On Mobility Extensions of UML Statecharts. A Pragmatic Approach *

Diego Latella and Mieke Massink

Consiglio Nazionale delle Ricerche, Istituto di Scienze e Tecnologie dell'Informazione - 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. The extension builds on the notion of μCharts, as proposed in [4]. A μChart is a collection of Statecharts, where each Statechart is associated with a unique input queue. In a μChart each Statechart can address its output events directly to each (other) Statechart of the μChart by explicitly mentioning the name of (the unique queue of) such Statechart in its output actions. The extension we present consists in relaxing the unique association between each Statechart and its input-queue and in allowing the 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.

Keywords: UML Statecharts, Mobility and Global Computing, Formal Semantics, ACM-F.1.2, MSC-68Q85.

1 Introduction

The Unified Modelling Language (UML) is a graphical modelling language for object-oriented software and systems [11]. 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] μCharts have been introduced together with their formal semantics. Briefly, a μChart 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 μCharts is an interleaving one with an asynchronous/asymmetric/static pattern of communication, while within each component UMLSC a clock-synchronous model of computation is used. The semantics of a μChart 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, 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 event to be sent and the specification of a *destination* component (queue) to which it must be delivered[1]. 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

* This work has been carried out in the context of Project EU-IST IST-2001-32747 Architectures for Mobility (AGILE), http://jazz.pst.informatik.uni-muenchen.de/projekte/agile/.

[1] In [4] a set of destinations can be specified instead of a single one. Here we restrict the destinetion to a singleton for simplicity.

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 easily 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. In a more general setting, 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 [12].

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 $\pi-$calculus [10], although in the more imperative-like framework of UMLSCs. Two variants of the semantics are provided for this extension. In the first one, the binding of the input value to a variable used in the input part of a transition label is actually performed *immediately after* the transition is fired. In other words such a value is accessible via the variable only from the *next* state configuration (on). This choice generates a one-step delay for the accessibility of variable values, which usually implies the use of extra-states and extra transition stimuli which may make specification difficult to define and to understand. Consequently we have defined a second variant of the semantics, which we call the "early-binding" semantics, where the input value bound to a variable in the input side of a transition is *immediately* available in the output action part of the transition. In this way, specifications get much simpler and more concise.

The paper is organized as follows: in Sect.2 μCharts are briefly recalled and some basic definitions are given. Sect.3 describes the extension we propose, which is refined in Sect.4 where "early-binding" is dealt with. For both extensions a formal semantics is given using deductive techniques. Two versions of the Hand-over protocol specification are given in order to show the advantages of early-binding. Finally in Sect. 5 some conclusions are drawn and directions for future work are sketched. Detailed proofs relevant to the present paper are given in the appendix.

2 μCharts

In this section the basic definitions related to μCharts are briefly recalled. They are treated in depth in [4] where the interested reader is referred to. We use Hierarchical Automata (HAs) 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 Fig.1. 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 action*, 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. In the sequel we will be concerned only with HAs. The structure of transition labels will be properly accommodated in order to support the mobility extensions.

A μChart is a collection of UML Statecharts communicating via input queues. In our work we consider a restricted subset of UML Statcharts, 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; variables and data are restricted to 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].

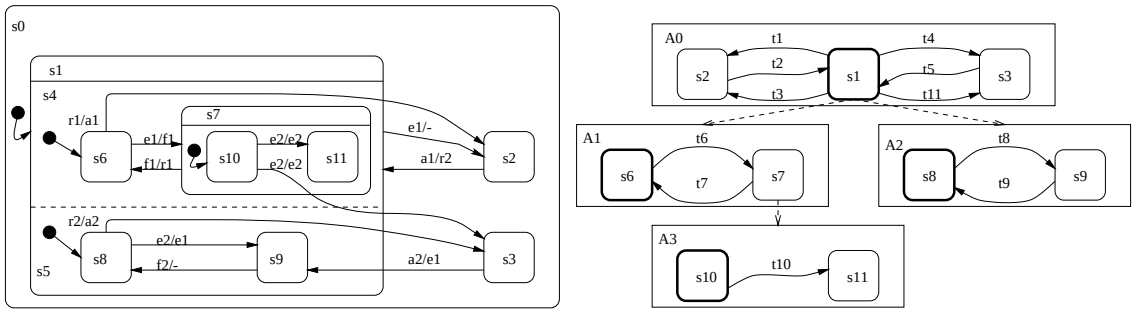

Fig. 1. A UMLSC and its HA

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	$e2$	$e2$
$TD\ t$	$\emptyset$	$\{s6, s8\}$	$\emptyset$	$\emptyset$	$\{s6, s9\}$	$\{s10\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

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

2.1 Basic definitions

The first notion we need to define is that of (sequential) automaton[2].

[2] 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. $f\ a_1\ a_2 \ldots\ a_n$ will be used instead of $f(a_1, a_2, \ldots, a_n)$ and function application will be considered left-associative; (ii) for function $f : X \mapsto Y$ 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 $P\ x$"

Definition 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.

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 $SRC, TGT : \delta_A \mapsto \sigma_A$ 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 $E \subset \mathcal{N}$; the* refinement function *$\rho : \bigcup_{A \in F} \sigma_A \mapsto 2^F$ 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. From the above definition the reader can see that the only type of events we deal with are queue names. The following are auxiliary definitions in the context of HAs. 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 proper restriction of ρ.

Definition 3. *For $A \in F$ the* automata, states *and* transitions under *A are defined respectively as*
$\mathcal{A}\ A \triangleq \{A\} \cup (\bigcup_{A' \in \left(\bigcup_{s \in \sigma_A} (\rho_A s)\right)} (\mathcal{A}\ A'))$, $\mathcal{S}\ A \triangleq \bigcup_{A' \in \mathcal{A}\ A} \sigma_{A'}$, *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, ρ_A), where $F_A = (\mathcal{A}\ A)$, and $\rho_A = \rho_{|(\mathcal{S}\ A)}$, 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. 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 $t \# t'$. Priorities are assigned to (conflicting) transitions via a function π and a partial order $\sqsubseteq$ is defined on them, based on the state nesting partial order $\preceq$. So, we say that t has lower priority than (the same priority as) t' iff $\pi t \sqsubseteq \pi t'$. Intuitively transitions coming from deeper states have higher priority; the reader interested in the formal definitions of state precedence, conflict and priority is referred to [7, 4]. μCharts are defined as follows:

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

The complete formal semantics for μCharts which has been defined in [4] is recalled in the Appendix. A μChart is mapped into a LTS, each state of which coincides with a tuple of configuration-queue pairs of all all HAs in the μChart. The transition relation of the LTS is defined by means of a derivation system composed by four rules. 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 μCharts 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)* $\forall s, A.\ s \in \mathcal{C} \wedge A \in \rho\ s \Rightarrow \exists_1 s' \in A.\ s' \in \mathcal{C}$

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 $\mathcal{P} \stackrel{\Delta}{=} \mathcal{N} \times \mathcal{N}$, that is, in this paper message bodies can be only queue names, for the sake of simplicity. Due to the presence of variables we need also packet *terms* and stores. The set $\mathcal{P}t$ of packet terms is defined as $\mathcal{P}t \stackrel{\Delta}{=} (\mathcal{N} \cup \mathcal{V}) \times (\mathcal{N} \cup \mathcal{V})$. Stores are defined as follows:

Definition 7 (Stores). *A* store β *is a partial function* $\beta : \mathcal{V} \mapsto \mathcal{N}$. *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:*

$\hat{\beta}\ q = q$ *for* $q \in \mathcal{N}$
$\hat{\beta}\ v = \beta\ v$ *for* $v \in \mathcal{V}$
$\hat{\beta}\ (d, b) = (\hat{\beta}\ d, \hat{\beta}\ b)$ *for* $(d, b) \in \mathcal{P}t$

