

Definition 3.1 *A binary relation $R \subseteq CF_1 \times CF_2$ between two set of configurations is a bisimulation iff the following diagram commutes:*

$$\begin{array}{ccccc} CF_1 & \xleftarrow{\pi_1} & R & \xrightarrow{\pi_2} & CF_2 \\ \downarrow \alpha & & \downarrow \gamma & & \downarrow \beta \\ \mathsf{T}(CF_1) & \xleftarrow{\mathsf{T}\pi_1} & \mathsf{T}(R) & \xrightarrow{\mathsf{T}\pi_2} & \mathsf{T}(CF_2) \end{array}$$

where π_1 and π_2 are the projection functions from R to CF_1 and CF_2 respectively. Two configurations s_1 and s_2 are equivalent if there is a bisimulation R such that $s_1 R s_2$.

Some standard properties of bisimulation apply:

Proposition 3.1 *For given sets of configurations CF_1, CF_2 and CF_3, we have*

1. *The identity relation on CF_1 is a bisimulation.*
2. *If $R \subseteq CF_1 \times CF_2$ is a bisimulation, then $R^{-1} \subseteq CF_2 \times CF_1$ is a bisimulation.*
3. *If $R \subseteq CF_1 \times CF_2$ is a bisimulation and $R' \subseteq CF_2 \times CF_3$ is a bisimulation, then the composition $R \circ R' \subseteq CF_1 \times CF_3$ is a bisimulation.*
4. *If $\{R_i \mid R_i \subseteq CF_1 \times CF_2\}_{i \in I}$ is a family of bisimulations, then the relation $R = \cup_{i \in I} R_i \subseteq CF_1 \times CF_2$ is a bisimulation.*

Corollary 3.1 *The set of all bisimulations between sets of configurations CF_1 and CF_2 forms a complete lattice.*

Since the arbitrary union of bisimulation relations between two set of configurations CF_1 and CF_2 is still a bisimulation, if there exists a bisimulation relation between two set of configurations, there also exists a greatest bisimulation. Such a greatest bisimulation can be defined as the union of all bisimulations for this type, leading to the following proof principle, again quite standard in coinductive reasoning:

Definition 3.2 *Two statecharts SC_1 and SC_2 are equivalent iff there is a bisimulation R between their sets of configurations CF_1 and CF_2 such that their initial configurations I_1 and I_2 satisfies $I_1 R I_2$.*

Example 3.1 *For two statecharts SC_1 and SC_2 as shown in Figure 1. Let*

$$\begin{aligned} R = \{&((s_{11}, s_{12}, p_1, p_3), (s_{21}, p_1, p_3)), \\ &((s_{11}, s_{12}, p_2, p_4), (s_{22}, p_2, p_4))\} \end{aligned}$$

IEEE COMPUTER SOCIETY

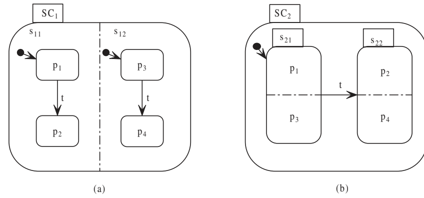

Figure 1. Equivalent Statecharts

It is easy to check that R is a bisimulation and their initial configurations are in R. Therefore, the two statecharts are equivalent to each other.

Note that the equivalence of statecharts captures the *two-way refinement relations*, where we require the behavior of the refined statechart and the abstract statechart to be exactly the same. In the following section, we give a notion of refinement on the *inequational* side, which is more useful in step-wise developments.

4. Refinement

We are now ready to define the notion of refinement for statecharts we are interested in. What we want to investigate first is whether a configuration is simulated (refined) by another one. Then we extend the notion to statecharts as well.

Generally, a refinement approach should be semantically compatible. That means, if a concrete statechart is a refinement of an abstract one, then the structure and the behavior specified by the abstract one should in a certain sense be preserved (or reflected) by the concrete one. This property is formulated via forward (and backward, respectively) morphism between coalgebras as discussed in [16].

The refined statechart may impose further functional and nonfunctional requirements in addition to those imposed by the given abstract statechart. In this paper, we only focus on behavior refinement which relates statecharts with the same event set. We first recall the definition of behavior refinement in the generic setting, then we instantiate it in the context of statecharts and give a family of refinement laws according to the definition.

