

On Mobility Extensions of UML Statecharts. A Pragmatic Approach*

Diego Latella and Mieke Massink

CNR/ISTI – A. Faedo, Via Moruzzi 1, I56124 Pisa, Italy
{d.latella,m.massink}@cnuce.cnr.it

Abstract. In this paper an extension of a behavioural subset of UML Statecharts for modeling mobility issues is proposed. In this extension we relax the unique association between each Statechart - in a collection of Statecharts modeling a system - and its input-queue and we allow the use of (queue) name variables in communication actions. The resulting communication paradigm is much more flexible than the standard asymmetric one and is well suited for the modelling of mobility-oriented as well as fault tolerant systems.

1 Introduction

The Unified Modelling Language (UML) is a graphical modelling language for object-oriented software and systems [12]¹. It has been specifically designed for visualizing, specifying, constructing and documenting several aspects of - or views on - systems. In this paper we concentrate on a behavioural subset of UML Statecharts (UMLSCs) and in particular on a simple but powerful extension of this notation in order to deal with a notion of mobility which can be modeled by the use of a dynamic communication structure and which is sometimes referred to as *mobile computing* (as opposed to *mobile computation*) [1]. In [4] μ Charts have been introduced together with their formal semantics². Briefly, a μ Chart models the behaviour of a system and is a *collection* of UMLSCs, each UMLSC being uniquely associated with its input queue. The computational model of μ Charts is an interleaving one with an asynchronous/asymmetric/static pattern of communication. The semantics of a μ Chart is a Labelled Transition System (LTS) where each state corresponds to the tuple of statuses of the component UMLSCs, each status being composed by the current configuration and the current input queue of the component UMLSC. A transition in the LTS models a step-transition of a component UMLSC. Each step transition corresponds to the selection of an event from the input queue of the component and to the parallel firing of a maximal set of non-conflicting transitions of such component,

* This work has been carried out in the context of Project EU-IST IST-2001-32747 Architectures for Mobility (AGILE),

<http://www.pst.informatik.uni-muenchen.de/projekte/agile/>.

¹ Although we base our work on UML 1.3, the main features of the notation of interest for our work did not change in later versions.

² Note that the name ' μ Charts' is used also in [13], but with a different meaning.

which are enabled in its current configuration by the selected event and which do not violate transition priority constraints. The firing of a transition implies also the execution of the output actions associated with such a transition. An output action consists of an *output event* to be sent and the specification of a *destination* component (queue) to which it must be delivered. Its execution consists in delivering the event to the destination queue. Thus the pattern of communication is *asynchronous*, via the input queues, and *asymmetric* because while the sender component specifies to which destination component an event must be addressed, a destination component cannot choose from which sender component to receive input events; it simply *has* to receive any event which has been delivered to its input queue, possibly producing no reaction to such trigger event. This is quite a common situation in the realm of object-oriented notations. There are important features of mobile systems that cannot be expressed directly using only an asymmetric style of communication. In fact there are situations in which we want to make a receiver be able to get input events from *more than one* queue, *explicitly* choosing *when* to receive events from *which* queue. An example of the need of such pattern of communication is, a. o., the Hand-over protocol for mobile telephones, which we shall deal with in the present paper. Moreover, it is well-known [2] that patterns of communication which allow the explicit and dynamic choice of the input queue/entity are essential for the development of fault-tolerant systems since they contribute to fulfilling well established entity isolation principles of error confinement much better than asymmetric patterns. Thus, the extension we propose in this paper consists in letting the *trigger event* specification of transition labels be equipped with the explicit reference to the queue from which the event should be taken. Moreover, such a reference can also be a queue name variable, thus allowing more dynamicity in the choice of the queue(s) from which events are to be received by a UMLSC. Queue names can thus be communicated around and assigned to variables in a way which resembles the π -calculus [11], although in the more imperative-like framework of UMLSCs.

We are not aware of any other work on extensions of UMLSCs formal semantics with notions of mobile computing, like dynamic addressing, and name-passing. For what concerns formal semantics of UMLSCs numerous contributions can be found in the literature. For a discussion on such contributions and a comparison with our overall approach to UMLSCs semantics we refer the interested reader to [4]. More recently, an SOS approach to (single) UMLSCs formal semantics has been proposed in [15], which is partially based on [7, 10] and assumes the standard UML single input queue per statechart paradigm. Some issues related to communication concepts for (single) Harel statecharts are investigated in [14, 13]. The main focus there is the restriction of classical statechart broadcast by means of explicit feedback interfaces rather than the issues of input queue selection and dynamic addressing in UML statecharts, which we are interested in, in the present paper.

The paper is organized as follows: in Sect. 2 μ Charts are briefly recalled and some basic definitions are given. Sect. 3 describes the extension we propose,

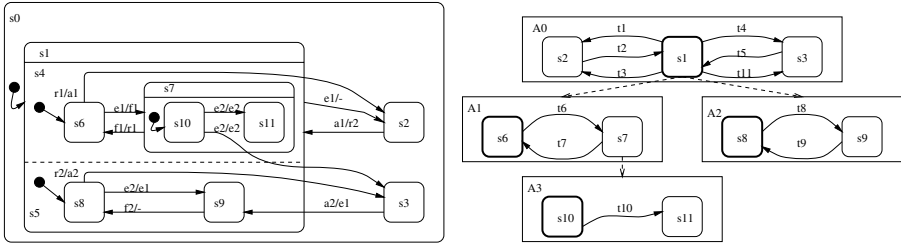


Fig. 1. A UMLSC and its HA.