Definition 8 (unit store). *For* $v \in \mathcal{V}$ *and* $q \in \mathcal{N}$ *the unit store* $[v \mapsto q]$ *is defined as follows:*

$[v \mapsto q]\ v' = q$, *if* $v' = v$
$[v \mapsto q]\ v' = \perp$, *if* $v' \neq v$

Definition 9 ($\triangleleft$). *For stores* β_1 *and* β_2 *we let store* $\beta_1 \triangleleft \beta_2$ *be defined as follows:*

$(\beta_1 \triangleleft \beta_2)\ v = \beta_2\ v$, *if* $\beta_2\ v \neq \perp$
$(\beta_1 \triangleleft \beta_2)\ v = \beta_1\ v$, *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 definition of UMLSC 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). On the other hand, the nature of the environment has an impact on the structures to be used for modelling *compound output actions*, i.e. those resulting from actions of parallel components of the HA. We choose *not* to fix any particular semantics such as a set, or a multi-set or a FIFO queue etc., but to model compound actions 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. The elements of D will be called the *atomic elements* of Θ_D. In particular, for $\mathcal{D}, \mathcal{D}'$, etc. in Θ_D, and $d \in D$, $(add\ \mathcal{D}\ d)$ denotes the structure obtained by adding d to structure $\mathcal{D}$. Similarly, $(join\ \mathcal{D}\ \mathcal{D}')$ denotes the structure obtained by merging $\mathcal{D}$ with $\mathcal{D}'$. The predicate $is\_join^n_{j=1}\mathcal{D}_j\ \mathcal{J}$ states that $\mathcal{J}$ is a possible *join* of $\mathcal{D}_1 \ldots \mathcal{D}_n$ and it is a way for expressing non-deterministic merge of $\mathcal{D}_1 \ldots \mathcal{D}_n$. By $(Sel\ \mathcal{D}\ d\ \mathcal{D}')$ we mean that $\mathcal{D}'$ is the structure resulting from selecting d from $\mathcal{D}$, the selection policy depending on the choice for the particular semantics. So, for instance, if sets are chosen, then $(add\ \mathcal{D}\ d) = \mathcal{D} \cup \{d\}$, $(join\ \mathcal{D}\ \mathcal{D}') = \mathcal{D} \cup \mathcal{D}'$ and, for $d \in \mathcal{D}$, $(Sel\ \mathcal{D}\ d\ \mathcal{D}') \equiv (\mathcal{D}' = \mathcal{D} \setminus \{d\})$. 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 $\mathcal{D}'$. Furthermore, for $d \in D, \mathcal{D} \in \Theta_D$ and $\mathcal{U} \in \Theta_{(D \times D)}$ we let $ext\ d\ \mathcal{D}\ \mathcal{U}$ be defined as $join\ \mathcal{D}\ (\mathcal{U} \downarrow d)$, where $(\mathcal{U} \downarrow d)$ is the projection of $\mathcal{U}$ on d, namely the structure obtained by first removing from $\mathcal{U}$ those elements (d_1, d_2) the first component of which is different from d (i.e. $d_1 \neq d$) and then taking the structure composed only of the second components of the remaining elements. For instance, for FIFO queue $\mathcal{U} =< (a, g) : (c, m) : (d, g) : (d, d) : (d, m) >$ we have $(\mathcal{U} \downarrow a) =< g >$,

and $(\mathcal{U} \downarrow d) =< g : d : m >$. Finally, given sequence $r \in D^*$, *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.

3 μCharts with Explicit Dynamic Channels

In this section we shall describe our mobility extension of μCharts. 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. The introduction of variables requires a re-definition of the transition labels HAs. Let $H = (F, E, \rho)$ be a HA of a μChart 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 *trigger event* $EV\ t$, the *input-queue* $IQ\ t$, the *destination queue* $DQ\ t$ and the *output event* $AC\ t$ 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:

- $Sel\ \mathcal{E}\ q\ e\ \mathcal{E}' \equiv e$ 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 $\mathcal{E}$.
- $\mathcal{E}[(q, e)]$ is 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 $\mathcal{U} \in \Theta_{\mathcal{P}}$, $\mathcal{E}[\mathcal{U}]$ is defined in the obvious way; for instance if $\mathcal{U}$ is a sequence $(q_1, e_1), (q_2, e_2), \ldots, (q_n, e_n)$ we have $\mathcal{E}[\mathcal{U}] = (\mathcal{E}[(q_1, e_1)])[(q_2, e_2), \ldots, (q_n, e_n)]$.
The Operational Semantics of $\delta\mu$Chart $S = (E, H_1, \ldots H_k)$ is a transition system. Each global state is a tuple $(\mathcal{E}, Loc_1, \ldots, Loc_k)$. $\mathcal{E}$ 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 $(\mathcal{C}_j, \beta_j)$ 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 [11] 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 from the fact 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, deferred[3]). In the case of $\delta\mu$Charts, as we mentioned above, there is no longer a single input queue associated to each HA, but a collection of queues - the multi-queue - and (dynamic) associations between such queues and the HAs of the $\delta\mu$Chart. Thus, in $\delta\mu$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. We anticipate here only that the queue selection issue is related to stuttering: for instance, selecting a queue which is mentioned only in the input labels of transitions the sources of which are *not* in the current configuration would bring to an obvious stuttering. We consider such a stuttering choice inappropriate[4].

[3] We do not deal with deferred events in our current work.
[4] Of course such a judgment is part of our notation design decisions and is not defined in the current definition of the UML.

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 $A \uparrow P :: (\mathcal{C}, \beta) \stackrel{(q,e)/\mathcal{U}}{\longrightarrow}_L (\mathcal{C}', \beta')$ which models the step-transitions of generic 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, $A \uparrow P :: (\mathcal{C}, \beta) \stackrel{(q,e)/\mathcal{U}}{\longrightarrow}_L (\mathcal{C}', \beta')$ should be read as "A, on configuration and store $(\mathcal{C}, \beta)$, with input event e from queue q can fire the transitions in the set L moving to configuration and store $(\mathcal{C}', \beta')$, 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 the following auxiliary functions:

$$LE_A\ \mathcal{C}\ \beta\ q\ e \stackrel{\Delta}{=} \{t \in \delta_A \mid \{(SRC\ t)\} \cup (SR\ t) \subseteq \mathcal{C} \wedge \hat{\beta}(IQ\ t) = q \wedge \texttt{MATCH}\ e\ (EV\ t) \wedge (\mathcal{C}, \beta, e) \models (G\ t)\}$$

$$E_A\ \mathcal{C}\ \beta\ q\ e \stackrel{\Delta}{=} \bigcup_{A' \in (\mathcal{A}\ A)} LE_{A'}\ \mathcal{C}\ \beta\ q\ e$$

where function MATCH is defined as follows: $\texttt{MATCH}\ e\ x \stackrel{\Delta}{=} (x \in \mathcal{V} \vee x = e)$.
For generic HA A, $LE_A\ \mathcal{C}\ \beta\ q\ e$ is 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. In order for a transition $t \in \delta_A$ to be enabled it is required that all its source states, i.e. both its proper source state $SRC\ t$ and its source restriction $SR\ t$ are included in the current configuration $\mathcal{C}$; moreover, the input queue referred to in the transition, $IQ\ t$, must be q - i.e $IQ\ t$ must evaluate to q - and the specification of the trigger event $EV\ t$ must *match* the input event - i.e. it must either be the same event or a variable; finally, the guard $G\ t$ must be satisfied[5]. Function E_A extends LE_A in order to cover *all* the transitions of A including those of sub-automata of A. Notice that $\hat{\beta}(IQ\ t) = \perp$ implies $LE_A\ \mathcal{C}\ \beta\ q\ e = \emptyset$.