4.1. What is Behavior Refinement?

The most fundamental notion of refinement underlying the coalgebraic framework is *behavior refinement*. In the following we recall the precise definition of behavior refinement in the generic context and the notion of simulation being used as a sound proof technique for behavior refinement [16].

In data refinement, there is a 'recipe' to identify a refinement situation: look for a *retrieve function* to witness it. I.e., a morphism in the relevant category, from the 'concrete' to the 'abstract' model such that the latter can be *recovered* from the former up to a suitable notion of equivalence, though, typically, not in a unique way. In [18] such a retrieve function is an epi and the 'suitable notion of equivalence' is, of course, $\mathbf{Set}$ isomorphism.

In our coalgebraic framework, however, things do not work this way. The reason is obvious: initial states preserving coalgebra morphisms are known (e.g., [21]) to entail bisimilarity. Therefore we have to look for some *weaker* notion of morphism between coalgebras. This is formalized by forward (and backward) morphisms between coalgebras ([16]), building on the notion of *order on an endofunctor*, which is defined in [8].

Given a $\mathbf{Set}$ endofunctor T, an order $\leq$ on T is defined as a functor $\leq$ from $\mathbf{Set}$ to $\mathbf{PreOrder}$ (concretely, mapping every set U into a collection of preorders $\leq_{\mathsf{T}(U)}$) making the following diagram to commute:

$$\begin{array}{ccc} & & (\mathsf{T}(U), \leq_{\mathsf{T}(U)}) \\ & \overset{\leq}{\nearrow} & \downarrow \mathsf{G} \\ U & \xrightarrow[\mathsf{T}]{} & \mathsf{T}(U) \end{array}$$

where G is the forgetful functor which forgets the preorder structure for every preordered set and gives its underlying set. In the sequel $\leq$ will be referred as a *refinement preorder*. Then,

Definition 4.1 *Let T be an extended polynomial functor on $\mathbf{Set}$ and consider two T-coalgebras $c = (U, \alpha : U \to \mathsf{T}(U))$ and $a = (V, \beta : V \to \mathsf{T}(V))$. A* forward *morphism $h : c \to a$ with respect to a refinement preorder $\leq$, is a function from U to V such that*

$$\mathsf{T}h \circ \alpha \leq \beta \circ h$$

Dually, h is called a backward *morphism if*

$$\beta \circ h \leq \mathsf{T}h \circ \alpha$$

It has been proved in [16] that forward (respectively, backward) morphism preserves (respectively, reflects) transition relations of coalgebras.

Forward and backward morphisms compose respectively and both can be taken as witnesses of refinement situations. Formally, behavior refinement between coalgebras is defined as the existence of a forward morphism *up to bisimilarity*.

Definition 4.2 *Given two coalgebras c and a, c is a* behavior refinement *of a, written $c \sqsubseteq_B a$, if there exist coalgebras p and q such that $c \sim p$, $a \sim q$ and there exists a (initial state preserving) forward morphism from p to q.*

IEEE COMPUTER SOCIETY

Example 4.1 *The exact meaning of a refinement assertion $c \sqsubseteq_B a$ depends on the refinement preorder $\leq$ adopted. For example, we can define the preorder for extended polynomial functor T by induction as follows:*

$$
\begin{array}{rll}
x \subseteq_{Id} y & \text{iff} & x = y \\
x \subseteq_{K} y & \text{iff} & x =_K y \\
x \subseteq_{\mathsf{T}_1 \times \mathsf{T}_2} y & \text{iff} & \pi_1 x \subseteq_{\mathsf{T}_1} \pi_1 y \ \wedge \ \pi_2 x \subseteq_{\mathsf{T}_2} \pi_2 y \\
x \subseteq_{\mathsf{T}_1 + \mathsf{T}_2} y & \text{iff} & \begin{cases} x = \iota_1 x' \wedge y = \iota_1 y' \Rightarrow \ x' \subseteq_{\mathsf{T}_1} y' \\ x = \iota_2 x' \wedge y = \iota_2 y' \Rightarrow \ x' \subseteq_{\mathsf{T}_2} y' \end{cases} \\
x \subseteq_{\mathsf{T}^K} y & \text{iff} & \forall k \in K.\, x(k) \subseteq_{\mathsf{T}} y(k) \\
x \subseteq_{\mathcal{P}\mathsf{T}} y & \text{iff} & \forall e \in x.\, \exists e' \in y.\, e \subseteq_{\mathsf{T}} e'
\end{array}
$$

Here π_i (ι_i, respectively) are the projection (injection) functions for the products (sums). A behavior refinement of non deterministic behavior based on $\subseteq_{\mathsf{T}}$ captures the classical notion of nondeterminism reduction.

