TOWARDS A FORMAL OPERATIONAL SEMANTICS OF UML STATECHART DIAGRAMS*

Diego Latella†
Istvan Majzik‡
Mieke Massink§

Abstract: Statechart Diagrams are a notation for describing behaviours in the framework of UML, the Unified Modeling Language of object-oriented systems. UML is a semi-formal language, with a precisely defined syntax and static semantics but with an only informally specified dynamic semantics. UML Statechart Diagrams differ from classical statecharts, as defined by Harel, for which formalizations and results are available in the literature. This paper sets the basis for the development of a formal semantics for UML Statechart Diagrams based on Kripke structures. This forms the first step towards model checking of UML Statechart Diagrams. We follow the approach proposed by Mikk and others: we first map Statechart Diagrams to the intermediate format of extended hierarchical automata and then we define an operational semantics for these automata. We prove a number of properties of such semantics which reflect the design choices of UML Statechart Diagrams.

*The work described in this paper has been performed in the context of the ESPRIT Project n. 27439 - HIDE.
†C.N.R., Ist. CNUCE, Pisa, Italy, d.latella@cnuce.cnr.it
‡Technical University of Budapest, Dept. of Measurement and Information Systems, Budapest, Hungary, majzik@mit.bme.hu. Partially supported by the CNR-NATO Guest Fellowship Programme.
§University of York, Dept. of Computing, York, United Kingdom, mieke@cs.york.ac.uk. Supported by the TACIT network under the European Union TMR Programme, Contract ERB FMRX CT97 0133.

INTRODUCTION

The Unified Modeling Language (UML) is a general-purpose, object-oriented, visual modeling language that is designed to specify, visualize, construct and document the artifacts of a software system [4].

Since UML is a graphical notation, a UML model (i.e. a specification of a system) is composed of different kinds of diagrams each representing a different view (or part) of the system. Use case diagrams show the relationships among actors, such as users or other systems, and basic functions of the system. Class diagrams show the types and interfaces of elements of the problem domain, while object diagrams present instances of classes. Sequence diagrams describe the interactions among objects, collaboration diagrams also include links to these objects. Statechart and activity diagrams capture the behavior of objects, systems or subsystems by presenting their internal states and reactions to (external) events. Component diagrams show the structure of the software components (program code) and dependencies among them. Deployment diagrams show the physical layout of components on hardware nodes.

UML is a semi-formal language, since its syntax and static semantics (the model elements, their interconnection and well-formedness) are defined precisely [4], but its dynamic semantics are specified neither formally nor algorithmically.

The work described in this paper has been performed in the context of the EC founded project HIDE. The aim of HIDE is the extension of UML design environments with transformations (and tools) from UML to dependability assessment models as well as semantics models for formal verification. UML provides a front-end for the designer to specify the system and its requirements. The models for dependability assessment and formal verification will be derived automatically by means of translations from UML models. The results will be back-annotated into the same UML model, this way eliminating the need of both costly re-modeling of the system and expertise in underlying formalisms.

The scope of our investigation are UML statechart diagrams. By using a statechart, the behavior of any classifier in UML, such as use cases, classes (objects) and hardware nodes can be specified. We focus on specifications made up of a single statechart. The interaction of subsystems or model elements specified by separate statecharts is not being considered here. Moreover, issues which we do not deal with in this paper and which we leave for future research are those related to “object orientation”, like inheritance, sub-behavior etc. Notice, however, that our current work is a mandatory prerequisite for any formal treatment of all these advanced features. In fact, in order to be able to perform any formal verification, the first step is obviously to map statechart diagrams to a formal semantics model. This is the subject of the present paper. The verification technique we aim at is model checking, thus, following the approach of [8], we choose to use Kripke structures as semantics models. To define the mapping, we use an intermediate model, which is a slightly modified variant of *Extended Hierarchical Automata*, proposed in [8]. We chose to follow the above mentioned approach because it seems to be one of the simplest ones.

The next section is a short introduction to UML statecharts, restricted to those features we will consider in this paper. Then, in the following section, the intermediate model is introduced and an example of the translation from UML statecharts to such

a model is given. A sketch of the translation is provided in the Appendix. The subsequent section introduces our formal operational semantics of Extended Hierarchical Automata. Finally, the conclusions summarize our work and sketch our lines of future research.

For space reasons, proofs are omitted here. The fully detailed proofs can be found in [6].

A SUBSET OF UML STATECHART DIAGRAMS

UML statecharts is an (object-oriented) variant of classical Harel statecharts [1, 2]. The statecharts formalism itself is an extension of traditional state transition diagrams. In this section we will briefly describe those features of UML statecharts diagrams which are of interest for this paper. We will describe them by means of the example of Fig. 1. In order to keep the description simple, we will keep it at a very superficial level. The detailed description of UML statecharts diagrams can be found in [5] and [4].

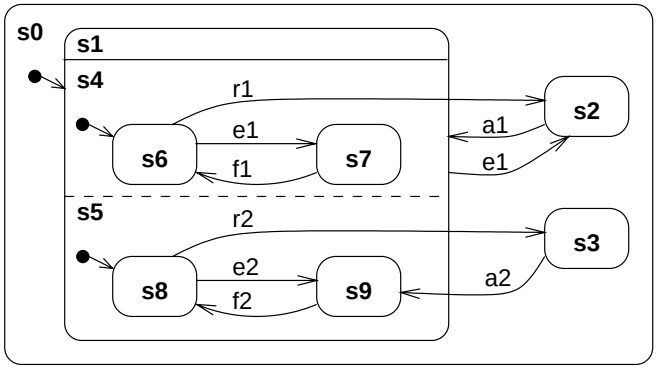

Figure 1 Example of an UML statechart

One of the main notions of statecharts is the notion of state refinement. In Fig. 1 state $s0$ is refined into an automaton consisting of three states, $s1$, $s2$, and $s3$. State $s1$ is further refined into two states, namely $s4$ and $s5$, each of them refined in turn into a distinct automaton. States like $s0$, $s1$, $s4$ and $s5$ are called *composite* and in particular $s1$ is a said to be *concurrent*.