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). As an effect of firing transition t the event $\hat{\beta}\ (AC\ t)$ is sent to destination $\hat{\beta}\ (DQ\ t)$. Notice that the destination and message body in the action of a transition are computed using the store available *before* the transition is fired, as it can be seen in the consequent of the Progress Rule.

For transition t, $DST\ t$ is defined as the set $\{s \mid \exists s' \in (TD\ t).\ (TGT\ t) \preceq s \preceq s'\}$. 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:

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: $A \uparrow P :: (\mathcal{C}, \beta) \stackrel{(q,e)/\mathcal{U}}{\longrightarrow}_L (\mathcal{C}', \beta')$ for some $\mathcal{C}', \beta'$ iff L is a maximal set, under set inclusion, which satisfies all the following properties: (i) L is conflict-free, i.e. $\forall t, t' \in L.\ \neg t \# t'$;*

[5] In this paper we do not deal with guard evaluation issues. They do not raise any particular interesting problem.

Progress rule

$$\frac{\begin{array}{ll} t \in LE_A\ \mathcal{C}\ \beta\ q\ e & (1)\\ \beta' = \text{if } (EV\ t) \in \mathcal{V} \text{ then } [(EV\ t) \mapsto e] \text{ else } \perp\!\!\!\perp & (2)\\ \nexists t' \in P \cup E_A\ \mathcal{C}\ \beta\ q\ e.\ \pi t \sqsubset \pi t' & (3)\end{array}}{A \uparrow P :: (\mathcal{C}, \beta) \overset{(q,e)/\hat{\beta}\ (DQ\ t, AC\ t)}{\longrightarrow}_{\{t\}} (DST\ t, \beta')}$$

Composition Rule

$$\frac{\begin{array}{ll} \{s\} = \mathcal{C} \cap \sigma_A & (1)\\ \rho_A\ s = \{A_1, \ldots, A_n\} \neq \emptyset & (2)\\ \left(\bigwedge_{j=1}^{n} A_j \uparrow P \cup LE_A\ \mathcal{C}\ \beta\ q\ e :: (\mathcal{C}, \beta) \overset{(q,e)/\mathcal{U}_j}{\longrightarrow}_{L_j} (\mathcal{C}_j, \beta_j)\right) \wedge is\_join_{j=1}^{n} \mathcal{U}_j\ \mathcal{U} & (3)\\ \left(\bigcup_{j=1}^{n} L_j = \emptyset\right) \Rightarrow (\forall t \in LE_A\ \mathcal{C}\ \beta\ q\ e.\ \exists t' \in P.\ \pi t \sqsubset \pi t') & (4)\end{array}}{A \uparrow P :: (\mathcal{C}, \beta) \overset{(q,e)/\mathcal{U}}{\longrightarrow}_{\bigcup_{j=1}^{n} L_j} (\{s\} \cup \bigcup_{j=1}^{n} \mathcal{C}_j, \bigcup_{j=1}^{n} \beta_j)}$$

Stuttering Rule

$$\frac{\begin{array}{ll} \{s\} = \mathcal{C} \cap \sigma_A & (1)\\ \rho_A\ s = \emptyset & (2)\\ \forall t \in LE_A\ \mathcal{C}\ \beta\ q\ e.\ \exists t' \in P.\ \pi t \sqsubset \pi t' & (3)\end{array}}{A \uparrow P :: (\mathcal{C}, \beta) \overset{(q,e)/nil}{\longrightarrow}_{\emptyset} (\{s\}, \perp\!\!\!\perp)}$$

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

(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

As mentioned above, the selection from the multi-queue is tightly connected to the possibility of generating unwanted extra-stuttering. In this paper we make the following choice:

- 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 *local relevant* and the *relevant* queues of $A \in H = (F, E, \rho)$ are defined as follows :

$$LR_A\ \mathcal{C}\ \beta \triangleq \{q \in E \mid \exists t \in \delta_A.\ (SRC\ t) \in \mathcal{C} \wedge \hat{\beta}(IQ\ t) = q\}$$

$$R_A\ \mathcal{C}\ \beta \triangleq \bigcup_{A' \in (\mathcal{A}\ A)} LR_{A'}\ \mathcal{C}\ \beta$$

For $A \in F$ of $HA\ H = (F, E, \rho)$, $\mathcal{C} \in \mathrm{Conf}_A$, store β, multiqueue $\mathcal{E}$ on E, the set of *Potentially Enabled Transitions* of A, on configuration $\mathcal{C}$, store β and multi-queue $\mathcal{E}$ is the set $PE_A\ \mathcal{C}\ \beta\ \mathcal{E}$ defined below:

$$PE_A\ \mathcal{C}\ \beta\ \mathcal{E} \stackrel{\Delta}{=} \{t \in \mathcal{T}\ A \mid \exists q \in R_A\ \mathcal{C}\ \beta, e \in E.\ Sel\ \mathcal{E}\ q\ e\ \mathcal{E}' \wedge t \in E_A\ \mathcal{C}\ \beta\ q\ e\}$$

Obviously, the fact that a transition $t \in PE_A\ \mathcal{C}\ \beta\ \mathcal{E}$ 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. Lemma 2 guarantees that stuttering does not change the store and the configuration.

Lemma 1.
For all $H = (F, E, \rho), \mathcal{C} \in \mathrm{Conf}_H$, store $\beta, \mathcal{E}$ multiqueue on E

i) for all $q, e \in E, \mathcal{E}'$ multiqueue on E:
if $PE_H\ \mathcal{C}\ \beta\ \mathcal{E} = \emptyset$ and $q \in R_H\ \mathcal{C}\ \beta$, and $Sel\ \mathcal{E}\ q\ e\ \mathcal{E}'$ and $H \uparrow \emptyset :: (\mathcal{C}, \beta) \stackrel{(q,e)/\mathcal{U}}{\longrightarrow}_L (\mathcal{C}', \beta')$ for some $\mathcal{C}', \beta', L$, then $L = \emptyset$

ii) if for all $q, e \in E, \mathcal{E}'$ multiqueue on E such that $Sel\ \mathcal{E}\ q\ e\ \mathcal{E}'$, $\mathcal{C}', \beta'$ we have $H \uparrow \emptyset :: (\mathcal{C}, \beta) \stackrel{(q,e)/\mathcal{U}}{\longrightarrow}_\emptyset (\mathcal{C}', \beta')$ then we also have $PE_A\ \mathcal{C}\ \beta\ \mathcal{E} = \emptyset$

Proof. Trivial from Lemma 3 in the appendix when $A = H$ and $P = \emptyset$.

Lemma 2.
For all $H = (F, E, \rho), \mathcal{C} \in \mathrm{Conf}_H$, store $\beta, q, e \in E$ if $H \uparrow \emptyset :: (\mathcal{C}, \beta) \stackrel{(q,e)/\mathcal{U}}{\longrightarrow}_\emptyset (\mathcal{C}', \beta')$ then $\mathcal{C}' = \mathcal{C}$ and $\beta' = \perp\!\!\!\perp$.

Proof. Similar to the proof of Lemma A.1 (ii) in [8].