Table 1. Transition Labels for the HA of Fig. 1.

t	$t1$	$t2$	$t3$	$t4$	$t5$	$t6$	$t7$	$t8$	$t9$	$t10$	$t11$
$SR\ t$	$\{s6\}$	\emptyset	\emptyset	$\{s8\}$	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset	$\{s10\}$
$EV\ t$	$r1$	$a1$	$e1$	$r2$	$a2$	$e1$	$f1$	$e2$	$f2$	$e2$	$e2$
$AC\ t$	$a1$	$r2$	ϵ	$a2$	$e1$	$f1$	$r1$	$e1$	ϵ	$e2$	$e2$
$TD\ t$	\emptyset	$\{s6, s8\}$	\emptyset	\emptyset	$\{s6, s9\}$	$\{s10\}$	\emptyset	\emptyset	\emptyset	\emptyset	\emptyset

including its formal semantics definition. The Hand-over protocol specification example is given in Sect. 4. Finally in Sect. 5 some conclusions are drawn and directions for further work are sketched. Detailed proofs relevant to the present paper can be found in [9].

2 μ Charts

In this section the basic definitions related to μ Charts are briefly recalled. They are treated in depth in [4] where the interested reader is referred to. We use Hierarchical Automata (HAs) [10] as the abstract syntax for UMLSCs. HAs are composed of simple sequential automata related by a *refinement function*. In [7] an algorithm for mapping a UMLSC to a HA is given; the reader interested in its technical details is referred to the above mentioned paper. Here we just recall the main ingredients of this mapping, by means of a simple example. Consider the UMLSC of Fig.1 (left). Its HA is shown on the right side of the figure. Roughly speaking, each OR-state of the UMLSC is mapped into a sequential automaton of the HA while basic and AND-states are mapped into states of the sequential automaton corresponding to the OR-state immediately containing them. Moreover, a refinement function maps each state in the HA corresponding to an AND-state into the set of the sequential automata corresponding to its component OR-states. In our example (Fig.1, right), OR-states $s0, s4, s5$ and $s7$ are mapped to sequential automata $A0, A1, A2$ and $A3$, while state $s1$ of $A0$, corresponding to AND-state $s1$ of our UMLSC, is refined into $\{A1, A2\}$. Non-interlevel transitions are represented in the obvious way: for instance transition $t8$ of the HA represents the transition from state $s8$ to state $s9$ of the UMLSC. The labels of transitions are collected in Table 1; for example the *trigger event* of $t8$, namely $EV\ t8$, is $e2$ while its associated *output event*, namely $AC\ t8$ is $e1$. An interlevel transition is represented as a transition t departing from (the HA state corresponding to) its highest source and pointing to (the HA state

corresponding to) its highest target. The set of the other sources, resp., targets, are recorded in the *source restriction - SR* t , resp. *target determinator TD* t , of t . So, for instance, $SR\ t1 = \{s6\}$ means that a necessary condition for $t1$ to be enabled is that the current state configuration contains not only $s1$ (the source of $t1$), but *also* $s6$. Similarly, when firing $t2$ the new state configuration will contain $s6$ and $s8$, besides $s1$. Finally, each transition has a guard G t , not shown in this example. The structure of transition labels will be properly accommodated later on in this paper in order to support the mobility extensions. Transitions originating from the same state are said to be in *conflict*. The notion of *conflict* between transitions needs to be extended in order to deal with state hierarchy. When transitions t and t' are in conflict we write $t\#\#t'$. The complete formal definition of conflict for HAs can be found in [7, 4] where also the notion of *priority* for (conflicting) transitions is defined. Intuitively transitions coming from deeper states have higher priority. For the purposes of the present paper it is sufficient to say that priorities form a partial order. We let πt denote the priority of transition t and $\pi t \sqsubseteq \pi t'$ mean that t has lower priority than (the same priority as) t' . In the sequel we will be concerned only with HAs.

A μ Chart is a collection of UML Statecharts (actually HAs) communicating via input queues. We consider a restricted subset of UML Statecharts, which, nevertheless includes all the interesting conceptual issues related to concurrency in the dynamic behaviour, like sequentialisation, non-determinism and parallelism. We call such a subset a “Behavioural subset of UML Statecharts”, UMLSCs in short. More specifically, we do not consider history, action and activity states; we restrict events to signal ones without parameters (actually we do not interpret events at all); time and change events, object creation and destruction events, and deferred events are not considered as are branch transitions; for the sake of simplicity, given that in the present paper our main focus is on the manipulation of queues for achieving dynamic communication structures, we restrict data values to a single type, namely *queue names*. We also abstract from entry and exit actions of states. The interested reader can find a complete discussion on the above choices together with their motivations in [4].

2.1 Basic Definitions

The first notion we need to define is that of (sequential) automaton³.

³ In the following we shall freely use a functional-like notation in our definitions where: (i) currying will be used in function application, i.e. $f\ a_1\ a_2\ \dots\ a_n$ will be used instead of $f(a_1, a_2, \dots, a_n)$ and function application will be considered left-associative; (ii) for function $f : X \rightarrow Y$ and $Z \subseteq X$, $f\ Z = \{y \in Y \mid \exists x \in Z. y = fx\}$, *rng* f denotes the *range* of f and $f|_Z$ is the restriction of f to Z . (iii) by $\exists_1 x. P\ x$ we mean “there exists a unique x such that $P\ x$ ”. Finally, for set X , we let X^* denote the set of finite sequences over D . The empty sequence will be denoted by ϵ and the concatenation of sequence x with sequence y will be indicated by xy . For sequences x , y and z we let predicate $\text{mrg}\ x\ y\ z$ hold iff z is a non-deterministic merge of x and y , that is z is a permutation of xy such that the occurrence order in x (resp. y) of the elements of x (resp. y) is preserved in z ; its extension $\text{mrg}_{j=1}^n\ x_j\ z$ to n sequences is defined in the obvious way.

Definition 1 (Sequential Automata). A sequential automaton A is a 4-tuple $(\sigma_A, s_A^0, \lambda_A, \delta_A)$ where σ_A is a finite set of states with $s_A^0 \in \sigma_A$ 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.