A behavior refinement can be established by the existence of a simulation R connecting the state spaces of the 'concrete' and the 'abstract' coalgebras. Again, the notion of a simulation also depends on the adopted refinement preorder.

Definition 4.3 *([8]) Given a $\mathbf{Set}$ endofunctor T and a refinement preorder $\leq$, for two T-coalgebra $c = (U, \alpha : U \to \mathsf{T}U)$ and $a = (V, \beta : V \to \mathsf{T}V)$ define a lax relation lifting as an operation $Rel_{\leq}(\mathsf{T})$ on relations assigning a relation R to $\leq \circ \mathsf{T}(R) \circ \leq$, where $\circ$ is relational composition.*

A simulation is just a $Rel_{\leq}(\mathsf{T})$ coalgebra, i.e., a relation R such that, for all $u \in U, v \in V$, $\langle u, v \rangle \in R \Rightarrow \langle \alpha\, u, \beta\, v \rangle \in Rel_{\leq}(\mathsf{T})(R)$.

Note that the expression forward and backward simulations are used in data refinement [5, 10], where they are usually denoted by L and L^{-1}. Although our use of them is a bit more specific (corresponding to the existence of a forward (respectively, backward) morphism), the definition of simulation above includes both L and L^{-1}-simulation because $\leq$ is reflexive. Actually, if we let the first $\leq$ in $\leq \circ \mathsf{T}(R) \circ \leq$ be $=$, then the result simulation corresponds to L^{-1}-simulation, i.e. $\mathsf{T}(R) \circ \alpha \leq \beta \circ R$. If we let the second $\leq$ be $=$, then the result simulation corresponds to L-simulation, i.e. $\alpha \circ R^{-1} \leq (\mathsf{T}(R))^{-1} \circ \beta$. Furthermore, simulation provides a sound proof technique for behavior refinement. See [16] for the proof of the soundness of simulation for behavior refinement.

4.2. Refinement for Statecharts

This section presents some discussions on refinement for statecharts. We start by proposing a suitable refinement preorder and formalizing the notion of simulation for statecharts. According to the coalgebraic semantics of statecharts, the state spaces represent sets of configurations. Thus a simulation relation in the style of the Milner-Park simulation relation between abstract and concrete statecharts is given in the following definition.

Definition 4.4 *Let $SA = (F_A, E_A, \xi_A)$ be an abstract statechart and $SC = (F_C, E_C, \xi_C)$ a concrete statechart, $e \in E_A$ is an arbitrary event in E_A. The simulation relation R is defined as the greatest binary relation $R \subseteq CF_{SC} \times CF_{SA}$ satisfying the following conditions:*

1. *Non-determinism reduction: $(s_C R s_A \wedge s_C \xrightarrow{e} s'_C) \Rightarrow \exists s'_A \,.\, (s_A \xrightarrow{e} s'_A \wedge s'_C R s'_A)$;*
2. *Lack of new divergence: $s_C R s_A \Rightarrow \neg(\exists \{s_n\}_{n=1}^{\infty} \,.\, s_C = s_1 \wedge s_i \xrightarrow{\emptyset} s_{i+1}) \vee (\exists \{s'_n\}_{n=1}^{\infty} \,.\, s_A = s'_1 \wedge s'_i \xrightarrow{\emptyset} s'_{i+1})$;*
3. *Lack of new deadlocks: $(s_C R s_A \wedge \forall s'_C \in CF_{SC}, e \in E \,.\, (s_C \xrightarrow{e} s'_C) \notin En_{SC}(s_C)) \Rightarrow (\forall s'_A \in CF_{SA}, e \in E \,.\, (s_A \xrightarrow{e} s'_A) \notin En_{SA}(s_A))$;*
4. *Stuttering transition introduction: $(s_C R s_A \wedge s_C \xrightarrow{\emptyset} s'_C) \Rightarrow s'_C R s_A$;*
5. *Stuttering transition removing: $(s_C R s_A \wedge s_A \xrightarrow{\emptyset} s'_A) \Rightarrow s_C R s'_A$.*

The presence of condition 1 in Definition 4.4 is consistent with the refinement preorder given in Example 4.1. Condition 2 means that the refinement does not authorize new infinite loops with no interaction with the environment. Condition 3 implies that any deadlock in SC corresponds to a deadlock in SA, which means that new deadlocks are forbidden in the refinement. Define a trace as a finite sequence of external events in which a statechart participates with its environment, then Condition 4 together with Condition 1 mean that every trace of SC refines some trace of SA. Because idle steps modulo observation equivalence of states, the simulation can be obtained by either introducing or removing stuttering transitions. This is illustrated by Condition 5 together with Condition 4.