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 it can be derived from the definitions. 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).
As for μCharts computations are defined also for $\delta\mu$Charts in the obvious way:

Definition 10 (Computations). *A* computation *of $\delta\mu$Chart $S = (E, H_1, \ldots H_k)$ is a sequence of global states of S of the form $GS_0 \longrightarrow GS_1 \longrightarrow GS_2 \ldots GS_n \ldots$, where $GS_j \longrightarrow GS_{j+1}$ is obtained using the rules of Fig.3 and which is* maximal, *i.e. either it is infinite or it is finite with terminal element GS_n which has the property that $GS_n \longrightarrow GS'$ for no GS'.*

Global Progress rule

$$\frac{\begin{array}{ll} q \in R_{H_j}\ \mathcal{C}_j\ \beta_j & (1)\\ Sel\ \mathcal{E}\ q\ e\ \mathcal{E}' & (2)\\ H_j \uparrow \emptyset :: (\mathcal{C}_j, \beta_j) \overset{(q,e)/\mathcal{U}_j}{\longrightarrow}_{L_j} (\mathcal{C}'_j, \beta'_j) & (3)\\ L_j \neq \emptyset & (4) \end{array}}{\begin{array}{l} (\mathcal{E}, (\mathcal{C}_1, \beta_1), \ldots, (\mathcal{C}_j, \beta_j), \ldots, (\mathcal{C}_k, \beta_k)) \longrightarrow \\ (\mathcal{E}'[\mathcal{U}_j], (\mathcal{C}_1, \beta_1), \ldots, (\mathcal{C}'_j, \beta_j \triangleleft \beta'_j), \ldots, (\mathcal{C}_k, \beta_k)) \end{array}}$$

Global Stuttering rule

$$\frac{\begin{array}{ll} q \in R_{H_j}\ \mathcal{C}_j\ \beta_j & (1)\\ Sel\ \mathcal{E}\ q\ e\ \mathcal{E}' & (2)\\ PE_H\ \mathcal{C}_j\ \beta_j\ \mathcal{E} = \emptyset & (3) \end{array}}{\begin{array}{l} (\mathcal{E}, (\mathcal{C}_1, \beta_1), \ldots, (\mathcal{C}_j, \beta_j), \ldots, (\mathcal{C}_k, \beta_k)) \longrightarrow \\ (\mathcal{E}', (\mathcal{C}_1, \beta_1), \ldots, (\mathcal{C}_j, \beta_j), \ldots, (\mathcal{C}_k, \beta_k)) \end{array}}$$

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

3.3 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 (see Fig.4):

- A mobile station mounted in a car moving through two different geographical areas (cells), that provide services to an end user;
- A switching center that is the controller of the radio communications within the whole area composed by the two cells;
- Two base station modules, one for each cell, that are the interfaces between the Switching Center and the mobile station.

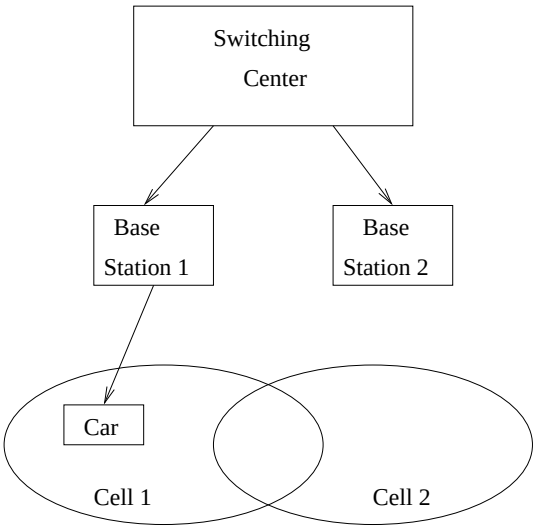

Fig. 4. The Hand-over Protocol

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. 5[6]. We use the convention of writing queue names in upper-case characters, while variables are written in lower-case characters [7]. 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 of the fact that the car moved from one cell to the other; we model this situation also 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 picture, where also the initial value of the multi-queue is given: all queues are empty except INIT_C and XE, containing the values, <TB1:SB1> and <XE> respectively. The only name which does not correspond to a queue is obviously MSG.

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. Such X-queues/events will be used elsewere in the specification as a means for (re-)activation.

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 (3). Upon receiving a message from queue IN in state (1) the SWITCH moves to state (2) sending the reactivation event XS to queue XS. In state (2) the last message received via IN is accessible via local variable msg[8]. Using this variable, and being reactivated by event XS, SWITCH forwards the message to BASE1, via queue TC1, moving back to state (1). This message-forwarding loop goes on until/unless a CNG event arrives. In state (3) 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 (4), which is symmetric to state (1). State (5), resp. (6), is symmetric to (2), resp. (3), with BASE1 and BASE2 swapped.

---

[6] 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.

[7] 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.

[8] We remind the reader that a message bound to a variable via an input action of a transition is available only *after* the transition is fired.

The specification of the base stations, BASE1 and BASE2 should be self-explanatory. Notice that they are essentially the same kind of statechart instantiated to different channels and with different initial state. 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*. In all statecharts, the fact that input variables are bound to their input values only *after* the transitions in which they appear have been fired is dealt with by means of the introduction of an extra-state plus a self-reactivation event.

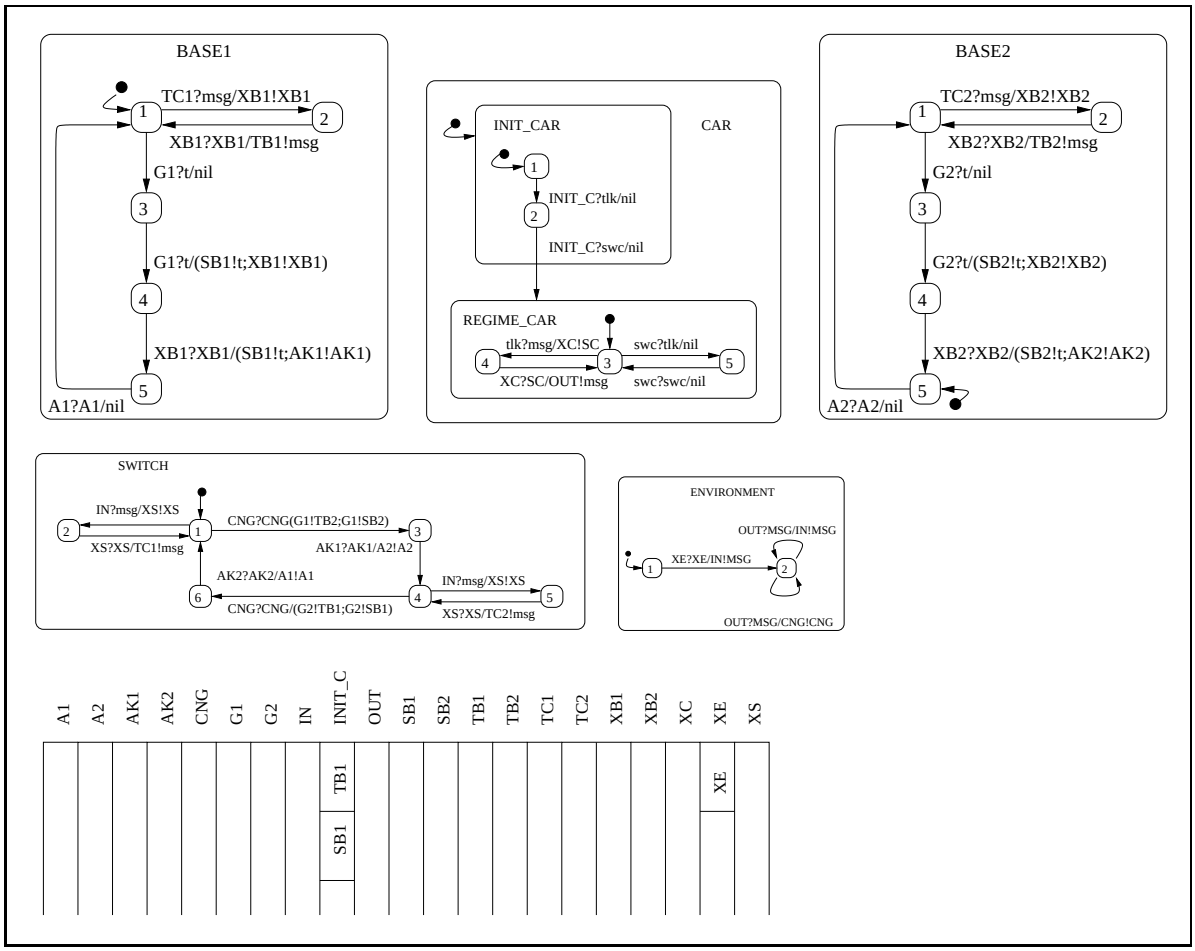

Fig. 5. The Hand-over Protocol $\delta\mu$Chart with its initial multi-queue

4 μCharts with Explicit Dynamic Channels with Early Binding

In Sect.3 we have seen that when *EV* t is a variable, say x, the value bound to x as effect of firing transition t is accessible, via x only *after* t has fired, namely in the *next* configuration. This extra delay often results in extra-states and can be source of confusion and errors in the specifications.