The labels in λ_A have a particular structure as we briefly mentioned above and we shall discuss later in more detail. Moreover, we assume that all transitions are uniquely identifiable. This can be easily achieved by just assigning them arbitrary unique names, as we shall do throughout this paper. For sequential automaton A let functions $SRC, TGT : \delta_A \rightarrow \sigma_A$ be defined as $SRC(s, l, s') = s$ and $TGT(s, l, s') = s'$. Let \mathcal{N} be a set of (queue) names. HAs are defined as follows:

Definition 2 (Hierarchical Automata). A HA 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, with $E \subseteq \mathcal{N}$; the refinement function $\rho : \bigcup_{A \in F} \sigma_A \rightarrow 2^F$ imposes a tree structure to F , i.e. (i) there exists a unique root automaton $A_{root} \in F$ such that $A_{root} \notin \bigcup \text{rng } \rho$, (ii) every non-root automaton has exactly one ancestor state: $\bigcup \text{rng } \rho = F \setminus \{A_{root}\}$ and $\forall A \in F \setminus \{A_{root}\}. \exists! s \in \bigcup_{A' \in F \setminus \{A\}} \sigma_{A'}. A \in (\rho s)$ 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. From the above definition the reader can see that the only type of events we deal with are queue names, as mentioned above. Every sequential automaton $A \in F$ characterizes a HA in its turn: intuitively, such a HA is composed by all those sequential automata which lay below A , including A itself, and has a refinement function ρ_A which is a restriction of ρ :

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

The definition of sub-hierarchical automaton follows:

Definition 4 (Sub-Hierarchical Automata). For $A \in F, (F_A, E, \rho_A)$, where $F_A \triangleq (\mathcal{A} A)$, and $\rho_A \triangleq \rho|_{(\mathcal{S} A)}$, is the HA characterized by A .

In the sequel for $A \in F$ we shall refer to A both as a sequential automaton and as the sub-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 HAs. μ Charts are defined as follows:

Definition 5 (μ Chart). A μ Chart is a tuple (E, H_1, \dots, H_k) where (i) $E \subseteq \mathcal{N}$, (ii) $H_j = (F_j, E, \rho_j)$ is a HA for $j = 1 \dots k$ and (iii) $\mathcal{S} H_j \cap \mathcal{S} H_i = \emptyset$ for $i \neq j$.

The complete formal semantics for μ Charts can be found in [4]. A μ Chart is mapped into a LTS. Each state of such LTS is a tuple of configuration-queue

pairs; each pair records the current configuration and the current input queue of a distinct HA in the μ Chart. The transition relation of the LTS is defined by means of a derivation system composed by four rules. One of them, the *top rule*, defines the transition relation and uses in turn an auxiliary relation defined by the three remaining rules, the so called *core semantics*. In the next sections we shall deal with the semantics of the *extension* of μ Charts we propose. Before proceeding with the semantics definitions we need a few more concepts:

Definition 6 (Configurations). *A configuration of HA $H = (F, E, \rho)$ is a set $\mathcal{C} \subseteq (\mathcal{S} H)$ such that (i) $\exists_1 s \in \sigma_{A_{root}}. s \in \mathcal{C}$ and (ii) $\forall s, A. s \in \mathcal{C} \wedge A \in \rho \ s \Rightarrow \exists_1 s' \in \sigma_A. s' \in \mathcal{C}$*

A *configuration* denotes a global state of a HA, composed of local states of component sequential automata. For $A \in F$ the set of all configurations of A is denoted by Conf_A . In the extension we propose we will deal with queue name variables. Thus we assume a universe \mathcal{V} of such variables, with $\mathcal{N} \cap \mathcal{V} = \emptyset$. We also need communication packets; a packet is a pair (*destination, message-body*). The set \mathcal{P} of *packets* is defined as $\mathcal{P} \triangleq \mathcal{N} \times \mathcal{N}$, that is message bodies can be only queue names. Due to the presence of variables we need also *packet terms* and stores. The set \mathcal{Pt} of packet terms is defined as $\mathcal{Pt} \triangleq (\mathcal{N} \cup \mathcal{V}) \times (\mathcal{N} \cup \mathcal{V})$. Stores are defined as follows:

Definition 7 (Stores). *A store β is a partial function $\beta : \mathcal{V} \rightarrow \mathcal{N}$. As usual $\beta v = \perp$ means that v is not bound by β to any value, namely β is undefined on v . We let $\perp\!\!\!\perp$ be the function such that $\perp\!\!\!\perp v = \perp$ for all $v \in \mathcal{V}$. For store β we let $\hat{\beta}$ denote its extension to names and packet terms, in the usual way: $\hat{\beta} q \triangleq q$, if $q \in \mathcal{N}$, $\hat{\beta} v \triangleq \beta v$, if $v \in \mathcal{V}$, and $\hat{\beta} (d, b) \triangleq (\hat{\beta} d, \hat{\beta} b)$. For $v \in \mathcal{V}$ and $q \in \mathcal{N}$ the unit store $[v \mapsto q]$ is the function such that $[v \mapsto q]v' \triangleq q$, if $v' = v$, and $[v \mapsto q]v' \triangleq \perp$, if $v' \neq v$. Finally, for stores β_1 and β_2 we let store $\beta_1 \triangleleft \beta_2$ be the function such that $(\beta_1 \triangleleft \beta_2) v \triangleq \beta_2 v$, if $\beta_2 v \neq \perp$ and $(\beta_1 \triangleleft \beta_2) v \triangleq \beta_1 v$, if $\beta_2 v = \perp$.*

In the following, we shall often consider stores as sets of pairs and compose them using set-union, when the domains of the component functions are mutually disjoint. While in classical statecharts the environment is modelled by a set, in the official definition of UMLSCs the particular nature of the environment is not specified. Actually it is stated to be a *queue*, the *input queue*, but the management policy of such a queue is not defined. We choose *not* to fix any particular semantics such as a set, or a multi-set or a FIFO queue etc., but to model the input queue in a policy-independent way, freely using a notion of abstract data types. In the following we assume that for set D , Θ_D denotes the set of all structures of a certain kind (like FIFO queues, or multi-sets, or sets) over D and we assume to have basic operations for inserting and removing elements from such structures. Among such operations, the predicate $(\text{Sel } \mathcal{D} d \mathcal{D}')$ which states that \mathcal{D}' is the structure resulting from selecting d from \mathcal{D} , is of particular importance in the context of the present paper. Of course, the selection policy

depends on the choice for the particular semantics. In the present paper we assume that if \mathcal{D} is the empty structure, nil then $(Sel \ \mathcal{D} \ d \ \mathcal{D}')$ is false for all d and \mathcal{D}' . In the sequel we shall often speak of the *input queue* or simply *queue* meaning by that a structure in Θ_D , for proper D , and abstracting from its particular semantics.

