HAL
open science

Strongly Consistent Languages for Modelling Mobility

Juliana Bowles, Leila Kloul

To cite this version:

Juliana Bowles, Leila Kloul. Strongly Consistent Languages for Modelling Mobility. 2009. hal-00419934

HAL Id: hal-00419934

https://hal.science/hal-00419934v1

Preprint submitted on 25 Sep 2009

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Strongly Consistent Languages for Modelling Mobility *

Juliana Bowles† and Leïla Kloul‡

† University of St-Andrews, School of Computer Science
North Haugh, St-Andrews KY16 9SX
juliana@cs.st-andrews.ac.uk

‡ PRiSM, Université de Versailles
45, Avenue des Etats-Unis, 78000 Versailles, France
kle@prism.uvsq.fr

Abstract

A seamless approach suitable for both design and analysis of mobile and distributed software systems is a challenge. The object-based Unified Modeling Language (UML) is a popular medium for effective design of most systems. PEPA nets are a recent performance modelling technique which offers capabilities for capturing notions such as location, synchronisation and message passing, and are thus suited for performance modelling of mobile and distributed software. In this paper, we provide a new constructive approach that links both models by deriving a PEPA net which realises the same language (legal set of traces) as a given Interaction Overview Diagram (IOD) in UML2. We prove that the languages are *strongly consistent* (equivalent) by establishing the one-to-one correspondence between the traces of the models.

Keywords: UML 2 Interaction Diagrams, PEPA nets, Formal Transformation, Mobility, Performance Analysis

1 Introduction

The increasing complexity of mobile and distributed software systems requires a more careful and sophisticated design approach for successful implementation. The object-based Unified Modeling Language (UML) can describe the structural and behavioural aspects of these systems and is a popular medium for effective design. We are interested in performance modelling and analysis of mobile distributed software systems, and thus in the combination of UML-based design and formal modelling techniques for performance analysis of these systems. PEPA nets are our performance modelling technique of choice, combining the stochastic process algebra PEPA with coloured Petri nets. PEPA nets [4] have capabilities for capturing notions such as location, synchronisation and message passing, and are thus ideally suited for performance analysis of mobile and distributed software. Furthermore, there is an extensive suite of tools available for the process algebra PEPA The combination of UML with PEPA nets should, however, be completely seamless and transparent to software developers. In other words, a software designer models a system using UML2, and is able to analyse the models with no knowledge of the underlying performance technique.

*This work is supported by the CNRS/Royal Society Project PETMoDs (IKSLU85079)

In previous work [10], we have shown how to model mobility and performance information at the design level using UML 2 and PEPA nets. In particular, we introduced new notation at the UML level to be able to capture performance information. In UML, we use behavioural models, namely sequence diagrams and interaction overview diagrams (IODs), where sequence diagrams capture the behaviour of (mobile and static) objects at the locations in the system, and the IOD is used to model the overall distributed system and how mobile objects move between locations. The UML models are extended with a notion of activity, an action type with its corresponding rate, to express performance information.

In this paper, we present a formalisation of performance annotated IODs and IOD nodes taking into account complex behaviour within a node determined by several and possibly nested interaction fragments. We define the languages associated with IODs and PEPA nets, and present an algorithm to synthesise a PEPA net model from an IOD model. We further show how the algorithm guarantees that our languages are *strongly consistent*. In other words, the legal traces of an IOD have a one-to-one correspondence to the legal traces of the underlying PEPA net model. This is a crucial advantage as it guarantees the absence of implied (unspecified or unacceptable) behaviours that can be observed in the synthesised model. The absence of implied scenarios in our approach facilitates an accurate performance analysis on the given UML design models.

Structure of this paper: In the next section, we present the UML2 interaction diagrams and extended notation used to model mobile distributed systems. In section 3, we describe the performance modelling technique PEPA nets. In section 4, we give a detailed description of the formal model for IODs and IOD nodes. We introduce a notion of region, as a subset of events, to capture interaction fragments in IOD nodes. In this way we have a framework which allows us to generate the exact corresponding PEPA expressions for the locations of the net. Section 5 describes the languages (set of legal traces) associated with both models. Section 6 describes the synthesis algorithm and gives a proof for the equivalence of the languages. Finally, section 7 describes related work, and section 8 concludes the paper with our plans for future work.