In this section we propose another version of the semantics of μCharts where the input value bound to a variable in a transition is available *immediately* in the same transition. We achieve this as follows: the core semantics does *not* evaluate the variables occurring in the output actions of transitions; it only builds a symbolic structure with them, while recording their bindings in the stores it generates as well; such a structure is instead 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. Notice that in Sect.3 the global current -not yet recording the new binding - store β is used instead.

The new core semantics is given in Fig. 6. The top rules are given in Fig. 7 where, as usual, higher order function *map* applied on function β and structure $\mathcal{D}$ returns the structure obtained applying β to each and every atomic element of $\mathcal{D}$.

Progress rule

$$\frac{\begin{array}{ll} t \in LE_A\ \mathcal{C}\ \beta\ q\ e & (1) \\ \beta' = \text{if } (EV\ t) \in \mathcal{V} \text{ then } [(EV\ t) \mapsto e] \text{ else } \perp\!\!\!\perp & (2) \\ \nexists t' \in P \cup E_A\ \mathcal{C}\ \beta\ q\ e.\ \pi t \sqsubset \pi t' & (3) \end{array}}{A \uparrow P :: (\mathcal{C}, \beta) \xrightarrow{(q,e)/(DQ\ t, AC\ t)}_{\{t\}} (DST\ t, \beta')}$$

Composition Rule

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

Stuttering Rule

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

Fig. 6. Core operational semantics rules for $\delta\mu$Charts. Early binding

A new version of the Hand-over protocol, using the Early binding semantics is given in Fig. 8. It is easy to see that there is no longer any need of extra-states for accessing values stored in variables in the previous step - like e.g. states (2) and (5) of SWITCH or state (2) of BASE1 or BASE2 - in Fig.5, and, consequently, for "self-reactivation" extra-messages - like XS, XB1 and XB2 in the same $\delta\mu$Chart.

5 Conclusions

In this paper we presented an extension of μCharts 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. The value received via an input action of a transition are accessible via the variables bound to them *after* the transition is fired in a step. We provided also an early-binding semantics which allows the value bound to a variable by the effect of an input action

Global Progress rule

$$\frac{\begin{array}{ll} q \in R_{H_j}\ \mathcal{C}_j\ \beta_j & (1)\\ Sel\ \mathcal{E}\ q\ e\ \mathcal{E}' & (2)\\ H_j \uparrow \emptyset :: (\mathcal{C}_j, \beta_j) \xrightarrow{(q,e)/\mathcal{U}_j}_{L_j} (\mathcal{C}'_j, \beta'_j) & (3)\\ L_j \neq \emptyset & (4)\\ \mathcal{U}'_j = map\ (\widehat{\beta_j \triangleleft \beta'_j})\ \mathcal{U}_j & (5) \end{array}}{\begin{array}{l} (\mathcal{E}, (\mathcal{C}_1, \beta_1), \ldots, (\mathcal{C}_j, \beta_j), \ldots, (\mathcal{C}_k, \beta_k)) \longrightarrow \\ (\mathcal{E}'[\mathcal{U}'_j], (\mathcal{C}_1, \beta_1), \ldots, (\mathcal{C}'_j, \beta_j \triangleleft \beta'_j), \ldots, (\mathcal{C}_k, \beta_k)) \end{array}}$$

Global Stuttering rule

$$\frac{\begin{array}{ll} q \in R_{H_j}\ \mathcal{C}_j\ \beta_j & (1)\\ Sel\ \mathcal{E}\ q\ e\ \mathcal{E}' & (2)\\ PE_H\ \mathcal{C}_j\ \beta_j\ \mathcal{E} = \emptyset & (3) \end{array}}{\begin{array}{l} (\mathcal{E}, (\mathcal{C}_1, \beta_1), \ldots, (\mathcal{C}_j, \beta_j), \ldots, (\mathcal{C}_k, \beta_k)) \longrightarrow \\ (\mathcal{E}', (\mathcal{C}_1, \beta_1), \ldots, (\mathcal{C}_j, \beta_j), \ldots, (\mathcal{C}_k, \beta_k)) \end{array}}$$