Similar to Example 4.1, we can define the refinement preorder for other conditions. For example, if we take the singleton set $\mathbf{1} = \{*\}$ for denoting deadlock, then condition 3 corresponds to the following preorder (The only difference with Example 4.1 is $\subseteq_{\mathsf{T}_1 + \mathsf{T}_2}$):

$$
x \subseteq_{\mathsf{T}_1 + \mathsf{T}_2} y \quad \text{iff} \quad \begin{cases} x = \iota_1 x' \wedge y = \iota_1 y' & \Rightarrow \ x' \subseteq_{\mathsf{T}_1} y' \\ x = \iota_2 * & \Rightarrow \ y = \iota_2 * \end{cases}
$$

Our notion of simulation for statecharts is based on one assumption which has not been made explicit so far. Basically, the events triggering transitions and the actions being generated are assumed to be atomic and terminating. That's

Proceedings of the Second International Conference on Software Engineering and Formal Methods (SEFM'04)
0-7695-2222-X/04 $ 20.00 IEEE
IEEE COMPUTER SOCIETY

why we do not consider *causality* here, by which one transition may generate new events which may in turn trigger some new transitions while disabling others. Without the assumption, it is difficult to reason about the effect of behavior of one statechart. However, the assumption does not imply that all events and actions have to be atomic and terminating. In other words, one macro-step does not have to be a micro-step. The differences of the micro-steps in one macro-step can be hidden from outside by a natural transformation, which is called a view in [9].

Simulations have some important properties, which are particularly useful for reasoning refinement.

Proposition 4.1 *For a statechart $SC = (F_C, E_C, \xi_C)$, the identity relation on CF_{SC}, denoted by $id_{SC} \subseteq CF_{SC} \times CF_{SC}$, is a simulation.*

Proposition 4.2 *Let $SA = (F_A, E_A, \xi_A)$, $SB = (F_B, E_B, \xi_B)$ and $SC = (F_C, E_C, \xi_C)$ be statecharts, For two simulations $R \subseteq CF_{SB} \times CF_{SA}$ and $R' \subseteq CF_{SC} \times CF_{SB}$, we have that $R' \circ R$ is still a simulation.*

Proposition 4.3 *Let $SA = (F_A, E_A, \xi_A)$ and $SC = (F_C, E_C, \xi_C)$ be two statecharts, for simulations $R \subseteq CF_{SC} \times CF_{SA}$ and $R' \subseteq CF_{SC} \times CF_{SA}$, $R \cup R'$ is still a simulation.*

In fact, for two given statecharts, there may be many possible simulations. For fixed statecharts satisfying the properties required by the definition of simulation, the least simulation is the empty relation, whereas the greatest simulation is obtained as the union of all simulations. We assume that all the simulations being used in the following are not empty.

Definition 4.5 *Let SC and SA be statecharts. A configuration $s_C \in CF_{SC}$* refines *(or* simulates*) a configuration $s_A \in CF_{SA}$, denoted by $s_C \sqsubseteq s_A$, if there is a simulation $R \subseteq CF_{SC} \times CF_{SA}$ containing the pair (s_C, s_A). Formally,*

$$\sqsubseteq = \bigcup \{R \mid R \text{ is a simulation}\}$$

Note that $\sqsubseteq$ is the empty relation when there is no associated simulation. From this definition, we conclude that in order to prove that a state s_A is simulated by a state s_C, it is enough to find a simulation containing the pair (s_C, s_A).

We now come to some properties of the refinement relation on configurations. First, from the definition, we can easily know that refinement contains all the possible simulations.

Proposition 4.4 *Let SC and SA be statecharts. The refinement relation $\sqsubseteq$ is the greatest simulation between their configurations.*

The refinement relation on configurations is reflexive and transitive, as indicated by the following proposition.