2 Interaction diagrams in UML2.0

To model interactions, UML2.0 offers four kinds of diagrams: communication diagrams, sequence diagrams, timing diagrams and interaction overview diagrams. Here we are only interested in sequence diagrams and interaction overview diagrams.

2.1 Sequence diagrams

Sequence diagrams are the more commonly used diagrams for capturing inter-object behaviour. In UML2.0, a sequence diagram is enclosed in a frame and the five-sided box at the upper lefthand corner names the sequence diagram. Further, interactions can be structured using so-called interaction fragments. Each interaction fragment has at least one operator held in the five-sided box at the upper left corner of the fragment. There are several possible operators described below. Figure 1 shows an example of a sequence diagram using UML2.0 constructs.

The semantics of an interaction operator is described informally in the UML2.0 superstructure specification [14]. Below we give the meaning of some operators used in this paper:

sd names a sequence diagram.

ref references an interaction fragment which appears in a different diagram. This fragment is called an *interaction use*.

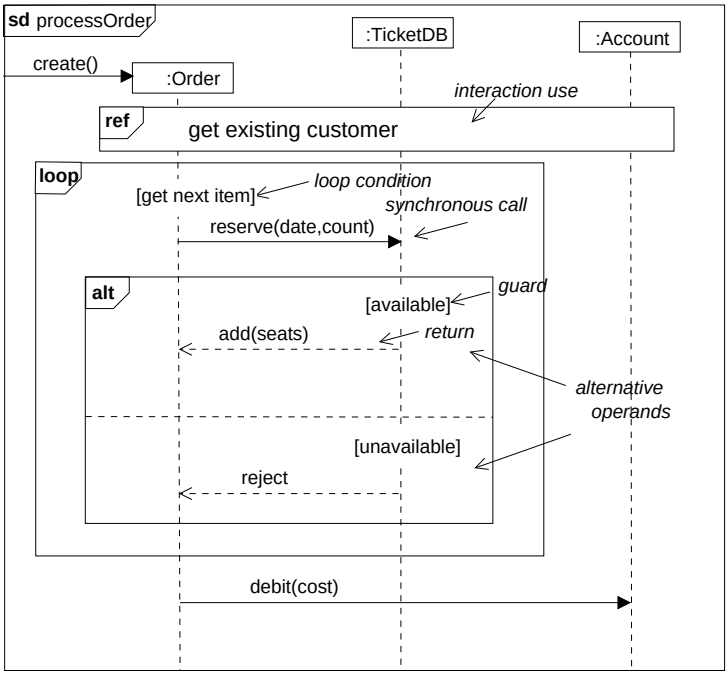

Figure 1: A sequence diagram.

alt designates that the fragment represents a choice of behaviour. At most one of the operands will execute. The operand that executes must have a guard expression that evaluates to true at this point in the interaction. If several guards are true, one of them is selected nondeterministically for execution.

par designates that the fragment represents a parallel merge between the behaviours of the operands. The event occurrences of the different operands can be interleaved in any way as long as the ordering imposed by each operand as such is preserved.

loop specifies an interaction fragment that shall be repeated some number of times. This may be indicated using a guard condition. The loop fragment is executed as long as the guard condition is true.

2.2 Interaction overview diagrams

IODs constitute a high-level structuring mechanism that is used to compose scenarios through sequence, iteration, concurrency or choice. IODs are similar to Hierarchical Message Sequence Charts (HMSCs), also known as Message Sequence Graphs (MSGs), which provide a structuring mechanism for MSCs [12].

IODs are a special and restricted kind of activity diagrams (ADs) in UML where nodes are interactions or interaction uses, and edges indicate the flow or order in which these interactions occur [14]. Semantically, however, IODs and ADs are given different interpretations. IODs follow, similarly to sequence diagrams, a trace semantics whereas ADs in UML2.0 are understood as Petri nets.