Fig. 7. $\delta\mu$Charts Transition Relation definition. Early binding

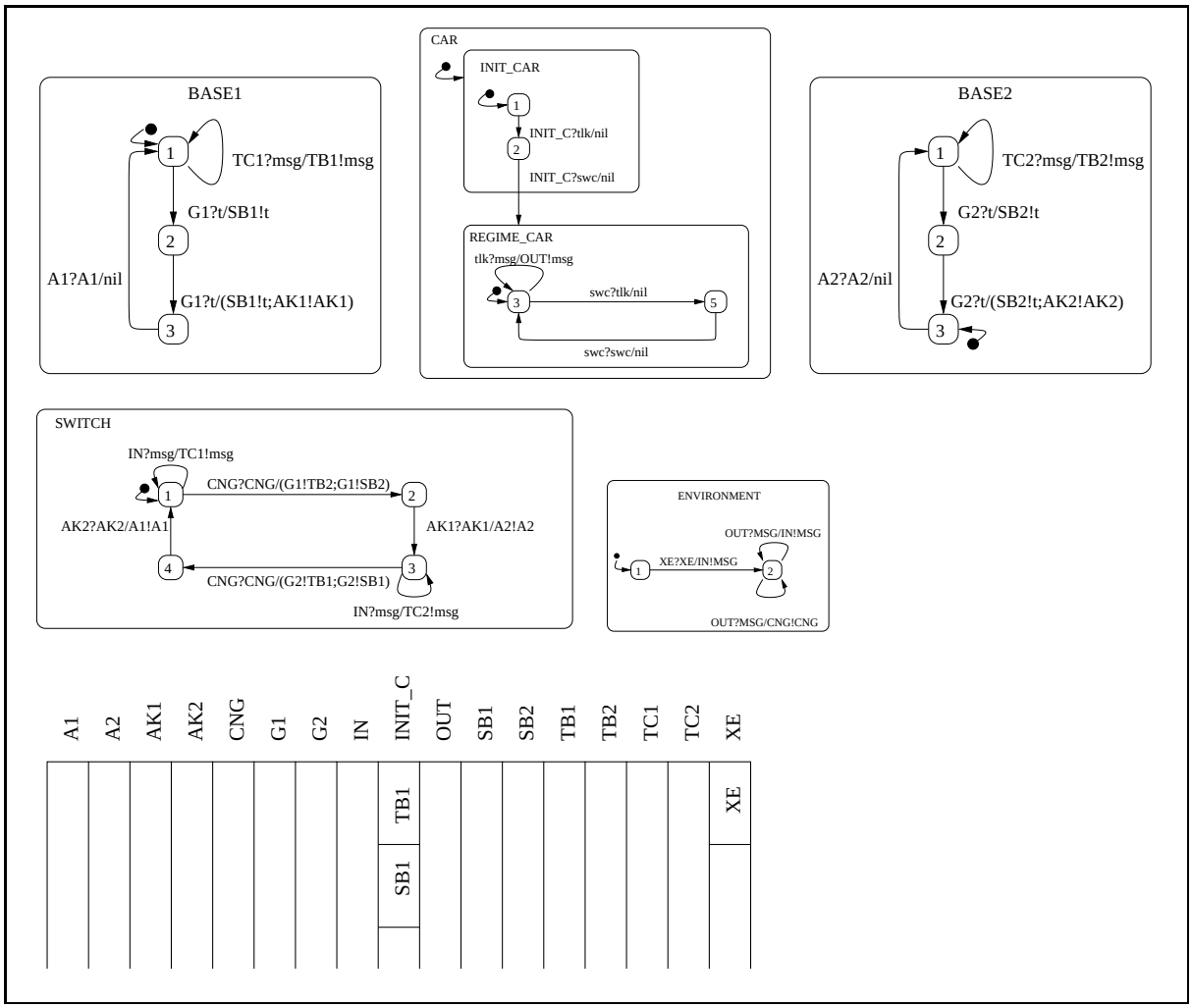

Fig. 8. The Hand-over Protocol $\delta\mu$Chart with its initial multi-queue. Early binding semantics

labeling a transition be accessible via the variable *immediately* in the output action labelling the same transition.

For both variants of the extension a formal semantics has been given 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 is provided, both using the early-binding semantics as well as the non-early binding one.

We plan to define a mapping from $\delta\mu$Charts to PROMELA for efficient model-checking with SPIN [5]. Another line of research we plan to investigate is to further extend our model with behaviour types and localities in order to explicitly model dynamic code creation/removal and mobility in the sense of *mobile computations* [1]. The paradigm we have in mind is similar to and inspired by KLAIM [2].

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. R. De Nicola, G. Ferrari, and R. Pugliese. KLAIM: A kernel language for agents interaction and mobility. *IEEE Transactions on Software Engineering*, 24(5):315–329, 1998.
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. A simplified formal operational semantics for a subset of UML statechart diagrams. Technical Report HIDE/T1.2/PDCC/5/v1, ESPRIT Project n. 27439 - High-Level Integrated Design Environment for Dependability HIDE, 1998.
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. Z. Manna, S. Ness, and J. Vuillemin. Inductive methods for proving properties of programs. *Communications of the ACM*, 16(8):491–502, 1973.
10. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Information and Computation*, 100(1):1–77, 1992. Parts 1-2.
11. Object Management Group, Inc. *OMG Unified Modeling Language Specification - version 1.3*, 1999.
12. Denning P. Fault tolerant operating systems. *ACM Computing Surveys*, 8(4):359–389, 1976.

Appendix

A Formal Operational Semantics of μCharts

In this section we recall the Formal Operational Semantics of μCharts proposed in [4] together with the main theorem proved therein.

The Operational Semantics of μChart $S = (E, H_1, \ldots H_k)$ is a transition system, where each *global state* is a tuple $(Loc_1, \ldots, Loc_k)$ - Loc_j being the current status of HA H_j - and each transition corresponds to a step of one component HA H_j. The status Loc_j of a component HA H_j is a pair $(\mathcal{C}_j, \mathcal{E}_j)$ where $\mathcal{C}_j$ is the current configuration of H_j and $\mathcal{E}_j$ is the current value of the

input queue of H_j. The transition relation is defined by means of a deduction system consisting of the rule shown in Fig.9, where, for notational simplicity, we assume that, for μChart S as above, the name of (the queue of) HA H_j is simply j. The rule stipulates that (1) from the fact that

$$\frac{\begin{array}{ll} j \in \{1, \ldots, k\} & (1) \\ (Sel\ \mathcal{E}_j\ e\ \mathcal{E}'_j) & (2) \\ H_j \uparrow \emptyset :: \mathcal{C}_j \xrightarrow{e/\mathcal{U}}_L \mathcal{C}'_j & (3) \end{array}}{((\mathcal{C}_1, \mathcal{E}_1), \ldots, (\mathcal{C}_j, \mathcal{E}_j), \ldots, (\mathcal{C}_k, \mathcal{E}_k)) \longrightarrow ((\mathcal{C}_1, (ext\ 1\ \mathcal{E}_1\ \mathcal{U})), \ldots, (\mathcal{C}'_j, (ext\ j\ \mathcal{E}'_j\ \mathcal{U})), \ldots, (\mathcal{C}_k, (ext\ k\ \mathcal{E}_k\ \mathcal{U})))}$$

Fig. 9. μCharts Transition Relation definition

the $j-th$ HA H_j is selected and (2) the fact that event e is a possible selection from the current input queue $\mathcal{E}_j$ of H_j and (3) the fact that the HA H_j, on the current configuration $\mathcal{C}_j$ and on such input event e fires all and only the transitions in set L - producing output $\mathcal{U}$ and moving to configuration $\mathcal{C}'_j$ - a transition from the current state $(Loc_1, \ldots, Loc_j, \ldots, Loc_k)$ of S to the next one $(Loc'_1, \ldots, Loc'_j, \ldots, Loc'_k)$ can be deduced. Loc'_j will be composed by the new configuration of H_j, $\mathcal{C}'_j$, and the input queue obtained by extending $\mathcal{E}'_j$ - the queue resulting from removing e from $\mathcal{E}$ - with the events produced in the step which are addressed to H_j. Notice that the elements of $\mathcal{U}$ are *packets* and that function *ext* will take care of unpacking the message-body from each relevant packet. For all $i \neq j$, Loc'_i will be unchanged w.r.t. Loc_i except that the input queue of H_i will be updated with all events produced in the step which are addressed to H_i. In the definition of the operational semantics we use an auxiliary relation, namely

$$A \uparrow P :: \mathcal{C} \xrightarrow{e/\mathcal{U}}_L \mathcal{C}'$$

Such relation models the step-transitions of generic HA A, and L is the set containing the transitions of A which are fired. We call the definition of the $\xrightarrow{e/\mathcal{U}}_L$ transition relation the *core* semantics of μCharts. 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, $A \uparrow P :: \mathcal{C} \xrightarrow{e/\mathcal{U}}_L \mathcal{C}'$ should be read as "A, on configuration $\mathcal{C}$, with input event e can perform L moving to configuration $\mathcal{C}'$, producing output $\mathcal{U}$, when required to fire transitions with priorities not smaller than that of any transition in P". Obviously, no restriction is made on the priorities for H in the definition of the Transition Relation (Fig. 9), but 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 deduction system for $\xrightarrow{e/\mathcal{U}}_L$ is shown in Fig. 10 where the following auxiliary functions are used:

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

$$E_A\ \mathcal{C}\ e \triangleq \bigcup_{A' \in (\mathcal{A}\ A)} LE_{A'}\ \mathcal{C}\ e$$

For generic HA A, $LE_A\ \mathcal{C}\ e$ is the set of those transitions in δ_A, i.e. *local* to the root of A, which are enabled in the current configuration $\mathcal{C}$ with input event e. In order for a transition $t \in \delta_A$ to be enabled it is required that all its source states, i.e. both its proper source state $SRC\ t$ and its source restriction $SR\ t$ are included in the current configuration $\mathcal{C}$; moreover, the specification of the trigger event $EV\ t$ must be equal to the input event; finally, the guard $G\ t$ must be satisfied[9].