Proposition 4.5 *For statecharts SA, SB and SC,*

1. *for any $s_C \in CF_{SC}$, $s_C \sqsubseteq s_C$;*
2. *$s_C \sqsubseteq s_B$ and $s_B \sqsubseteq s_A$ imply $s_C \sqsubseteq s_A$, where $s_C \in CF_{SC}$, $s_B \in CF_{SB}$, $s_A \in CF_{SA}$.*

Based on the notion of refinement of configurations, we can define refinement for statecharts.

Definition 4.6 *For two statecharts SC and SA, we say that SC is a* behavior refinement *of SA, written $SC \sqsubseteq SA$, if there is a simulation $R \subseteq CF_{SC} \times CF_{SA}$ containing the pair of their initial configurations (I_{SC}, I_{SA}).*

Similarly to refinement of configurations, refinement of statecharts also has reflexive and transitive properties.

Proposition 4.6 *For statecharts SA, SB and SC,*

1. *$SC \sqsubseteq SC$;*
2. *$SC \sqsubseteq SB$ and $SB \sqsubseteq SA$ imply $SC \sqsubseteq SA$.*

4.3. Discussions on Initial Configurations and Reachability

We have previously assumed there is only one initial configuration in one statechart. However, there may be several initial configurations. In general, one could allow, for each statechart, a set of initial configurations. For example, the statechart in Figure 2 has two initial configurations $\{s_0, s_1\}$ and $\{s_0, s_2\}$, and satisfies the well-formedness rules of statecharts. This does not fundamentally change the idea. When we refine a statechart with multiple initial configurations, we only need to find a simulation relationship containing all the initial configurations of the abstract statechart. For simplicity, we still assume the uniqueness of initial configuration.

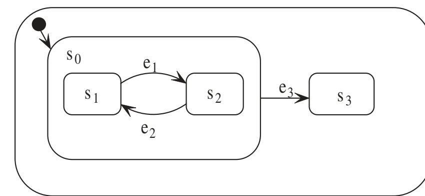

Figure 2. Statecharts with multiple initial configurations

In Definition 4.6, it is required that the pair (I_{SC}, I_{SA}) of initial configurations must be included in the simulation R. This is a rather restricted condition. In general, we only

IEEE COMPUTER SOCIETY

need to require that I_{SA} is simulated by some configuration in CF_{SC}, which is reachable from I_{SC} by taking a sequence of transitions.

By choosing for the simulation relation R the empty relation, one can always prove that there is a simulation between any two arbitrary statecharts. Such an anomaly can be ruled out by adding the reachability requirement: The initial configuration of the abstract statechart must be related to some configuration of the concrete one which is reachable from the initial configuration of the concrete statechart. This is indeed a reasonable obligation for statechart refinement.

4.4. Refinement Laws

Let us look at some refinement laws for statecharts. Additional conditions are often added whose validity is required to ensure that a law can be successfully applied. While from the theoretical point of view a complete and powerful set of refinement laws might be desirable, from the practical point of view it is more important that these conditions can be effectively checked.

The laws we give here are very elementary. Their full power only reveals by their adequate composition to more powerful refinement laws.

Law 1 *A development process may start by creating a new statechart with arbitrary states and transitions.*

$$\forall SC \ . \ SC \sqsubseteq \emptyset.$$

Law 2 *Adding a state to an existing statechart is a refinement:*

$$SC' = (F', E', \xi') \sqsubseteq SC = (F, E, \xi)$$

if $E' = E$, $\exists A \in F,\ A' \in F'$, *such that*

$$(F \setminus \{A\} = F' \setminus \{A'\}) \wedge (\Sigma_{A'} = \Sigma_A \cup \{\sigma\} \wedge s^0_{A'} = s^0_A \wedge L_{A'} = L_A \wedge \delta_{A'} = \delta_A)$$

where σ *is the state being added to* SC, *and* $\forall \sigma' \in \bigcup_{A \in F} \Sigma_A \ . \ \xi'(\sigma') = \xi(\sigma')$.

Law 3 *Removing an unreachable state from a statechart is a refinement:*

$$SC' = (F', E', \xi') \sqsubseteq SC = (F, E, \xi)$$

if $E' = E$, $\exists A \in F,\ A' \in F'$, *such that*

$$(F \setminus \{A\} = F' \setminus \{A'\}) \wedge \exists \sigma \in \Sigma_A \ . \ (\Sigma_{A'} = \Sigma_A \setminus \{\sigma\} \wedge \forall s = (G, E) \in CF_{SC} \ . \ \sigma \notin G)$$

and $\forall \sigma' \in \bigcup_{A' \in F'} \Sigma_{A'} \ . \ \xi'(\sigma') = \xi(\sigma')$.