The notation used for IODs incorporates notation from sequence diagrams, essentially references (interaction uses) and sequence diagrams (inline interactions), with forks, joins, decision and merge

nodes from ADs. Branching and joining of branches in an IOD must be properly nested. The edges in an IOD denote control flow only and according to the UML specification [14] object flow cannot be represented.

Object flow in an AD is shown in Figure 2. The figure shows two alternative notations to denote the flow of an object of type Order from one node FillOrder to another node ShipOrder. In the first case the object is depicted in the middle of the edge, whereas in the second case the nodes have an output and input pin of type Order. After the node FillOrder has been completed a token of type Order is placed in the output pin of FillOrder. As soon as the edge fires the token moves to the input pin of ShipOrder.

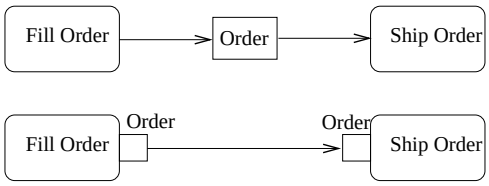

Figure 2: Object flow examples.

A node can have more than one object as in/output. In this case, there are several edges between the underlying nodes, one for each type of token, and the edges can be fired independently. However, whichever token reaches a target pin first will have to wait for the others before the final target node can be initiated. Unless otherwise indicated, all pins are required as input values before a node can be executed.

By default the number of tokens that are carried along an edge is one, but an input or output pin can collect several tokens of the same type. For instance in Figure 2, it may be the case that several orders have been filled (the node has been executed several times) and the corresponding tokens are placed in the output pin of FillOrder waiting for the edge to fire and the tokens (one at a time) to move to the input pin of ShipOrder. It is also possible that a pin can only accept a certain number of tokens. We write $\{upperBound = 50\}$ next to a pin to indicate that the maximum number of tokens that can be stored in that pin is 50. If the current number of tokens collected at the pin is 50 and the pin is an input pin, then no edge leading to that pin is allowed to fire. We assume that by default the value of the upperBound is one unless otherwise indicated.

Further, it is possible to have multiple edges leaving an output pin as shown in Figure 3. Notice that we cannot duplicate tokens on edges which means effectively that we have a case where the edges have to compete for a token.

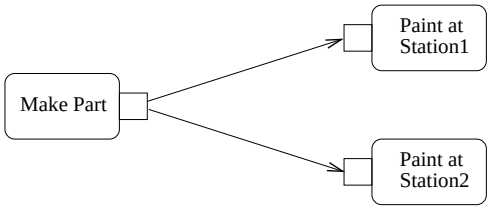

Figure 3: Nondeterministic choice.

In this example, we are modelling that after a part has been made it will (nondeterministically) be either painted at Station1 or at Station2. To avoid nondeterministic choice as in this example, it is common to use mutually exclusive guards on edges. As in ADs, edges in IODs can have guards (boolean

expressions) to determine whether the edge may or not enable the target node. Guards are written in square brackets on the edge.

Finally, in an AD a node can only start execution if all its input pins contain tokens as required, and after execution tokens are placed in all output pins. Sometimes, however, behaviour has alternative inputs or outputs. In other words, we may want to allow the node to execute with just a few inputs and produce only a subset of the possible outputs. To denote this we use a double box around pins as shown in

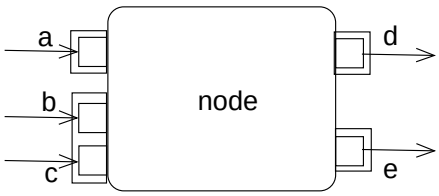

Figure 4: Alternative input and output pins.

Figure 4. Here, the activity node requires as input either one token of type `a`, or one token of type `b` and a token of type `c`. In one case it produces as output a token of type `d` or a token of type `e`. At this level we are not able to determine how input and output tokens are related. According to [16] it is also not clear what happens if both sets of inputs are available. Our interpretation here is that alternative input/output pins denote concurrent object flow and the node contains a concurrent execution for each of these inputs. This will become clearer later on with an example of an IOD.