[9] In this paper we do not deal with guard evaluation issues. They do not raise any particular interesting problem.

Function E_A extends LE_A in order to cover *all* the transitions of A including those of sub-automata of A. In the core semantics, the Progress Rule establishes that if there is a transition of A enabled

Progress rule

$$\frac{t \in LE_A\ \mathcal{C}\ e \quad (1) \qquad \nexists t' \in P \cup E_A\ \mathcal{C}\ e.\ \pi t \sqsubset \pi t' \quad (2)}{A \uparrow P :: \mathcal{C} \xrightarrow{e/new((DQ\ t),(AC\ t))}_{\{t\}} DST\ t}$$

Composition Rule

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

Stuttering Rule

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

Fig. 10. Core operational semantics rules for μCharts

and the priority of such a transition is "high enough" then the transition fires and a new status is reached accordingly. As an effect of firing transition t the event $(AC\ t)$ is sent to destination $(DQ\ t)$[10] . For transition t, $DST\ t$ is defined as the set $\{s \mid \exists s' \in (TD\ t).\ (TGT\ t) \preceq s \preceq s'\}$. The Composition Rule stipulates how automaton A delegates the execution of transitions to its sub-automata and these transitions are propagated upwards. 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 informal semantics of UML [11] [11]:

Theorem 2. *Given HA $H = (F, E, \rho)$ element of a μChart, for all $A \in F, L, \mathcal{C}, e, \mathcal{U}$ the following holds: $A \uparrow P :: \mathcal{C} \xrightarrow{e/\mathcal{U}}_{L} \mathcal{C}'$ for some $\mathcal{C}'$ 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. $L \subseteq E_A\ \mathcal{C}\ e$; (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.*
$\forall t \in L.\ \nexists t' \in E_A\ \mathcal{C}\ e.\ \pi t \sqsubset \pi t'$; and (iv) all transitions in L respect P, i.e. $\forall t \in L.\ \nexists t' \in P.\ \pi t \sqsubset \pi t'$.

We close this section with the definition of computations. We recall that in the UML official definition, the scheduler must dequeue events from the input queue of statecharts as long as such events are available. In other words, the scheduler is not allowed to stop or suspend a computation after a step occurred; it must dequeue the next event and pass it to the state machine. For modeling this behaviour, we define

[10] In [4] a set of destinations can be specified instead of a single one. Here we restrict the destinetion to a singleton for simplicity.

[11] Although we are basing all our work concerning UMLSDs on UML 1.3, the main features of the notation of interest for our work did not change in later versions.

Definition 11 (Computations). *A* computation *of μChart $S = (E, H_1, \ldots H_k)$ is a sequence of global states of S of the form $GS_0 \longrightarrow GS_1 \longrightarrow GS_2 \ldots GS_n \ldots$, where $GS_j \longrightarrow GS_{j+1}$ is obtained using the rule of Fig.9 and which is* maximal, *i.e. either it is infinite or it is finite with terminal element GS_n which has the property that $GS_n \longrightarrow GS'$ for no GS'.*

B Detailed Proofs

Lemma 3. *For all HA $H = (F, E, \rho)$ the following holds:*

i) For all $A \in F, \mathcal{C}, \mathcal{C}', \beta, \beta', P, \mathcal{E}, \mathcal{E}', \mathcal{U}, L, q, e$: if $q \in R_A\ \mathcal{C}\ \beta$ and $Sel\ \mathcal{E}\ q\ e\ \mathcal{E}'$ and for all $t \in PE_A\ \mathcal{C}\ \beta\ \mathcal{E}$ there exists $t' \in P$ such that $\pi t \sqsubset \pi t'$ and $A \uparrow P :: (\mathcal{C}, \beta) \overset{(q,e)/\mathcal{U}}{\longrightarrow}_L (\mathcal{C}', \beta')$ then $L = \emptyset$

ii) For all $A \in F, \mathcal{C}, \beta, P, \mathcal{E}$: if for all $q \in R_A\ \mathcal{C}\ \beta, e \in E, \mathcal{C}', \mathcal{E}', \mathcal{U}$ such that $Sel\ \mathcal{E}\ q\ e\ \mathcal{E}'$ we have $A \uparrow P :: (\mathcal{C}, \beta) \overset{(q,e)/\mathcal{U}}{\longrightarrow}_\emptyset (\mathcal{C}', \beta')$ then for all $t \in PE_A\ \mathcal{C}\ \beta\ \mathcal{E}$ there exists $t' \in P$ such that $\pi t \sqsubset \pi t'$

Proof.
Part (i), by contradiction:

$L \neq \emptyset \wedge A \uparrow P :: (\mathcal{C}, \beta) \overset{(q,e)/\mathcal{U}}{\longrightarrow}_L (\mathcal{C}', \beta') \wedge$

$q \in R_A\ \mathcal{C}\ \beta \wedge$

$Sel\ \mathcal{E}\ q\ e\ \mathcal{E}' \wedge$

$\forall t \in PE_A\ \mathcal{C}\ \beta\ \mathcal{E}.\ \exists t' \in P.\ \pi t \sqsubset \pi t'$

$\Rightarrow$ {Theorem 1}

$\exists t \in E_A\ \mathcal{C}\ \beta\ q\ e.\ \nexists t' \in P \cup E_A\ \mathcal{C}\ \beta\ q\ e.\ \pi t \sqsubset \pi t' \wedge$

$q \in R_A\ \mathcal{C}\ \beta \wedge$

$Sel\ \mathcal{E}\ q\ e\ \mathcal{E}' \wedge$

$\forall t \in PE_A\ \mathcal{C}\ \beta\ \mathcal{E}.\ \exists t' \in P.\ \pi t \sqsubset \pi t'$

$\Rightarrow$ {Def. of PE_A}

$\exists t \in PE_A\ \mathcal{C}\ \beta\ \mathcal{E}.\ \nexists t' \in P \cup E_A\ \mathcal{C}\ \beta\ q\ e.\ \pi t \sqsubset \pi t' \wedge$

$\forall t \in PE_A\ \mathcal{C}\ \beta\ \mathcal{E}.\ \exists t' \in P.\ \pi t \sqsubset \pi t'$

$\Rightarrow$ {Logics}

$FALSE$

Part (ii), by contradiction:

$\exists t \in PE_A\ \mathcal{C}\ \beta\ \mathcal{E}.\ \nexists t' \in P \cup E_A\ \mathcal{C}\ \beta\ q\ e.\ \pi t \sqsubset \pi t'$

$\Rightarrow$ {Def. of PE_A}

$\exists q \in R_A\ \mathcal{C}\ \beta, e \in E, \mathcal{E}'.\ Sel\ \mathcal{E}\ q\ e\ \mathcal{E}' \wedge t \in E_A\ \mathcal{C}\ \beta\ q\ e \wedge (\nexists t' \in P.\ \pi t \sqsubset \pi t'$

$\Rightarrow$ {Lemma 4}

$A \uparrow P :: (\mathcal{C}, \beta) \overset{(q,e)/\mathcal{U}}{\longrightarrow}_L (\mathcal{C}'', \beta'')$ for some $\mathcal{C}'', \beta'', \mathcal{U}$ and $L \neq \emptyset$

which contradicts the hypothesis

In the following lemmas we shall use the set of *Top Enabled Transitions*, i. e. the set of top elements of $E_A\ \mathcal{C}\ \beta\ q\ e$:

$$TopE_A\ \mathcal{C}\ \beta\ q\ e \triangleq \{t \in E_A\ \mathcal{C}\ \beta\ q\ e \mid \nexists t' \in E_A\ \mathcal{C}\ \beta\ q\ e.\ \pi t \sqsubset \pi t'\}$$

Lemma 4. *For all HA* $H = (F, E, \rho)$ $A \in F, \mathcal{C}, \beta, q, e, P, t$*: if* $t \in E_A\ \mathcal{C}\ \beta\ q\ e$ *and there exists no* $t' \in P$ *such that* $\pi t \sqsubset \pi t'$ *then there exist* $\mathcal{C}', \beta', \mathcal{U}$ *and* L *such that* $L \neq \emptyset$ *and* $A \uparrow P :: (\mathcal{C}, \beta) \xrightarrow{(q,e)/\mathcal{U}}_L (\mathcal{C}', \beta')$

Proof.