3 μ Charts with Explicit Dynamic Channels

In this section we describe our mobility extension of μ Charts. As before, a system is modeled by a fixed collection of UMLSCs (actually HAs), but now each HA can be associated to several *input-queues*. The association is specified by explicit reference, in any transition of the HA, to the input queue from which the trigger event of that transition is to be selected. The association is *dynamic* since, besides queue names, uniquely associated to distinct queues, queue name *variables* can be used as well. Consequently we have to re-define the labels of the transitions of HAs. Let $H = (F, E, \rho)$ be a HA of a μ Chart S . The label l of transition $t = (s, l, s') \in \delta_A$, for $A \in F$, is the tuple $(SR \ t, IQ \ t, EV \ t, G \ t, DQ \ t, AC \ t, TD \ t)$. The meaning of $SR \ t, G \ t$ and $TD \ t$ is the same as briefly discussed in Sect. 2. The specification of the queue from which the *trigger event* $EV \ t$ should be selected is given by the *input-queue* component of the label, $IQ \ t$. In the present paper, for the sake of simplicity, the only kind of actions a HA can perform when firing a transition is the sending of an event to a queue. Consequently, on the output side, the specification of the *destination queue* $DQ \ t$ is added to that of the *output event* $AC \ t$. At the concrete syntax level, the label of a transition t of a UMLSC will have the form $q?e/q'!e'$ where $IQ \ t = q$, $EV \ t = e$, $DQ \ t = q'$ and $AC \ t = e'$ ⁴. As we said above the only values we are dealing with are queue-names⁵ and we allow the use of variables in order to express dynamic addresses. Consequently the trigger event, the input-queue, the output event and the destination queue can be queue-names or queue-name variables, i.e. $EV \ t, IQ \ t, DQ \ t, AC \ t \in E \cup \mathcal{V}$. Furthermore, the above assumptions imply that there is a pool of “shared” queues through which the HAs communicate. We call such a pool a *multi-queue* and a collection of HAs communicating via a multi-queue is called a $\delta\mu$ Chart. Multi-queues are an extension of input-queues. We need to redefine the selection relation and multi-queue extension. We do this informally as follows: $Sel \ \mathcal{E} \ q \ e \ \mathcal{E}'$ holds iff e is the element selected from queue (named) q of multi-queue \mathcal{E} and \mathcal{E}' is the multi-queue resulting from deleting e from queue (named) q of \mathcal{E} . We let $\mathcal{E}[(q, e)]$ denote the multi-queue equal to \mathcal{E} except that on queue (named) q of \mathcal{E} element e is inserted, where $(q, e) \in \mathcal{P}$; for $U \in \mathcal{P}^*$, $\mathcal{E}[U]$ is defined in the obvious way: $\mathcal{E}[(q_1, e_1), (q_2, e_2), \dots, (q_n, e_n)] \triangleq (\mathcal{E}[(q_1, e_1)])[(q_2, e_2), \dots, (q_n, e_n)]$ where $\mathcal{E}[\epsilon] \triangleq \mathcal{E}$.

⁴ Notice that in this paper we use a syntax for event sending (namely exclamation mark) which is slightly different from the standard syntax for method calling (namely a dot). This is only for symmetry with our notation for input (namely question mark) and for our deliberate focusing on semantical more than syntactical issues.

⁵ This is a similar situation as in the π – calculus [11], although in a completely different context.

The Operational Semantics of $\delta\mu\text{Chart}$ $S = (E, H_1, \dots, H_k)$ is a transition system. Each global state is a tuple $(\mathcal{E}, Loc_1, \dots, Loc_k)$. \mathcal{E} is the current value of multi-queue and Loc_j is the current status of HA H_j . The status Loc_j of a component HA H_j is a pair (\mathcal{C}_j, β_j) where \mathcal{C}_j is the current configuration of H_j and β_j is the current store of H_j . Each transition corresponds to a step of one component HA H_j . We recall here that in μCharts , being each HA uniquely associated to a distinct queue, the hypothetical scheduler associated to the semantics of state machines in [12] is only left with the job of (i) choosing a HA to execute a step and (ii) selecting an event *in the input queue of the selected HA* to feed into its state machine in order to perform such a step. Notice that the scheduler is somehow “blind” in this job w.r.t. the event: it chooses and *de-queues* an event regardless of whether such an event will be actually used by the state machine, i.e. there are transitions which are enabled by the event. If this is not the case, the state machine stutters and the event is lost (or, at most, deferred⁶). In the case of $\delta\mu\text{Charts}$ there is no longer a single input queue associated to each HA, but the multi-queue. Thus, in $\delta\mu\text{Charts}$ the scheduler must also select the *particular input queue* from which the event is to be selected. We shall deal with input queue selection in Sect.3.2, where we define the top rule(s) of the derivation system for the transition relation. Since such a derivation system exploits some properties of the core semantics on which it is based, we prefer to first define, in Sect. 3.1, the core semantics which is obviously parametric w.r.t. the selected queue.