Even though IODs only describe control flow and cannot show object flow and pins, the notion of object flow is implicitly present. A node in an IOD is a sequence diagram containing objects that can progress to a further interaction according to the edges at the IOD level. Moreover, from an IOD we can derive the expected traces of behaviour for each of the instances involved.

Take the example in Figure 5 showing an IOD with two inline interactions.

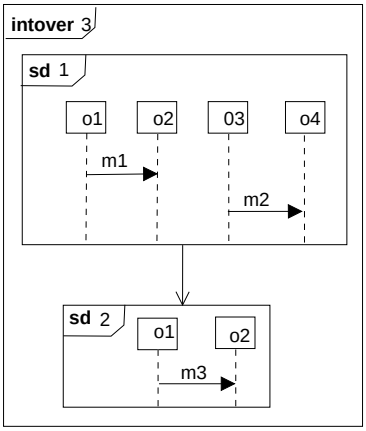

Figure 5: Simple IOD with two inline interactions.

The first interaction (`sd1`) shows object `o1` sending a message `m1` to object `o2` and independently object `o3` sending a message `m2` to object `o4`. The second interaction (`sd2`) shows just object `o1` sending a message `m3` to object `o2`. At the higher level, there are two ways of understanding the edge from `sd1` to `sd2` which correspond to the two possible interpretations of sequential composition of

interactions in an IOD. The first interpretation could be that interaction `sd1` has to complete before the behaviour described in interaction `sd2` can start. This is the typical interpretation of transitions in an AD and corresponds to the notion of *strong sequential composition*. However, it is not entirely justified in the case of IODs as will be made clear shortly.

A second and weaker interpretation could be that since only objects `o1` and `o2` are involved in the second interaction, these two objects can move from the first interaction to the second after completing their behaviour in the first interaction. In other words, it should be possible for `o1` and `o2` to proceed to the second interaction after `o1` and `o2` have synchronised on message `m1` independently of whether `o3` and `o4` have synchronised on message `m2` or not.

We are primarily interested in explicitly modelling the mobility of objects, and thus we borrow the notation of object flow and pins from activity diagrams as shown in Figure 6. In other words, we do not allow simple edges between nodes as depicted in Figure 5, and we always have to indicate pins on edges. Implicitly, this means that we assume the second interpretation (weak sequential composition) by default.

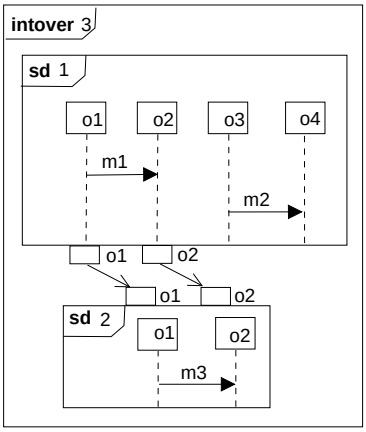

Figure 6: Independent object progression in an IOD.

We consider that all objects that want to progress from one interaction to another have an output pin with the name and type of the object, and an input pin with the same name and type in the following interaction. As soon as an object completes its behaviour as described in the first interaction, a token is placed in the corresponding output pin and the edge can fire provided the target pin has enough space. Whether or not the following interaction can execute depends on how many input tokens are required (recall also the case in Figure 4). In Figure 6, interaction `sd2` can only start executing once both tokens (one of type `o1` and one of type `o2`) are available in the respective input pins, but regardless of whether message `m2` has been sent or not.

In order to allow both interpretations of sequential composition, we can represent strong sequential composition using a fork as shown in Figure 7.

In this example, the edges for objects `o1` and `o2` cannot fire independently andare synchronised with the edges for objects `o3` and `o4`. Only when all tokens are available on the fork can execution proceed, with objects `o1` and `o2` moving to node `sd2` and objects `o3` and `o4` returning to node `sd1` (notice that we could have made them move to a different node if we wanted to avoid the repeated occurrence of message `m2`). In other words, a fork is used to synchronise the objects associated with the edges it cuts accross.