A transition connects a *source* to a *target* state. The transition is labeled by a trigger event, a boolean guard and a sequence of actions. In our example, only trigger events are used.

"System states" are modeled by *configurations*, which are sets of states. For instance, our system can be in any of the following configurations: $\{s1, s6, s8\}$, $\{s1, s6, s9\}, \{s1, s7, s8\}, \{s1, s7, s9\}, \{s2\}, \{s3\}$.

A transition is enabled and can fire if and only if its source state is in the current *configuration*, its trigger is offered by the external environment and the guard is satisfied. In this case the source state is left, the actions are executed, and the target state is entered.

In our example, if event $a1$ is given and the current configuration is $\{s2\}$, state $s2$ is left and state $s1$ is entered. In particular, being $s1$ composite, we also have to say which are the particular *sub-states* which are reached. In the case at hand they are the default ones, i.e. the *initial* states of $s4$ and $s5$, namely $s6$ and $s8$.

In the general case, some target substates can be explicitly specified. In our example, when the current configuration is $\{s3\}$ and event $a2$ is offered, the configuration resulting from firing the transition labeled by $a2$ will be $\{s1, s6, s9\}$, where $s9$ is explicitly pointed to by the transition. A transition like the above mentioned one is called an *interlevel* transition and such transitions can in general have more than one target in order to explicitly point to all relevant states (*fork* transitions).

Symmetrically, also the transition from $s6$ to $s2$ and the one from $s8$ to $s3$ are interlevel. Firing, say, the first one requires the system to be in a configuration containing $s6$, regardless of the state in which $s5$ resides, and brings it to state $s2$. Interlevel transitions can also have more than one source state, the meaning being that all such states must be in the current configuration for the transition to be enabled (*join* transitions). *Compound* transitions can be either join or fork transitions.

In general, more than one event can be available in the environment. The UML semantics assumes then a *dispatcher* which selects an event at a time from the environment, modeled as a queue, and offers it to the state machine. In general, more than one transition can be enabled at this point. Some of them can be in conflict: this happens when the intersection of the sets of states left by the transitions is not empty. Some conflicts can be resolved using priorities. Roughly speaking a transition has higher priority than another transition if its source state is a substate of the source of the other one. For instance, if the statechart of Fig. 1 is in a configuration containing both $s1$ and $s6$, and the event selected by the dispatcher is $e1$ then the transition from $s6$ to $s7$ will be fired since it has higher priority than the one to $s2$. If the conflict cannot be resolved using priorities, then any of the conflicting enabled transitions may be fired. Due to concurrent states, it is possible that more than a single transition is fired as a reaction to a given event. In particular the set of transitions that will fire is a maximal set of enabled, non-conflicting transitions, such that no enabled transition outside the set has higher priority than a transition in the set. When the effects of all such transitions and related actions are complete a new event is selected by the dispatcher and a new cycle is started.

In this paper, we will refer to a quite restricted subset of UML statechart diagrams which, nevertheless, includes all the interesting conceptual issues related to concurrency in the dynamic behaviour, like sequentialization, non-determinism and parallelism.

More specifically, we do not consider history, action and activity states; we restrict events to signal and call ones, without parameters; time and change events, object creation and destruction events and deferred events are not considered as well as branch transitions; variables and data are not allowed so that actions are required to be just (a sequence of) events. We also abstract from entry and exit actions of states.

The above simplifications are made essentially for keeping the level of readability of the paper acceptable since, in our opinion, most of them do not have any strong impact on the semantics at a conceptual level.

Other limitations, like the fact that we do not deal with the object-oriented features of UML statechart diagrams, e.g. sub-behaviours, etc, are more serious and we leave them for further study, together with extensions like deterministic/stochastic time ones.

EXTENDED HIERARCHICAL AUTOMATA

In this section we recall the notion of Extended Hierarchical Automata defined in [8], although our notation is slightly different from that used therein, and we informally show how Extended Hierarchical Automata can faithfully represent statecharts. We start by the notion of (sequential) automaton[1].

Def. 1 (Sequential Automata) *A sequential automaton A is a 4-tuple $(\sigma_A, s^0_A, \lambda_A, \delta_A)$ where σ_A is a finite set of* states *with $s^0_A \in \sigma_A$ the* initial state, *λ_A is a finite set of* transition labels *and $\delta_A \subseteq \sigma_A \times \lambda_A \times \sigma_A$ is the* transition relation.

We shall use a particular structure for the labels in λ_A which will be described later. For sequential automaton A let functions $SRC, TGT : \delta_A \longrightarrow \sigma_A$ be defined as $SRC(s, l, s') = s$ and $TGT(s, l, s') = s'$. Extended Hierarchical Automata [8] are defined as follows:

Def. 2 (Extended Hierarchical Automata) *An extended hierarchical automaton H is a 3-tuple (F, E, ρ), where F is a finite set of sequential automata with mutually disjoint sets of states, i.e. $\forall A_1, A_2 \in F.\ \sigma_{A_1} \cap \sigma_{A_2} = \emptyset$ and E is a finite set of* events*; the* refinement function *$\rho : \bigcup_{A \in F} \sigma_A \longrightarrow 2^F$ imposes a tree structure to F, i.e. (i) there exists a unique root automaton $A_{root} \in F$ such that $A_{root} \notin \bigcup rng\ \rho$, (ii) every non-root automaton has exactly one ancestor state: $\bigcup rng\ \rho = F \setminus \{A_{root}\}$ and $\forall A \in F \setminus \{A_{root}\}.\ \exists_1 s \in \bigcup_{A' \in F \setminus \{A\}} \sigma_{A'}.\ A \in (\rho\ s)$ and (iii) there are no cycles: $\forall S \subseteq \bigcup_{A \in F} \sigma_A.\ \exists s \in S.\ S \cap \bigcup_{A \in \rho\ s} \sigma_A = \emptyset$.*

We say that a state s for which $\rho\ s = \emptyset$ holds is a *basic* state. An example of an extended hierarchical automaton is presented in Figure 2. Here $F = \{A0, A1, A2\}$, and state $s1$ of the root $A0$ is refined by $A1$ and $A2$: $\rho\ s1 = \{A1, A2\}$. All states except $s1$ are basic. Initial states are indicated by double boxes.