3.1 Core Semantics

The core semantics is given in Fig.2 and has the same structure as that for μCharts . It defines the relation $A \uparrow P :: (\mathcal{C}, \beta) \xrightarrow{(q,e)/\mathcal{U}}_L (\mathcal{C}', \beta')$ which models the step-transitions of HA A , and L is the set containing the transitions of A which are fired. In such a relation 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, $A \uparrow P :: (\mathcal{C}, \beta) \xrightarrow{(q,e)/\mathcal{U}}_L (\mathcal{C}', \beta')$ should be read as “ A , on configuration and store (\mathcal{C}, β) , with input event e from queue q can fire the transitions in the set L moving to configuration and store (\mathcal{C}', β') , producing output \mathcal{U} , when required to fire transitions with priorities not smaller than that of any transition in P ”. 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 accumulate all transitions which are enabled in the ancestors of A . The Core Semantics definition uses functions LE_A and E_A . For HA A , $\text{LE}_A \mathcal{C} \beta q e$ is the set $\{t \in \delta_A \mid \{(SRC\ t)\} \cup (SR\ t) \subseteq \mathcal{C} \wedge \hat{\beta}(IQ\ t) = q \wedge \text{MATCH } e (EV\ t) \wedge (\mathcal{C}, \beta, e) \models (G\ t)\}$ i.e. the set of those transitions in δ_A , i.e. *local* to the root of A , which are enabled in the current configuration \mathcal{C} , store β with input event e from input queue q . Function MATCH is defined as follows: $\text{MATCH } e\ x \stackrel{\Delta}{=} (x \in \mathcal{V} \vee x = e)$.

⁶ We do not deal with deferred events in our current work.

Progress Rule

$$\frac{t \in \mathbf{LE}_A \mathcal{C} \beta q e \\ \beta' = \text{if } (EV t) \in \mathcal{V} \text{ then } [(EV t) \mapsto e] \text{ else } \perp \\ \exists t' \in P \cup \mathbf{E}_A \mathcal{C} \beta q e. \pi t \sqsubset \pi t'}{A \uparrow P :: (\mathcal{C}, \beta) \xrightarrow{(q,e)/(DQ t, AC t)}_{\{t\}} (DST t, \beta')}$$

Stuttering Rule

$$\frac{\{s\} = \mathcal{C} \cap \sigma_A \\ \rho_A s = \emptyset \\ \forall t \in \mathbf{LE}_A \mathcal{C} \beta q e. \exists t' \in P. \pi t \sqsubset \pi t'}{A \uparrow P :: (\mathcal{C}, \beta) \xrightarrow{(q,e)}_{\emptyset}^{\epsilon} (\{s\}, \perp)}$$

Composition Rule

$$\frac{\{s\} = \mathcal{C} \cap \sigma_A \\ \rho_A s = \{A_1, \dots, A_n\} \neq \emptyset \\ \left(\bigwedge_{j=1}^n A_j \uparrow P \cup \mathbf{LE}_A \mathcal{C} \beta q e :: (\mathcal{C}, \beta) \xrightarrow{(q,e)/\mathcal{U}_j} L_j(\mathcal{C}_j, \beta_j) \right) \wedge \text{mrg}_{j=1}^n \mathcal{U}_j \mathcal{U} \\ \left(\bigcup_{j=1}^n L_j = \emptyset \right) \Rightarrow (\forall t \in \mathbf{LE}_A \mathcal{C} \beta q e. \exists t' \in P. \pi t \sqsubset \pi t')}{A \uparrow P :: (\mathcal{C}, \beta) \xrightarrow{(q,e)/\mathcal{U}} \bigcup_{j=1}^n L_j (\{s\} \cup \bigcup_{j=1}^n \mathcal{C}_j, \bigcup_{j=1}^n \beta_j)}$$

Fig. 2. Core operational semantics rules for $\delta\mu$ Charts.

We skip the details of guard evaluation in this paper for lack of space. Function \mathbf{E}_A extends \mathbf{LE}_A in order to cover *all* the transitions of A including those of sub-automata of A , i.e. $\mathbf{E}_A \mathcal{C} \beta q e \triangleq \bigcup_{A' \in (A \ A)} \mathbf{LE}_{A'} \mathcal{C} \beta q e$. In the Core Semantics, the Progress Rule establishes that 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. The store generated contains *only* the information related to the possible binding of $EV t$, when the latter is a variable. The global store of the HA which A belongs to will be extended properly by the top-level rules (see below). Notice that the destination ($DQ t$) and the message body ($AC t$) are dealt with in a symbolic way at the Core Semantics level. They will be evaluated by the Global Progress Rule. The reason why the variable evaluation cannot be performed by the core semantics should be clear: a variable used in the output action of a transition t of a parallel component, say A of a HA might be bound by *another* parallel component of the same HA. So the correct store to be used for ($DQ t, AC t$) pairs is *not* available when applying the Progress Rule to A ⁷. For transition t , $DST t$ is defined as the set $\{s \mid \exists s' \in (TD t). (TGT t) \preceq s \preceq s'\}$. Intuitively $DST t$ comprises all states which are below ($TGT t$) in the state-hierarchy down to those in ($TD t$). The state preorder \preceq is formally defined e.g. in [4]. The Composition Rule stipulates how automaton A delegates the execution of transitions to its sub-automata and these transitions are propagated upwards. Notice that for all $v, i, j, \beta_i v \neq \perp$ and $\beta_j v \neq \perp$ implies $\beta_i v = \beta_j v = e$. Finally, if there is no transition of A enabled with “high enough” priority and moreover no sub-automata exist to which the execution of transitions can be delegated, then A has to “stutter”, as enforced by the Stuttering Rule. The following theorem links our semantics to the general requirements set by the official semantics of UML:

⁷ In [9] a “late” binding semantics is also provided where the destination and message body are evaluated using the *current* store.

Theorem 1. *Given HA $H = (F, E, \rho)$ element of a $\delta\mu$ Chart, for all $A \in F, e \in E, L, \mathcal{C}, \beta, q, \mathcal{U}$ the following holds: $A \uparrow P :: (\mathcal{C}, \beta) \xrightarrow{(q,e)/\mathcal{U}}_L (\mathcal{C}', \beta')$ for some \mathcal{C}', β' iff 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. $L \subseteq E_A \mathcal{C} \beta q e$; (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. $\forall t \in L. \nexists t' \in E_A \mathcal{C} \beta q e. \pi t \sqsubset \pi t'$; and (iv) all transitions in L respect P , i.e. $\forall t \in L. \nexists t' \in P. \pi t \sqsubset \pi t'$.*

Proof. The proof can be carried out in a similar way as for the main theorem of [4], by structural induction for the direct implication and by derivation induction for the reverse implication.