Law 4 *We allow for the refinement of a state into a more fine grained set of states (statecharts).*

$$SC' = (F', E', \xi') \sqsubseteq SC = (F, E, \xi)$$

if $E' = E$, $F \subseteq F'$, $\exists A \in F,\ A' \in F' \setminus F$, *such that*

$$\exists \sigma \in \Sigma_A \ . \ \xi(\sigma) = \emptyset \wedge A' \in \xi'(\sigma)$$

and $\forall \sigma' \in \bigcup_{A \in F} \ . \ \sigma' \neq \sigma \Rightarrow \xi'(\sigma') = \xi(\sigma')$.

Law 5 *The addition of transitions is a refinement if so far no corresponding transitions exist.*

$$SC' = (F', E', \xi') \sqsubseteq SC = (F, E, \xi)$$

if $\mathcal{T}SC \subseteq \mathcal{T}SC'$, *and* $\forall t' \in \mathcal{T}SC' \setminus \mathcal{T}SC$, $\nexists t \in \mathcal{T}SC$, *such that*

$$src(t) = src(t') \wedge ev(t) = ev(t') \wedge g(t) = g(t').$$

Law 6 *Removal of transitions is a refinement if alternative transitions exist.*

$$SC' = (F', E', \xi') \sqsubseteq SC = (F, E, \xi)$$

if $\mathcal{T}SC' \subseteq \mathcal{T}SC$, *and* $\forall t \in \mathcal{T}SC \setminus \mathcal{T}SC'$, $\exists t' \in \mathcal{T}SC'$, *such that*

$$src(t) = src(t') \wedge ev(t) = ev(t') \wedge g(t) = g(t') \wedge ac(t) = ac(t').$$

Law 7 *Changing the target of a transition entering a non-concurrent composite state from the composite state to one of its substate is a refinement if every other substate of the composite state is reachable from this substate.*

$$SC' = (F', E', \xi') \sqsubseteq SC = (F, E, \xi)$$

if $\exists t \in \mathcal{T}SC$, $\exists t' \in \mathcal{T}SC'$, *such that*

$$\mathcal{T}SC \setminus \{t\} = \mathcal{T}SC' \setminus \{t'\} \wedge src(t) = src(t') \wedge ev(t) = ev(t') \wedge g(t) = g(t') \wedge ac(t) = ac(t') \wedge tgt(t') \prec tgt(t)$$

and $\forall s \prec tgt(t)$, $s \neq tgt(t')$, $\exists \{t_i\}_{i=1,2,\cdots,n} \subseteq \mathcal{T}\xi(tgt(t))$, *such that* $src(t_1) = tgt(t')$, *for* $i = 1, 2, \cdots, n-1$, $tgt(t_i) = src(t_{i+1})$, *and* $tgt(t_n) = s$.

Law 8 *Changing the source of a transition exiting a substate of one non-concurrent composite state from the substate to the composite state is a refinement.*

$$SC' = (F', E', \xi') \sqsubseteq SC = (F, E, \xi)$$

if $\exists t \in \mathcal{T}SC$, $\exists t' \in \mathcal{T}SC'$, *such that*

$$\mathcal{T}SC \setminus \{t\} = \mathcal{T}SC' \setminus \{t'\} \wedge src(t) \prec src(t') \wedge ev(t) = ev(t') \wedge g(t) = g(t') \wedge ac(t) = ac(t') \wedge tgt(t) = tgt(t').$$

IEEE COMPUTER SOCIETY

5. Conclusions

This paper discusses a coalgebraic semantics for UML statecharts, which is consistent with an extension of its operational semantics given in [13]. Then we define an equivalence between statecharts which captures a two-way refinement relation. After that, we investigate the notion of behavior refinement generically and its instantiation for statecharts. Finally, a family of refinement laws for statecharts is given.

A fundamental point to be remarked in concluding this paper, is that the notion of "refinement" typically used in the UML specification [19] is different from the usage of this term in literature, e.g. [20]. Our proposal goes a bit far, based on the general refinement mechanism proposed in [16].