In the sequel we will implicitly make reference to a generic extended hierarchical automaton $H = (F, E, \rho)$.

Every sequential automaton $A \in F$ characterizes an extended hierarchical automaton in its turn: intuitively, such an extended hierarchical automaton is composed by all those sequential automata which lay below A, including A itself, and has a refinement function ρ_A which is a proper restriction of ρ.

Def. 3 *For $A \in F$ the* automata, states, *and* transitions under *A are defined respectively as $\mathcal{A}\ A = \{A\} \cup (\bigcup_{A' \in \rho_A\ \sigma_A} (\mathcal{A}\ A'))$, $\mathcal{S}\ A = \bigcup_{A' \in \mathcal{A}\ A} \sigma_{A'}$, and $\mathcal{T}\ A = \bigcup_{A' \in \mathcal{A}\ A} \delta_{A'}$*

[1] In the following we will freely use a functional-like notation in our definitions where: (i) currying will be used in function application, i.e. $f\ a_1\ a_2 \ldots\ a_n$ will be used instead of $f(a_1, a_2, \ldots, a_n)$ and function application will be considered left-associative; (ii) for function $f : X \longrightarrow Y$ and $Z \subseteq X$, $f\ Z = \{y \in Y \mid \exists x \in Z.\ y = f x\}$, $rng\ f$ denotes the *range* of f and $f_{|Z}$ is the restriction of f to Z.

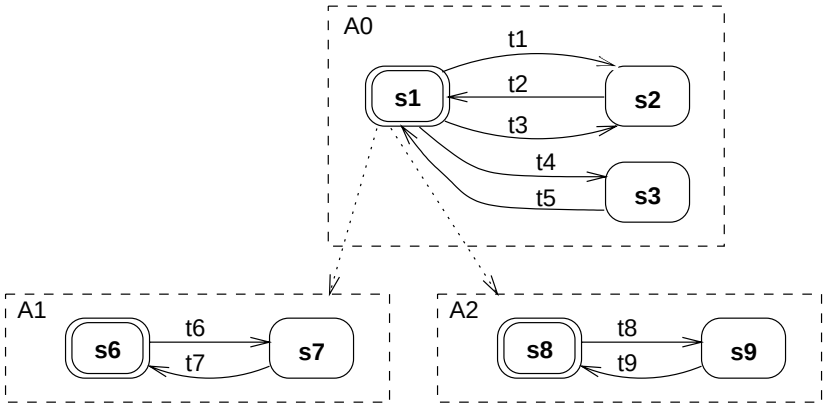

Figure 2 Example of an Extended Hierarchical Automaton

The definition of sub- extended hierarchical automaton follows:

Def. 4 *For $A \in F$, (F_A, E, ρ_A), where $F_A = (\mathcal{A}\ A)$ and $\rho_A = \rho_{|(\mathcal{S}\ A)}$, is the sub-extended hierarchical automaton characterized by A.*

In the sequel for $A \in F$ we shall refer to A both as a sequential automaton and as the sub-extended hierarchical automaton of H it characterizes, the role being clear from the context. H will be identified with A_{root}. Sequential Automata will be considered a degenerate case of Extended Hierarchical Automata. In Figure 2, automaton $A0$ refers to both the sequential automaton $A0 = (\{s1, s2, s3\}, s1, \lambda_A, \{t1, t2, t3, t4, t5\})$ and the extended hierarchical automaton $H = (\{A0, A1, A2\}, E, \rho)$ where $\rho\ s1 = \{A1, A2\}$.

Def. 5 (State Precedence) *For $s, s' \in \mathcal{S}\ H$, $s \prec s'$ iff $s' \in \mathcal{S}\ (\rho\ s)$. Let $\preceq$ denote the* reflexive *closure of $\prec$.*

Proposition 1 *Relation $\preceq$ is a partial order.*

Def. 6 (Orthogonal States) *Two states $s, s' \in \mathcal{S}\ H$ are* orthogonal, *written $s \ ||\ s'$, iff $\exists s'' \in (\mathcal{S}\ H), A, A' \in (\rho\ s'').\ A \neq A' \wedge s \in \mathcal{S}\ A \wedge s' \in \mathcal{S}\ A'$*

Obviously $s \ ||\ s'$ implies $s \neq s'$. For instance, with reference to our example, $s6$ and $s8$ are orthogonal, since $s6 \in \mathcal{S}\ A1$, $s8 \in \mathcal{S}\ A2$ and there is $s1$ for which $A1, A2 \in \rho\ s1$.
It is easy to see that orthogonal states satisfy the following property:

Lemma 1 *For all $s, s' \in \mathcal{S}\ H$ the following holds: $s \ ||\ s' \ \Rightarrow\ s \not\preceq s'$*

We say that $S \subseteq \mathcal{S}\ H$ is a *set of pairwise orthogonal states* iff $\forall s, s' \in S.\ (s \neq s' \Rightarrow s \ ||\ s')$. An obvious consequence of the above lemma is that for $S \subseteq \mathcal{S}\ H$ a

t	$t1$	$t2$	$t3$	$t4$	$t5$	$t6$	$t7$	$t8$	$t9$
$EV\ t$	$r1$	$a1$	$e1$	$r2$	$a2$	$e1$	$f1$	$e2$	$f2$
$SR\ t$	$\{s6\}$	$\emptyset$	$\emptyset$	$\{s8\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$TD\ t$	$\emptyset$	$\{s6, s8\}$	$\emptyset$	$\emptyset$	$\{s6, s9\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

set of pairwise orthogonal states, the following holds: $s, s' \in S$ and $s \preceq s'$ implies $s = s'$. Now we lift $\preceq$ to *sets of* states:

Def. 7 *For all* $S, S' \subseteq \mathcal{S}\ H$, $S \preceq^s S'$ *iff* $\forall s \in S.\ \exists s' \in S'.\ s \preceq s'$

Notice that $\preceq^s$ is only a preorder. Take for instance $S = \{s, s'\}$ and $S' = \{s'\}$ with $s \preceq s'$. Now $S \preceq^s S'$ and $S' \preceq^s S$, but $S \neq S'$. The following proposition holds:

Lemma 2 *For* $S, S' \subseteq \mathcal{S}\ H$ *sets of* pairwise orthogonal states $S \preceq^s S' \wedge S' \preceq^s S$ *implies* $S = S'$.