$t \in E_A\ \mathcal{C}\ \beta\ q\ e \wedge\ \nexists t' \in P.\ \pi t \sqsubset \pi t'$

$\Rightarrow$ {Lemma 5}

$\exists \hat{t} \in TopE_A\ \mathcal{C}\ \beta\ q\ e.\ \ \nexists t' \in P.\ \pi \hat{t} \sqsubset \pi t'$

$\Rightarrow$ {Lemma 6}

$\exists \mathcal{C}', \beta', \mathcal{U}, L.\ L \neq \emptyset \wedge A \uparrow P :: (\mathcal{C}, \beta) \xrightarrow{(q,e)/\mathcal{U}}_L (\mathcal{C}', \beta')$

Lemma 5. *For all HA* $H = (F, E, \rho)$ $A \in F, \mathcal{C}, \beta, q, e, P, t$*: if* $t \in E_A\ \mathcal{C}\ \beta\ q\ e$ *and there exists no* $t' \in P$ *such that* $\pi t \sqsubset \pi t'$ *then there exists* $\hat{t} \in TopE_A\ \mathcal{C}\ \beta\ q\ e$ *for which there exists no* $t' \in P$ *with* $\pi \hat{t} \sqsubset \pi t'$

Proof.
If there is no $\bar{t} \in E_A\ \mathcal{C}\ \beta\ q\ e$ such that $\pi t \sqsubset \pi \bar{t}$ then the assert holds with $\hat{t} = t$ by definition of $TopE_A$. Suppose instead that such a $\bar{t}$ exists and consider the following set $T \triangleq \{t' \in E_A\ \mathcal{C}\ \beta\ q\ e \mid \pi t \sqsubset \pi t' \wedge\ \nexists t'' \in E_A\ \mathcal{C}\ \beta\ q\ e.\ \pi t' \sqsubset \pi t''\}$. Notice that T is not empty since $\pi t \sqsubset \pi \bar{t}$ and there is a finite number of transitions in H. Moreover, $T \subseteq TopE_A\ \mathcal{C}\ \beta\ q\ e$. Now we show that for all $\hat{t} \in T$ there exists no $t' \in P$ with $\pi \hat{t} \sqsubset \pi t'$, which proves the assert. We proceed by contradiction. Suppose $\pi \hat{t} \sqsubset \pi t'$ for some $\hat{t} \in T$ and $t' \in P$. By definition of T, $\pi t \sqsubset \pi \hat{t}$ and this would bring to $\pi t \sqsubset \pi t'$ which contradicts the hypothesis.

Lemma 6. *For all HA* $H = (F, E, \rho)$ $A \in F, \mathcal{C}, \beta, q, e, P, t$*: if* $t \in TopE_A\ \mathcal{C}\ \beta\ q\ e$ *and there exists no* $t' \in P$ *such that* $\pi t \sqsubset \pi t'$ *then there exist* $\mathcal{C}', \beta', \mathcal{U}$ *and* L *such that* $L \neq \emptyset$ *and* $A \uparrow P :: (\mathcal{C}, \beta) \xrightarrow{(q,e)/\mathcal{U}}_L (\mathcal{C}', \beta')$

Proof. The proof is carried out by structural induction on the subset of F affected by $\mathcal{C}$. Let $F_{\mathcal{C}}$ be the set $\{A \in F \mid \mathcal{C} \cap \sigma_A \neq \emptyset\}$. It is easy to define a relation on $F_{\mathcal{C}}$ such that X is related to Y iff $s \in \mathcal{C} \cap \sigma_Y$ and $X \in (\rho_Y\, s)$. Notice that since $\mathcal{C}$ is a configuration, for each A in $F_{\mathcal{C}}$ there is a unique state $s \in \sigma_A \cap \mathcal{C}$. The transitive and reflexive closure of such a relation is a well-founded partial order, since antisymmetry is a consequence of property (iii) in the definition of HAs and the bottom elements are those X such that $\rho s = \emptyset$ for $s \in \mathcal{C} \cap \sigma_X$ [9]:

Base case ($\rho_A\ s = \emptyset$ where $\{s\} = \mathcal{C} \cap \sigma_A$):

$t \in TopE_A\ \mathcal{C}\ \beta\ q\ e \wedge\ \nexists t' \in P.\ \pi t \sqsubset \pi t'$

$\Rightarrow$ {Def. $TopE_A$; $\rho\ s = \emptyset$}

$t \in LE_A\ \mathcal{C}\ \beta\ q\ e \wedge\ \nexists t' \in P \cup E_A\ \mathcal{C}\ \beta\ q\ e.\ \pi t \sqsubset \pi t'$

$\Rightarrow$ {Progress Rule of the Core Semantics}

$A \uparrow P :: (\mathcal{C}, \beta) \xrightarrow{(q,e)/\mathcal{U}}_{\{t\}} (\mathcal{C}', \beta')$ for some $\mathcal{C}', \beta', \mathcal{U}$

Induction step ($\rho_A\ s = \{A_1, \ldots, A_n\}$where $\{s\} = \mathcal{C} \cap \sigma_A$):

If $t \in LE_A\ \mathcal{C}\ \beta\ q\ e$, then the proof proceeds as in the base case. Suppose $t \notin LE_A\ \mathcal{C}\ \beta\ q\ e$. By the definition of E_A and $TopE_A$ we know that $t \in E_{A_j}\ \mathcal{C}\ \beta\ q\ e$ for some $A_j \in \rho_A\ s$. Moreover, using lemma 7 and set theory we get $t \in TopE_{A_j}\ \mathcal{C}\ \beta\ q\ e$. We also know that there exists no $t' \in LE_A\ \mathcal{C}\ \beta\ q\ e$ such that $\pi t \sqsubset \pi t'$ since $LE_A\ \mathcal{C}\ \beta\ q\ e \subseteq E_A\ \mathcal{C}\ \beta\ q\ e$ and $t \in TopE_A\ \mathcal{C}\ \beta\ q\ e$. Thence there exists no $t' \in P \cup LE_A\ \mathcal{C}\ \beta\ q\ e$ such that $\pi t \sqsubset \pi t'$. We can now apply the Induction Hypothesis to get $A_j \uparrow P \cup LE_A\ \mathcal{C}\ \beta\ q\ e :: (\mathcal{C}, \beta) \xrightarrow{(q,e)/\mathcal{U}_j}_{L_j} (\mathcal{C}_j, \beta_j)$ for some $\mathcal{C}_j, \beta_j, \mathcal{U}_j$ and $L_j \neq \emptyset$. By using the Composition Rule of the Core Semantics - notice that also its premise (4) is satisfied - we get the assert.

Lemma 7. *For all HA* $H = (F, E, \rho)$ $A \in F, \mathcal{C}, \beta, q, e$*:*
$E_A\ \mathcal{C}\ \beta\ q\ e = LE_A\ \mathcal{C}\ \beta\ q\ e \cup (\bigcup_{A' \in (\rho\ \sigma_A)} E_{A'}\ \mathcal{C}\ \beta\ q\ e)$

Proof. The proof can easily be carried out in a similar way as the proof for Lemma 3 of [4], which can be found (as Lemma 10) in [6].