3.2 Input Queue Selection

The selection from the multi-queue is tightly connected to the possibility of generating unwanted extra-stuttering. For generating a step of a HA H starting from configuration \mathcal{C} and multi-queue \mathcal{E} , we consider only those queues in \mathcal{E} which are *relevant* in \mathcal{C} ; q is relevant in \mathcal{C} if there is a transition the source of which is contained in \mathcal{C} and the label of which uses q as input queue. No stuttering of a HA H in \mathcal{C} and multi-queue \mathcal{E} should be allowed which is caused by the selection of a (relevant) queue q of \mathcal{E} and an event dequeued from it for which no transition is enabled *while* there is another (relevant) queue q' of \mathcal{E} and/or another event such that a transition could be enabled. Thus we allow stuttering *only* on relevant queues and *only* if there is no transition enabled. Notice that a stuttering step modifies the multi-queue in a way which depends on the selected queue. The set $\text{LR}_A \mathcal{C} \beta$ of *local relevant* and the set $\text{R}_A \mathcal{C} \beta$ of the *relevant* queues of $A \in H = (F, E, \rho)$ are defined respectively as $\{q \in E \mid \exists t \in \delta_A. (\text{SRC } t) \in \mathcal{C} \wedge \hat{\beta}(\text{IQ } t) = q\}$ and $\bigcup_{A' \in (A \setminus A)} \text{LR}_{A'} \mathcal{C} \beta$. For $A \in F$ of HA $H = (F, E, \rho)$, $\mathcal{C} \in \text{Conf}_A$, store β , multiqueue \mathcal{E} on E , the set $\text{PE}_A \mathcal{C} \beta \mathcal{E}$ of *Potentially Enabled Transitions* of A , on configuration \mathcal{C} , store β and multi-queue \mathcal{E} is the set $\{t \in \mathcal{T} A \mid \exists q \in \text{R}_A \mathcal{C} \beta, e \in E. \text{Sel } \mathcal{E} q e \mathcal{E}' \wedge t \in E_A \mathcal{C} \beta q e\}$. Obviously, the fact that a transition $t \in \text{PE}_A \mathcal{C} \beta \mathcal{E}$ will actually be enabled depends on the choice of the particular relevant queue q and the selection of the particular event e from q . The following lemmas relate stuttering with the set of potentially enabled transitions and with the resulting configurations and stores. They will be useful for a better understanding of the top-level rules of the definition of the transition relation. Lemma 1 states that if there is no potentially enabled transition - in a given configuration, store and multi-queue - then every step from that configuration and store involving any relevant queue and any selected event is a stuttering step. Vice-versa, if from a given configuration, store and multi-queue only stuttering steps are possible, whatever choice is done for the queue and the input event, then there are no potentially enabled transitions. The lemma easily follows from Lemma 3 in [9]. Lemma 2 guarantees that stuttering does not change the store and the configuration. Its proof is similar that of Lemma A.1 (ii) in [8].

Lemma 1.

For all $H = (F, E, \rho), \mathcal{C} \in \text{Conf}_H$, store β, \mathcal{E} multiqueue on E the following holds:
 (i) for all $q, e \in E, \mathcal{E}'$ multiqueue on E : if $\text{PE}_H \mathcal{C} \beta \mathcal{E} = \emptyset$ and $q \in \mathbf{R}_H \mathcal{C} \beta$, and $\text{Sel } \mathcal{E} q e \in \mathcal{E}'$ and $H \uparrow \emptyset :: (\mathcal{C}, \beta) \xrightarrow{(q,e)/\mathcal{U}}_L (\mathcal{C}', \beta')$ for some \mathcal{C}', β', L , then $L = \emptyset$;
 (ii) if for all $q, e \in E, \mathcal{E}'$ multiqueue on E such that $\text{Sel } \mathcal{E} q e \in \mathcal{E}', \mathcal{C}', \beta'$ we have $H \uparrow \emptyset :: (\mathcal{C}, \beta) \xrightarrow{(q,e)/\mathcal{U}}_{\emptyset} (\mathcal{C}', \beta')$ then we also have $\text{PE}_A \mathcal{C} \beta \mathcal{E} = \emptyset$

Lemma 2.

For all $H = (F, E, \rho), \mathcal{C} \in \text{Conf}_H$, store $\beta, q, e \in E$ if $H \uparrow \emptyset :: (\mathcal{C}, \beta) \xrightarrow{(q,e)/\mathcal{U}}_{\emptyset} (\mathcal{C}', \beta')$ then $\mathcal{C}' = \mathcal{C}$ and $\beta' = \perp\!\!\!\perp$.

Global Progress rule

$$q \in \mathbf{R}_{H_j} \mathcal{C}_j \beta_j$$

$$\text{Sel } \mathcal{E} q e \in \mathcal{E}'$$

$$H_j \uparrow \emptyset :: (\mathcal{C}_j, \beta_j) \xrightarrow{(q,e)/\mathcal{U}_j} L_j (\mathcal{C}'_j, \beta'_j)$$

$$L_j \neq \emptyset$$

$$\mathcal{U}'_j = \text{map } (\widehat{\beta_j \triangleleft \beta'_j}) \mathcal{U}_j$$

$$\frac{}{(\mathcal{E}, (\mathcal{C}_1, \beta_1), \dots, (\mathcal{C}_j, \beta_j), \dots, (\mathcal{C}_k, \beta_k)) \longrightarrow}$$

$$(\mathcal{E}'[\mathcal{U}'_j], (\mathcal{C}_1, \beta_1), \dots, (\mathcal{C}'_j, \beta_j \triangleleft \beta'_j), \dots, (\mathcal{C}_k, \beta_k))$$

Global Stuttering rule

$$(1) \quad q \in \mathbf{R}_{H_j} \mathcal{C}_j \beta_j \quad (1)$$

$$(2) \quad \text{Sel } \mathcal{E} q e \in \mathcal{E}' \quad (2)$$

$$(3) \quad \text{PE}_H \mathcal{C}_j \beta_j \mathcal{E} = \emptyset \quad (3)$$

$$(4) \quad \frac{}{(\mathcal{E}, (\mathcal{C}_1, \beta_1), \dots, (\mathcal{C}_j, \beta_j), \dots, (\mathcal{C}_k, \beta_k)) \longrightarrow}$$

$$(\mathcal{E}', (\mathcal{C}_1, \beta_1), \dots, (\mathcal{C}_j, \beta_j), \dots, (\mathcal{C}_k, \beta_k))$$

Fig. 3. $\delta\mu$ Charts Transition Relation definition.