For the purpose of representing statechart diagrams using Extended Hierarchical Automata we shall require transition labels of transitions t of sequential automata $A \in F$ be 5-tuples (sr, ev, g, ac, td) where (i) the *source restriction* $sr \subseteq \mathcal{S}\ (\rho(SRC\ t))$ is a set of pairwise orthogonal states; (ii) $ev \in E \cup \{-\}$ is the event which *triggers* the transition, with $-$ representing that no event is required for triggering the transition; (iii) g is the *guard*, i.e. a boolean expression on states (which we shall not further specify in this paper); (iv) $ac \in E^*$ is the sequence of events to be generated when the transition is fired, i.e. the *sequence of actions* to be executed; and the *target determinator* $td \subseteq \mathcal{S}\ (\rho(TGT\ t))$ is a *maximal* (under set inclusion) set of pairwise orthogonal *basic* states. The target determinator and source restriction play a major role in the representation of compound/interlevel transitions, as we shall explain in a moment.

It should be already clear that the extended hierarchical automaton of Fig. 2 could be taken as an alternative representation for the statechart of Fig. 1. In fact there is a clear correspondence between the states of the two structures. Also the refinement of a state into one or more substates in the statechart is properly represented by the refinement function ρ. Non-interlevel transitions are represented in the obvious way. Consider now the interlevel transition from $s6$ to $s2$ in Fig. 1. Such a transition is represented in the extended hierarchical automaton by the transition from $s1$ (the highest ancestor of $s6$ "crossed" by the transition in the statechart) to $s2$, labeled by $t1$. The source restriction of such a transition will be $s6$. In general, for join transitions the source restriction will be a set of pairwise orthogonal states. The target determinator explicitly lists *all* the basic states which must be reached when a transition is fired. For example, the transition from $s3$ to $s9$ in Fig. 1 is represented in Fig. 2 by the transition labeled $t5$, the target determinator of which is $\{s6, s9\}$. Notice that actually $s6$ could be left out of the above set since it is an initial (i.e. "default") state. Here, for uniformity reasons, we prefer to list all the states, at the cost of a little redundancy. The table above completes the translation for our current example (omitting guards and actions).

In the sequel we shall use the following functions SR, EV, G, AC, TD defined in the obvious way; for transition $t = (s, (sr, ev, g, ac, td), s')$, $SR\ t = sr, EV\ t =$

$ev, G\ t = g, AC\ t = ac, TD\ t = td$. Finally, for transition $t \in \delta_A$ for $A \in F$ let $ORIG\ t$ be defined as follows:

$$ORIG\ t = \{s \mid s \in (SRC\ t) \wedge (SR\ t) = \emptyset\} \cup (SR\ t)$$

The following definition establishes when two transitions are *conflicting*:

Def. 8 *For $t, t' \in (\mathcal{T}\ H)$, t is conflicting with t', written $t\#t'$, iff* $t \neq t'$ *and* $(SRC\ t \preceq SRC\ t') \vee (SRC\ t' \preceq SRC\ t)$

The following lemma relates orthogonality and conflict:

Lemma 3 *For $t, t' \in (\mathcal{T}\ H)$ the following holds:* $(SRC\ t)\ ||\ (SRC\ t')$ *implies* $\neg(t\#t')$.

The following definition characterizes those structures which can be used for imposing priorities on transitions.

Def. 9 (Priority Schema) *A* Priority Schema *is a triple* $(\Pi, \sqsubseteq, \pi)$ *with* $(\Pi, \sqsubseteq)$ *a* partial order *and* $\pi : (\mathcal{T}\ H) \rightarrow \Pi$ *such that:* $\forall t, t' \in (\mathcal{T}\ H).\ (\pi\ t \sqsubseteq \pi\ t') \wedge t \neq t' \Rightarrow t\#t'$ *We say that t has lower priority than (equal priority as) t' iff* $\pi\ t \sqsubseteq \pi\ t'$.

The following lemma relates orthogonality and priority:

Lemma 4 *For $t, t' \in (\mathcal{T}\ H)$ the following holds:* $(SRC\ t)\ ||\ (SRC\ t')$ *implies* $\pi\ t \not\sqsubseteq \pi\ t'$.

The priority system we use in this paper is based on the origin of transitions. Let $PWO = \{X \subseteq (\mathcal{S}\ H) \mid X\ pairwise\ orthogonal\}$ and function f defined as $f\ t = ORIG\ t$.

Proposition 2 $(PWO, \preceq^s, f)$ *is a priority schema.*

FORMAL OPERATIONAL SEMANTICS

In this section we develop a formal semantics for Extended Hierarchical Automata which is different from that proposed in [8] in that it has to deal with the peculiarities of UML statechart diagrams. The main difference is the need to deal explicitly with priorities since UML priority rules do not directly match the hierarchical structure of Extended Hierarchical Automata, as it is the case with classical statecharts. Moreover, the environment is treated differently.

Operational Semantics Rules

We first define *configurations*. A configuration denotes a global state of an extended hierarchical automaton, composed of local states of component sequential automata.