Although equivalence of statecharts has already been investigated by a number of authors (see, e.g., [14]), there is still a lack of a formal definition of refinement for statecharts and corresponding refinement laws. One of the benefits of the notion of refinement discussed in this paper is that one can verify not only the equality between statecharts, but also when a statechart behaves better than another, a quite common situation in UML-like software development. Bisimulation and simulation provide the proof techniques for, respectively, equivalence and refinement under this coalgebraic framework. Furthermore, the refinement laws provide a syntax-based approach for constructing implementations from abstract software specifications written in statecharts, which is consistent with the semantics. Results given here should be useful in reasoning and transforming component-based designs.

For sake of brevity, transitions in a statechart triggered by timing event are not described in this paper. One immediate topic for our future work is to specify them as monoid actions in this coalgebraic framework, like in [7], and make the description more fine-grained.

References

[1] L. S. Barbosa and S. Meng. Generic components. In G. Hutton, editor, *Proceedings of First APPSEM-II Workshop*, Nottingham, March 2003. APPSEM Network Report.

[2] A. Corradini, R. Heckel, and U. Montanari. Compositional SOS and beyond: a coalgebraic view of open systems. *Theoretical Computer Science*, 280:163–192, 2002.

[3] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[4] D. Harel and A. Naamad. The STATEMATE Semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333, 1996.

[5] C. A. R. Hoare, H. Jifeng, and J. W. Sanders. Prespecification in data refinement. *Information Processing Letters*, 25:71–76, 1987.

[6] B. Jacobs. Objects and classes, co-algebraically. In B. Freitag, C. Jones, C. Lengauer, and H.-J. Schek, editors, *Object-Orientation with Parallelism and Persistence*, pages 83–103. Kluwer, 1996.

[7] B. Jacobs. Object-oriented hybrid systems of coalgebras plus monoid actions. *Theoretical Computer Science*, 239:41–95, 2000.

[8] B. Jacobs and J. Hughes. Simulations in coalgebra. In H. P. Gumm, editor, *Elect. Notes in Theor. Comp. Sci. (CMCS'03 - Workshop on Coalgebraic Methods in Computer Science)*, volume 82, pages 245–263, Warsaw, April 2003.

[9] B. Jacobs and H. Tews. Assertional and Behavioural Refinement in Coalgebraic Specification. In *Electronic Notes in Theoretical Computer Science*, volume 47. Elsevier Science Publishers, 2001.

[10] H. Jifeng. Process simulation and refinement. *Formal Aspects of Computing*, 1(3):229–241, 1989.

[11] J. Jürjens. Formal semantics for interacting UML subsystems. In B. Jacobs and A. Rensink, editors, *5th International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS 2002)*, pages 29–44. Kluwer Academic Publishers, March 2002.

[12] A. Kock. Strong functors and monoidal monads. *Archiv für Mathematik*, 23:113–120, 1972.

[13] D. Latella, I. Majzik, and M. Massink. Towards a formal operational semantics of UML statechart diagrams. In *Proc. FMOODS'99*. Kluwer, 1999.

[14] A. Maggiolo-Schettini, A. Peron, and S. Tini. Equivalences of Statecharts. In *7th International Conference on Concurrency Theory (CONCUR'96)*, volume 1119 of *LNCS*, pages 687–702. Springer, August 1996.

[15] S. Meng and B. K. Aichernig. Towards a Coalgebraic Semantics of UML: Class Diagrams and Use Cases. Technical Report 272, UNU/IIST, January 2003.

[16] S. Meng and L. S. Barbosa. On Refinement of Generic State-based Software Components. In *Proceedings of 10th International Conference on Algebraic Methods And Software Technology, AMAST'04*, LNCS. Springer, 2004.

[17] E. Mikk, Y. Lakhnech, and M. Siegal. Hierarchical automata as models for statecharts. In R. Shyamasundar and K. Euda, editors, *Third Asian Computing Science Conference. Advances in Computing Science - ASIAN'97*, volume 1345 of *LNCS*, pages 181–196. Springer, 1997.

[18] J. N. Oliveira. A reification calculus for model-oriented software specification. *Formal Aspects of Computing*, 2(1):1–23, 1990.

[19] OMG. *OMG Unified Modeling Language Specification, Version 1.4* , 2001.

[20] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1998.

[21] J. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249:3–80, 2000.

[22] B. Selic, G. Gullekson, and P. Ward. *Real-time Object Oriented Modeling and Design*. J. Wiley & Sons, 1994.

IEEE COMPUTER SOCIETY