The definition of the transition relation for $\delta\mu$ Charts is given in Fig. 3. It is composed of two top-level rules. The Global Progress rule produces all non-stuttering steps (premise (4)). Notice that only relevant queues are considered (premise (1)); in fact non relevant queues would generate (undesired) stuttering, as can be derived from the definitions. As usual, higher order function *map* applied on function β and sequence \mathcal{U} returns the sequence obtained by applying β to each and every element of \mathcal{U} (premise (5)). The Global Stuttering rule takes care of stuttering. When there are no potentially enabled transitions (premise (3)), from Lemma 1 (i) we know that *only* stuttering can occur; moreover, from (ii) of the same lemma we know that *any* stuttering situation will require that no transition is potentially enabled. Notice that also in this rule we restrict to *relevant* queues (premise (1)): we restrict to those stuttering steps which involve relevant queues. Should we have chosen a queue which is not relevant, we would have ended up with a step leading to a multi-queue where an element of such a non relevant queue (premise (2)) would have been removed. We consider this undesired behaviour (a too much blind scheduler!). Finally, the choice of different relevant queues produces different stuttering steps, due to different multi-queues in the next global state (premise (2) and consequent); on the other hand, all configurations and store remain unchanged, including those of the HA which generated the stuttering step (Lemma 2).

4 Example: The Hand-Over Protocol

In this section we model the Hand-over protocol for mobile phones as described in [3]. The example scenario consists of a mobile station, a switching center and two base stations. The mobile station is mounted in a car moving through two different geographical areas (cells) and provides services to an end user; the switching center is the controller of the radio communications within the whole area composed by the two cells; the two base stations, one for each cell, are the interfaces between the switching center and the mobile station. The switching center receives messages addressed to the car (user) from the external environment and forwards them to the base station of the cell where the car currently resides. The base station of such cell forwards these messages to the car which presents them to the user. The communication between the base station and the car takes place via a private data channel shared with the car while the latter is in the related cell. As soon as the switching center is signalled of the fact that the car is leaving the current cell and entering the other one, it sends the base station of the current cell the control information necessary to the car in order to get connected to the base station of the other cell. The base station of the current cell forwards this information to the car using a private control channel shared with the car while the latter is in the related cell. The above mentioned information consists of the references to the data channel and the control channel of the other base station. Finally the current base station acknowledges the switching center and waits idle to be resumed by the latter. Once the car received the control information mentioned above, it uses it for getting messages from the other cell. Once the switching center received the acknowledgement from the base station of the current cell it wakes up the other one. At this point the flow of messages from the external environment to the car (user) continues, but using the base station of the other cell. The complete $\delta\mu$ Chart of the protocol is given in Fig. 4. We use the convention of writing queue names in upper-case characters, while variables are written in lower-case characters⁸. We abstract from the user, which we actually consider part of the environment; so, the messages received by the car are actually sent out/back to the environment. Moreover, we abstract from the real content of the messages; we use a single value MSG for representing any message. We also abstract from the details by which the switching center is notified that the car moved from one cell to the other; we model this situation in the environment using non-determinism. Finally we assume that initially the car resides in the cell associated to the first base station. The elements of the multi-queue are assumed to be FIFO queues. The alphabet of the $\delta\mu$ Chart is the set of all queue names appearing in the Statecharts in the figure. The only name which does not correspond to a queue in the multi-queue is obviously MSG. Initially, all queues are empty

⁸ Notice that in the example we use a slight extension of the notation presented in Sect. 3 consisting in letting transitions be labeled by a *sequence* of output actions instead of a single destination/event pair. Such extension does not affect the semantics at a conceptual level, but helps very much in writing concise and effective specifications.

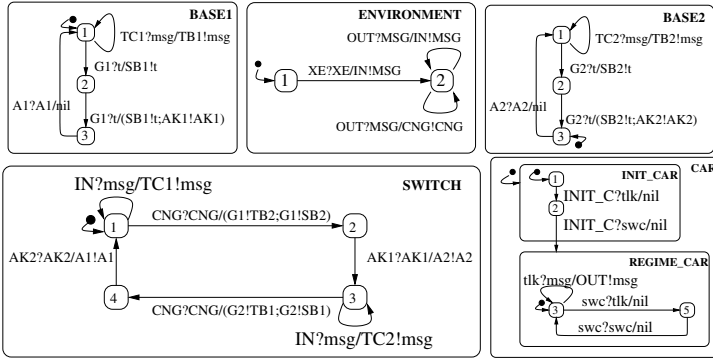


Fig. 4. The Hand-over Protocol $\delta\mu$ Chart.

except INIT_C and XE, containing the values, $\langle TB1:SB1 \rangle$ (TB1 at the top) and $\langle XE \rangle$ respectively. The ENVIRONMENT initially sends a message to the SWITCH via queue IN. Subsequently, on receiving a message from queue OUT, it may non-deterministically send the next message via queue IN or send the indication (CNG) that the car moved to the other cell, via queue CNG. Notice that activation of ENVIRONMENT is done via auxiliary queue/event XE. The switching center, modeled by SWITCH, in its initial state (1) is ready to receive either messages from queue IN or the information that the base station to which the car is connected must be changed (message CNG from queue CNG). Upon receiving a CNG indication in state (1) the SWITCH sends to BASE1 via queue G1 the data (TB2) and control (SB2) channels of BASE2, moving to state (2). Upon receiving a message from queue IN in state (1) the SWITCH forwards the message, bound to variable *msg*, to BASE1, via queue TC1, moving back to state (1). This message-forwarding loop goes on until/unless a CNG event arrives. In state (2) SWITCH waits for the ack AK1 from BASE1 via queue AK1, and after that it sends the wakeup event A2 via queue A2 to BASE2 and moves to state (3), which is symmetric to state (1). State (4) is symmetric to (2) with BASE1 and BASE2 swapped. The specification of the base stations, BASE1 and BASE2 should be self-explanatory. Also the specification of the CAR should be easy to understand. Notice that the current data channel is stored in variable *tlk* while the current control channel is kept in *swc*.