Def. 10 (Configurations) *A* configuration *of H is a set $C \subseteq (\mathcal{S}\ H)$ such that (i)* $\exists_1 s \in \sigma_{A_{root}}.\ s \in C$ *and (ii)* $\forall s, A.\ s \in C \wedge A \in \rho\ s \Rightarrow \exists_1 s' \in A.\ s' \in C$

For $A \in F$ the set of all configurations of A is denoted by Conf_A. Possible configurations of the extended hierarchical automaton of Fig. 2 are: $\{s2\}$, $\{s1, s6, s8\}$, $\{s1, s7, s9\}$ whereas $\{s1\}$ is not (it is not downward closed), as well as $\{s7\}$ (no state from the root) or $\{s1, s2\}$ (two states belonging to the same sequential automaton). The following result easily follows from the definitions:

Proposition 3 *For $A \in F$ and $A' \in \rho_A\ \sigma_A$ the following holds:*
$(\mathcal{C} \in \mathrm{Conf}_A \wedge \mathcal{C} \cap \sigma_{A'} \neq \emptyset) \Rightarrow \mathcal{C} \cap (\mathcal{S}\ A') \in \mathrm{Conf}_{A'}$

The operational semantics of an extended hierarchical automaton will be defined as a Kripke structure, which is a set of states related by a (transition) relation. Usually, the states are called *statuses* and the transition relation is called the *STEP relation*. Each status is composed by a configuration and the current *environment* with which the extended hierarchical automaton is supposed to interact. While in classical statecharts the environment is modeled by a set, in the definition of UML statechart diagrams the particular nature of the environment is not specified (actually it is stated to be a *queue*, but the management policy of such a queue is not defined). We choose *not* to fix any semantics such as a set, or a bag or a FIFO queue etc. for the environment. In the following definition we will then assume that for set X, ΘX denotes the set of all structures of a certain kind (like FIFO queues, or bags, or sets) over X and we shall assume to have basic operations for inserting and removing elements from such structures. In particular $(add\ \mathcal{E}\ e)$ will denote the structure obtained by adding e to environment $\mathcal{E}$. Similarly, $(join\ \mathcal{E}\ \mathcal{E}')$ denotes the environment obtained by merging $\mathcal{E}$ with $\mathcal{E}'$. Moreover, by $(Sel\ \mathcal{E}\ e\ \mathcal{E}')$ we mean that $\mathcal{E}'$ is the environment resulting from selecting e from $\mathcal{E}$, the selection policy depending on the choice for the particular semantics of the environment. Finally, nil is the empty structure and given sequence $r \in X^*$, $(new\ r)$ is the structure containing the elements of r (again, the existence and nature of any relation among the elements of $(new\ r)$ depends on the semantics of the particular structure).

So, for instance, if sets are chosen, then $(add\ \mathcal{E}\ e) = \mathcal{E} \cup \{e\}$, $(join\ \mathcal{E}\ \mathcal{E}') = \mathcal{E} \cup \mathcal{E}'$ and, for $e \in \mathcal{E}$, $(Sel\ \mathcal{E}\ e\ \mathcal{E}') \equiv (\mathcal{E}' = \mathcal{E} \setminus \{e\})$. Details like what is the result of attempting to select an event from an empty environment etc. are left unspecified here since they are part of the semantics of the environment and will be specified when such a semantics is fixed.

Def. 11 (Operational semantics) *The operational semantics of an extended hierarchical automaton H is a Kripke structure $\mathbf{k} = (\mathbf{S}, \mathbf{s^0}, \stackrel{STEP}{\longrightarrow})$ where (i) $\mathbf{S} = \mathrm{Conf}_H \times (\Theta\ E)$ is the set of statuses of $\mathbf{k}$, (ii) $\mathbf{s^0} = (\mathcal{C}_0, \mathcal{E}_0) \in \mathbf{S}$ is the initial status, and (iii) $\stackrel{STEP}{\longrightarrow}$ is the transition relation defined in the sequel.*

A transition of $\mathbf{k}$ is a maximal set of non-conflicting transitions of the sequential automata of H which respect priorities. As in [8], we shall define the $\stackrel{STEP}{\longrightarrow}$ relation by means of a deduction system, and we shall do this both for the case in which the environment can be manipulated from outside the system specified by H (open systems semantics) and for the case in which this is not allowed (closed systems semantics). The rules follow:

Def. 12 (Closed Systems)

$$\frac{(Sel\ \mathcal{E}\ e\ \mathcal{E}'') \quad (1)}{H \uparrow \emptyset :: (\mathcal{C}, \{e\}) \xrightarrow{L} (\mathcal{C}', \mathcal{E}') \quad (2)}$$
$$(\mathcal{C}, \mathcal{E}) \xrightarrow{STEP} (\mathcal{C}', (join\ \mathcal{E}''\ \mathcal{E}'))$$

Def. 13 (Open Systems)

$$\frac{(Sel\ \mathcal{E}\ e\ \mathcal{E}'') \quad (1)}{H \uparrow \emptyset :: (\mathcal{C}, \{e\}) \xrightarrow{L} (\mathcal{C}', \mathcal{E}') \wedge \mathcal{E}' \subseteq \mathcal{E}''' \quad (2)}$$
$$(\mathcal{C}, \mathcal{E}) \xrightarrow{STEP} (\mathcal{C}', (join\ \mathcal{E}''\ \mathcal{E}'''))$$

In the above rules we make use of an auxiliary relation, namely $A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L} (\mathcal{C}', \mathcal{E}')$. The relation $\xrightarrow{L}$ models labeled transitions of the extended hierarchical automaton A, and L is the set containing the transitions of the sequential automata of A which are selected to fire. We shall call $\xrightarrow{L}$ *step* transitions in order to avoid confusion with transitions of sequential automata. P is a set of transitions. It represents a constraint on each of the transitions fired in the step, namely that it must not be the case that there is a transition in P with a higher priority. So, informally, $A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L} (\mathcal{C}', \mathcal{E}')$ should be read as "A, on status $(\mathcal{C}, \mathcal{E})$ can perform L moving to status $(\mathcal{C}', \mathcal{E}')$, when required to perform transitions with priorities not smaller than any in P". Obviously, no restriction is made on the priorities for H, but, as we shall see later, set P will be used to record the transitions a certain automaton can do when considering its sub-automata. More specifically, for sequential automaton A, P will cumulate (the priority information of) all transitions which are enabled in the ancestors of A. In the sequel we shall formalize all the above concepts by means of defining a deduction system for relation $\xrightarrow{L}$. We first need a few auxiliary definitions.

Def. 14 (Enabled Transitions) *For $A \in F$, set of states $\mathcal{C}$ and environment $\mathcal{E}$,*
(i) the set of all the enabled local *transitions of A in $(\mathcal{C}, \mathcal{E})$, $LE_A\ \mathcal{C}\ \mathcal{E}$ is defined as follows*[2]*:*

$$LE_A\ \mathcal{C}\ \mathcal{E} = \{t \in \delta_A \mid \{(SRC\ t)\} \cup (SR\ t) \subseteq \mathcal{C} \wedge (EV\ t)\ in\ \mathcal{E} \wedge (\mathcal{C}, \mathcal{E}) \models (G\ t)\}$$

(ii) the set of all enabled *transitions of A in $(\mathcal{C}, \mathcal{E})$ considered as an extended hierarchical automaton, i.e. including those of descendents of A, $E_A\ \mathcal{C}\ \mathcal{E}$ is defined as follows:*

$$E_A\ \mathcal{C}\ \mathcal{E} = \bigcup_{A' \in (\mathcal{A}\ A)} LE_{A'}\ \mathcal{C}\ \mathcal{E}$$

[2] $(\mathcal{C}, \mathcal{E}) \models g$ means that guard g is true of status $(\mathcal{C}, \mathcal{E})$. Its formalization is immaterial for the purposes of the present paper. Notice anyway that the modular structure of the operational semantics used in the present paper forces g to be defined only on configurations for A. For more global guards slight notational changes are required.

Moreover, $A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L}$ will stand for: there exists $\mathcal{C}'$ and $\mathcal{E}'$ such that $A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L} (\mathcal{C}', \mathcal{E}')$. Finally, for state s and set $S \subseteq \mathcal{S}$ $(\rho\ s)$, such that $s \preceq s''$ for all $s'' \in S$, the *closure* of S, $(\mathbf{c}\ s\ S)$, is defined as the set $\{s' \mid \exists s'' \in S.\ s \preceq s' \preceq s''\}$.

Def. 15 (Progress rule) *If there is a transition of A enabled and the priority of such a transition is "high enough" then the transition fires and a new status is reached accordingly:*

$$\frac{\begin{array}{ll} t \in LE_A\ \mathcal{C}\ \mathcal{E} & (1) \\ \nexists t' \in P \cup E_A\ \mathcal{C}\ \mathcal{E}.\ \pi\ t \sqsubset \pi\ t' & (2) \end{array}}{A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{\{t\}} (\mathbf{c}\ (TGT\ t)\ (TD\ t), new(AC\ t))}$$

The rule essentially says that a (local) transition t of sequential automaton A can fire if it is enabled in the current configuration (1) and there is no higher priority transition in P (so t is "high enough" for P, or "respects" P), or in the set of all the currently enabled transitions of A or of any descendent of A.

Once transition t is taken, a new configuration is entered and proper actions are performed. For instance, in our example, when $\{s3\}$ is the current configuration and $a2$ is offered by the environment, the above rule can be used for firing transition $t5$, which will result in entering configuration $\{s1, s6, s9\}$

Def. 16 (Composition Rule) *This rule establishes how automaton A delegates the execution of transitions to its sub-automata and these transitions are propagated upwards.*

$$\frac{\begin{array}{ll} \{s\} = \mathcal{C} \cap \sigma_A & (1) \\ \rho_A\ s = \{A_1, \ldots, A_n\} \neq \emptyset & (2) \\ \bigwedge_{j=1}^{n} A_j \uparrow P \cup LE_A\ \mathcal{C}\ \mathcal{E} :: (\mathcal{C} \cap (\mathcal{S}\ A_j), \mathcal{E}) \xrightarrow{L_j} (\mathcal{C}_j, \mathcal{E}_j) & (3) \\ \left(\bigcup_{j=1}^{n} L_j = \emptyset\right) \Rightarrow (\forall t \in LE_A\ \mathcal{C}\ \mathcal{E}.\ \exists t' \in P.\ \pi\ t \sqsubset \pi\ t') & (4) \end{array}}{A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{\bigcup_{j=1}^{n} L_j} (\{s\} \cup \bigcup_{j=1}^{n} \mathcal{C}_j, join_{j=1}^{n} \mathcal{E}_j)}$$

First of all notice that the sub-automata are required to perform their step-transitions under the new set $P \cup LE_A\ \mathcal{C}\ \mathcal{E}$ which includes all the enabled local transitions of A (3) so that, in order to be selected, the transitions of such sub-automata must have a priority which is not lower than any of those of the enabled local transitions of A (and A's ancestors, recursively upwards). Notice also that if no transition of the sub-automata can be fired then the rule is applied *only* if also no local transition of A can fire (4), thus propagating the empty set of transitions upwards (see "stuttering" below). The new configuration will still include the current state of A but the possible new states of the sub-automata and related actions are recorded in the new status.

Def. 17 (Stuttering Rule) *If there is no transition of A enabled and with priority "high enough" and moreover no sub-automata exist to which the execution of transitions can*

be delegated, then A has to "stutter":

$$\frac{\begin{array}{ll}\{s\} = \mathcal{C} \cap \sigma_A & (1)\\ \rho_A\ s = \emptyset & (2)\\ \forall t \in LE_A\ \mathcal{C}\ \mathcal{E}.\ \exists t' \in P.\ \pi\ t \sqsubset \pi\ t' & (3)\end{array}}{A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{\emptyset} (\{s\}, nil)}$$

In our example, from status $(\{s1, s6, s8\}, new\ e1)$ automaton $A2$ can only stutter. Moreover, in the above status, automaton $A1$ can fire transition $t6$ and, via the progress rule it can generate a $\xrightarrow{\{t6\}}$ step-transition. Notice also that although transition $t3$ of $A0$ is enabled the progress rule cannot be applied just because of the above step-transition of $A1$ ($\pi\ t3 \sqsubset \pi\ t6$). On the other hand, the composition rule can be applied to $A0$ which will propagate the step of $A1$ and the stuttering of $A2$ to the level of a step transition of $A0$.