5 Conclusions

In this paper we presented an extension of μ Charts as a first step towards modeling mobility issues. In particular we addressed issues concerning device mobility, which imply the ability to dynamically change the system interconnection structure, by “opening” and “closing” connections (mobile computing [1]).

There are important features of mobile systems that cannot be expressed easily using only an asymmetric style of communication. Thus, our extension

consists in letting the *trigger event* specification of transition labels be equipped with the explicit reference to the queue from which the event should be taken. Such a reference can also be a queue name variable, thus allowing dynamicity in the choice. A formal semantics definition has been given for the extension using deductive techniques, and some correctness results concerning requirements put by the official UML semantics have been shown. As an example of use of the new notation a specification of the Hand-over protocol has been provided.

It is worth pointing out that our (core) semantics definition is a slight extension of the definition we proposed in [7]. In particular it preserves its hierarchical/recursive nature. We have used the latter definition (or minor extension thereof) for covering several aspects of UMLSCs, like stochastic ones, model-checking, and formal testing. This proves the high modularity and flexibility of our approach. Moreover, the use of recursive definitions on hierarchical structures greatly simplified the proofs of the properties of interest.

We plan to define a mapping from $\delta\mu$ Charts to PROMELA for efficient model-checking with SPIN [5]. Such work will be based on similar work we have already successfully done for μ Charts [6]. The PROMELA translator for μ Charts is currently being implemented by Intecs Sistemi s.r.l. in the context of the project PRIDE funded by the Italian Space Agency. Another line of research which we are currently investigating is to further extend our model with localities in order to explicitly model dynamic code creation/removal and mobility as migration in the sense of *mobile computations* [1]. Finally, there are several features of UMLSCs which we did not address in the present paper but which can be added to our model. Object features are already dealt with in the above mentioned work on UML with localities and data/variables are already incorporated in the context of PRIDE. Exit/entry events can be dealt with as in [15]. Deferred events can be incorporated by extending the selection predicate *Sel*. History states can be modeled by a proper re-definition of function *DST* and the use of stores.

References

1. L. Cardelli and A. Gordon. Mobile ambients. In M. Nivat, editor, *FoSSaCS'98*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–145. Springer-Verlag, 1998.
2. P. Denning. Fault tolerant operating systems. *ACM Computing Surveys*, 8(4):359–389, 1976.
3. G. Ferrari, S. Gnesi, U. Montanari, M. Pistore, and G. Ristori. Verifying mobile processes in the HAL environment. In A. Hu and M. Vardi, editors, *Computer Aided Verification*, volume 1427 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.
4. S. Gnesi, D. Latella, and M. Massink. Modular semantics for a UML Statechart Diagrams kernel and its extension to Multicharts and Branching Time Model Checking. *The Journal of Logic and Algebraic Programming. Elsevier Science*, 51(1):43–75, 2002.
5. G. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

6. D. Latella, I. Majzik, and M. Massink. Automatic verification of a behavioural subset of UML statechart diagrams using the SPIN model-checker. *Formal Aspects of Computing. The International Journal of Formal Methods*. Springer-Verlag, 11(6):637–664, 1999.
7. D. Latella, I. Majzik, and M. Massink. Towards a formal operational semantics of UML statechart diagrams. In P. Ciancarini, A. Fantechi, and R. Gorrieri, editors, *IFIP TC6/WG6.1 Third International Conference on Formal Methods for Open Object-Oriented Distributed Systems*, pages 331–347. Kluwer Academic Publishers, 1999. ISBN 0-7923-8429-6.
8. D. Latella and M. Massink. Relating testing and conformance relations for UML Statechart Diagrams Behaviours. Technical Report CNUCE-B4-2002-001, Consiglio Nazionale delle Ricerche, Istituto CNUCE, 2002. (Full version).
9. D. Latella and M. Massink. On mobility extensions of UML Statecharts; a pragmatic approach. Technical Report 2003-TR-12, Consiglio Nazionale delle Ricerche, Istituto di Scienza e Tecnologie dell’Informazione ‘A. Faedo’, 2003.
10. 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 Science - ASIAN’97*, volume 1345 of *Lecture Notes in Computer Science*, pages 181–196. Springer-Verlag, 1997.
11. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Information and Computation*, 100(1):1–77, 1992. Parts 1-2.
12. Object Management Group, Inc. *OMG Unified Modeling Language Specification - version 1.3*, 1999.
13. J. Philipps and P. Scholz. Compositional specification of embedded systems with statecharts. In M. Bidoit and M. Dauchet, editors, *TAPSOFT ’97: Theory and Practice in Software Development*, volume 1214 of *Lecture Notes in Computer Science*, pages 637–651. Springer-Verlag, 1997.
14. P. Scholz and D. Nazareth. Communication concepts for statecharts: A semantic foundation. In M. Bertran and T. Rus, editors, *Transformation-based Reactive Systems Development*, volume 1231 of *Lecture Notes in Computer Science*, pages 126–140. Springer-Verlag, 1997.
15. M. von der Beeck. A structured operational semantics for UML-statecharts. *Software Systems Modeling*. Springer, (1):130–141, 2002.