Notice that in general the progress rule and the composition rule have not mutually exclusive conditions, so that when both rules are applicable non-determinism arises and results in separate step-transitions from the same status. Another source of non-determinism is of course the presence of different enabled local transitions in the same sequential automaton which are selected by different applications of the progress rule. Finally notice that condition (4) of the composition rule prevents the propagation of stuttering above A when there are transitions of A which can fire.

Properties of the Operational Semantics

In the sequel we present a few results which show that the operational semantics we propose meet the informal requirements stated in the definition of UML [4]. We let $A \in F, \mathcal{C} \in \mathrm{Conf}_A, \mathcal{E} \in (\Theta E), P \in 2^{(\mathcal{T}\ H)}$ be respectively a generic automaton, a configuration, an environment and a set of transitions.

The proofs are carried out by induction either on the length of the derivation for proving $A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L} (\mathcal{C}', \mathcal{E}')$ [9] or on the structure of the subset of F affected by $\mathcal{C}$, $\{A \in F \mid \mathcal{C} \cap \sigma_A \neq \emptyset\}$, [7].

The following proposition guarantees that after firing a transition again a status is reached.

Proposition 4 *For all $L \in 2^{(\mathcal{T}\ H)}, \mathcal{C}', \mathcal{E}'$ the following holds:*
$A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L} (\mathcal{C}', \mathcal{E}') \Rightarrow ((\mathcal{C}' \in \mathrm{Conf}_A) \wedge (\mathcal{E}' \in (\Theta E)))$.

The next lemma expresses a safety property w.r.t. P; it essentially states that only transitions with a "high enough" priority are fired.

Lemma 5 *For all $L \in 2^{(\mathcal{T}\ H)}, t \in L$ the following holds:*
$A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L} \Rightarrow \nexists t' \in P.\ \pi\ t \sqsubset \pi\ t'$

The main result showing that our operational semantics satisfies the requirements informally defined in [4] follows:

Theorem 1 *For all $L \subseteq (\mathcal{T}\ A)$, $A \uparrow P :: (\mathcal{C}, \mathcal{E}) \xrightarrow{L}$ if and only if L is a maximal set, under set inclusion, which satisfies all the following properties: (i) L is conflict-free, i.e. $\forall t, t' \in L.\ \neg t \# t'$); (ii) all transitions in L are enabled in the current status, i.e. $L \subseteq E_A\ \mathcal{C}\ \mathcal{E}$; (iii) there is no transition outside L which is enabled in the current status and which has higher priority than a transition in L, i.e. $\forall t \in L.\ \ \nexists t' \in E_A\ \mathcal{C}\ \mathcal{E}.\ \pi\ t \sqsubset \pi\ t'$; and (iv) all transitions in L respect P, i.e. $\forall t \in L.\ \ \nexists t' \in P.\ \pi\ t \sqsubset \pi\ t'$*

CONCLUSIONS

In this paper we defined a formal operational semantics for a subset of UML Statechart Diagrams. Diagrams of this kind are used to specify behavioural aspects of systems in UML.

We focussed on specifications made up of a single statechart representing them formally by a variant of Extended Hierarchical Automata. We defined a formal semantics for these automata as Kripke structures. The semantics given in this paper differs from that in [8] because of the different priority rules of UML Statechart Diagrams which do not match the hierarchical structure of the automata.

The resulting formal semantics has only three rules; the progress rule, the composition rule and the stuttering rule. Having a small number of rules facilitates the formal proof of properties that show the correctness of the formal semantics with respect to the requirements formulated in the definition of UML [4]. A number of such properties have been formally stated and proven.

Moreover the formal semantics is parametric in aspects which are not (yet) completely defined for UML, like the management of the event queue and the priorities. In particular, parametricity of our semantics definition w.r.t. priorities makes it suitable for describing the behaviour of systems under different priority schemas. For instance, by using $(PWO, \succeq^s, f)$ instead of $(PWO, \preceq^s, f)$ we conjecture that the semantics of classical Statecharts are obtained. All the results on the semantics are preserved since they do not depend on the particular priority schema, provided the notion of conflict and orthogonality satisfy the general constraint here proved as Lemma 3, as it is the case with classical Statecharts.

The subset of UML we considered is rather small. Many features which we did not consider are not of conceptual importance from the semantics definition point of view. Others, like the more "object oriented" ones (e.g. object management, inheritance) are not to be considered as slight extensions of the ideas presented in this paper: they need further research. On the other hand, we consider the semantics presented here as an essential first step towards a more complete model for statecharts.

We also would like to mention the usefulness of our work with respect to finding mistakes and/or incompleteness and/or ambiguities in the informal description of UML statechart diegrams. Examples are the definition of priorities and the definition of the environment/dispatcher.

The definition of a formal semantics of UML Statecharts Diagrams is a necessary first step also towards the use of automatic tools for formal verification and analysis of statechart specifications yielding finite Kripke structures. One of our following steps will be the translation of statecharts into a language that is amenable to formal

verification by means of model checking. In particular the model checking tool SPIN [3] will be considered since it is one of the most efficient tools available. Its specification language, PROMELA, allows the specification of both state variables and communication actions. This feature turns out to be quite convenient when representing statecharts. We are currently experimenting with some examples of PROMELA models for extended hierarchical automata and we are considering the possibility of extending UML OCL (Object Constraint Language) to a simple Linear Time Temporal Logics in the style of that processed by SPIN. Finally, we think that we can use the work presented in this paper as a starting point for the definition of enriched semantics like deterministic-timed, stochastic-timed and probabilistic ones for UML Statecharts Diagrams.

Acknowledgments

We would like to thank E. Mikk from the University of Kiel for the fruitful email exchanges we had with him on the approach to STATEMATE semantics he is developing with his colleagues, which inspired our work on UML Statecharts.

Appendix: Translation of UML statecharts to extended hierarchical automata

The translation maps a UML statechart to an extended hierarchical automaton $H = (F, E, \rho)$ by defining the set of sequential automata F, the composition function ρ and the set of events E. For the sake of simplicity and readability, here we give just an informal sketch of the translation.

Set of sequential automata. Each automaton $A \in F, A = (\sigma_A, s^0_A, \lambda_A, \delta_A)$ is defined as follows.

- States. States of the statechart are uniquely mapped to states of sequential automata.
  - Root automaton H. If the (composite) top state s_0 of the statechart is concurrent then it is mapped to the single (initial) state of a degenerate root automaton H. Otherwise the direct substates of the top state are mapped to states σ_H of the root automaton H.
  - Sub-automata in $\mathcal{A}\ H$. Each non-concurrent composite substate s of the statechart defines the states of a unique sequential automaton A_s, as direct substates of s are mapped to states of σ_{A_s}. Note that regions (direct substates of a concurrent composite state) are not mapped to any state in the extended hierarchical automaton.
- Initial state. The initial state s^0_A of an automaton A is the state that corresponds to the state of the statechart marked by an initial pseudostate.
- Transitions. In order to define the mapping of the transitions, we need the following definitions. A transition of the statechart is characterized by its least common ancestor (LCA) state, which is the lowest level *non-concurrent* state that contains all the source states and target states (here the definition of [4] is slightly modified). The *main source* (*main target*) of a transition is the direct substate of its LCA that contains the sources (targets). According to the above rules, main sources and main targets are always transformed to states of the same automaton.

  Each transition τ in the statechart is mapped to a unique transition t of the extended hierarchical automaton as follows. The source $SRC\ t$ (target $TGT\ t$) of t is the state that corresponds to the main source (main target) of τ. This means that a compound or interlevel transition of the statechart is mapped to a transition of the automaton containing the states corresponding to its main source and main target (this automaton is a sub-automaton of the state representing the LCA). The original source and target states will be included in the label of the transition in the form of source restriction and target determinator, as described below.
- Transition labels. The label of a transition t is of the form $(SR\ t, EV\ t, G\ t, AC\ t, TD\ t)$. $SR\ t$ and $TD\ t$ are generated using the source(s) and target(s) of τ, while the $EV\ t$, $G\ t$ and $AC\ t$ of t are inherited from τ:
  - Source restriction. If the set of states that corresponds to the source(s) of τ is the same as $SRC\ t$, then $SR\ t$ must be empty, otherwise it is such a set of source(s).
  - Target determinator. $TD\ t$ is the normalized set of states that corresponds to the target(s) of τ. Normalizing means computing the maximal set of orthogonal basic states that are substates of the states entered by τ explicitly or by default. In this way, $TD\ t$ explicitly contains all the states which have to be entered when the transition is fired, while some of these states are not explicitly pointed to by τ. The

following is a sketch of a normalization algorithm which visits the states reached by (segments of) τ, starting from its main target:

  - If a basic state is reached then it is added to $TD\ t$ and recursion stops.
  - If a composite state is reached at its boundary then the algorithm is applied recursively to its initial substate, or to the initial substate of each of its regions.
  - If a non-concurrent composite state is reached and its boundary is crossed then the algorithm is applied recursively to its direct substate where the transition continues (note that branch segments are not considered in this paper).
  - If a concurrent composite state is reached and its boundary is crossed then the algorithm is applied recursively to (i) the direct substate(s) of those regions where the transition continues and (ii) the initial substates of the other regions.

- Trigger events. In UML statecharts, each transition (including compound transitions) can have at most one trigger event, since join, fork and branch segments can not have a trigger. Accordingly, $EV\ t$ is exactly the trigger event of τ.
- Guards. Since fork and joint segments have no guards, each transition may have a single guard (note that branch segments are not considered in this paper). Accordingly, $G\ t$ is exactly the guard of τ.
- Actions. $AC\ t$ is exactly the sequence of actions of τ.

Composition function. ρ is determined by the substate relationships of composite states. If a composite state s is non-concurrent and it is not a region then its direct substates form the states of A_s, a sub-automaton of s, where $\{A_s\} = (\rho\ s)$. If a composite state s is concurrent then every one of its regions forms a sub-automaton of s, in such a way that this automaton contains the direct substates of the region.

Set of events. E is defined as the union of two (not necessarily distinct) sets: the set of events used in the statechart as triggers of the transitions and the set of events generated by actions. In open systems, the set of events generated by the environment is also included.

References

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

[2] D. Harel. The STATEMATE semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4):293–333.

[3] G. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

[4] Rational Software * Microsoft * Hewlett-Packard * Oracle * Sterling Software * MCI Systemhouse * Unisys * ICON Computing * IntelliCorp * i Logix * IBM * ObjecTime * Platinum Technology * Ptech * Taskon * Reich Technologies * Softeam. *UML Semantics, version 1.1*, 1997.

[5] Rational Software * Microsoft * Hewlett-Packard * Oracle * Sterling Software * MCI Systemhouse * Unisys * ICON Computing * IntelliCorp * i Logix * IBM * ObjecTime * Platinum Technology * Ptech * Taskon * Reich Technologies * Softeam. *UML Notation Guide, version 1.1*, 1997.

[6] D. Latella, M. Massink, and I. Majzik. A Simplified Formal Semantics for a Subset of UML Statechart Diagrams. Technical Report HIDE/T1.2/PDCC/5/v1, ESPRIT Project n. 27439 - High-Level Integrated Design Environemnt for Dependability HIDE, 1998. Available in the HIDE Project Public Repository (https://asterix.mit.bme.hu:998/).

[7] Z. Manna, S. Ness, and J. Vuillemin. Inductive methods for proving properties of programs. *Communications of the ACM*, 16(8), 1973.

[8] E. Mikk, Y. Lakhnech, and M. Siegel. Hierarchical automata as model for statecharts. In R. Shyamasundar and K. Euda, editors, *Third Asian Computing Science Conference. Advances in Computing Sience - ASIAN'97*, volume 1345 of *Lecture Notes in Computer Science*, pages 181–196. Springer-Verlag, 1997.

[9] R. Milner. *Communication and Concurrency*. Series in Computer Science. Prentice Hall, 1989.