With both interpretations of sequential composition at an IOD level we obtain a powerful language

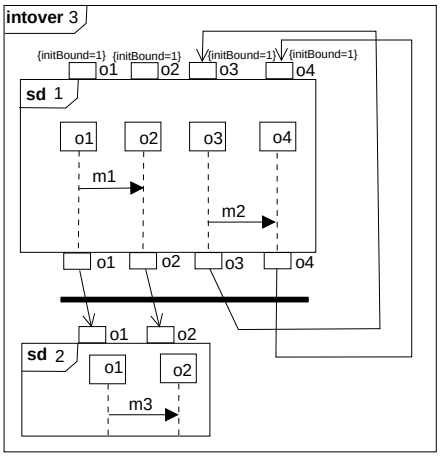

Figure 7: Strong sequential composition in an IOD.

to model and structure interactions. In particular, with our approach IODs can be used to describe interactions for mobile applications.

As shown in Figure 7, we use a tagged value $\{initBound = n\}$ which we write next to a pin to indicate the initial number of tokens n associated with that pin. If this tag is not given next to a pin then we are implicitly assuming $\{initBound = 0\}$. Using the tag $initBound$ simplifies our model as we do not have to indicate the initial node (and possible fork, or token constraints) of the IOD. This gives the initial marking of the IOD.

2.3 Performance Annotated Extension

In [10] we have shown how to use IODs and sequence diagrams for modelling mobility and performance information. In particular, we extended both diagrams to be able to add the performance information to the models. Here we only give a brief description of the added notation.

For modelling mobility through edges in an IOD, it is useful to be able to indicate, if intended, the explicit activity (an action type with its corresponding rate) that corresponds to the movement of an object from one node or location to another. We can indicate this additional activity at the source pin of an IOD edge. In the UML specification, a pin has a name and type (one or the other may be omitted). We assume here that a source pin of an edge carries the information on the associated activity by giving the corresponding *action type* and *rate*.

The textual label of a source pin is given by: `pin_type;action_type/rate` as shown in Figure 8.

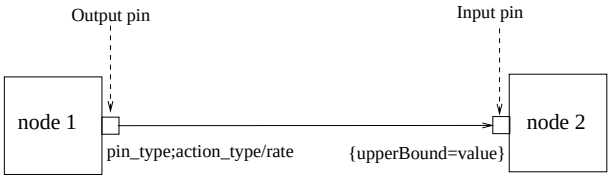

Figure 8: Input and output pins.

Similarly, the messages inside an IOD node (sequence diagram) are activities and represented by an

action type and one rate (denoting an individual object activity) or two rates (indicating synchronisation between objects).

3 PEPA nets

PEPA nets [4] combine the process algebra PEPA with stochastic coloured Petri nets. This hybrid formalism can be regarded as using the stochastic process algebra PEPA as the inscription language for labelled stochastic Petri nets. Viewed in another way, the net is used to provide a structure for combining related PEPA systems. In either view the combined modelling language naturally represents such applications as mobile code systems where the PEPA terms are used to model the program code which moves between network hosts (the places in the net).

In PEPA a system is described as an interaction of *components* which engage, either singly or multiply, in *activities*. These activities represent changes of state within a system. PEPA nets are motivated by the observation that in many systems we can identify two distinct types of change of state, as changes within the system may take place on different scales. Therefore there are two types of change of state in a PEPA net. We refer to these as *firings* of the net and as *transitions* of PEPA components. Firings of the net will typically be used to model macro-step (or *global*) changes of state such as context switches, breakdowns and repairs, one thread yielding to another, or a mobile software agent moving from one network host to another. Transitions of PEPA components will typically be used to model small-scale (or *local*) changes of state as components undertake activities.

In PEPA net, each activity has an *action type* and its duration is represented by a parameter of the associated exponential distribution: *activity rate*. This parameter may be any positive real number, or the distinguished symbol $\top$ (read as *unspecified*). Thus each activity, a, is a pair (α, r) consisting of the action type and the activity rate respectively. We assume a countable set of components, denoted $\mathcal{C}$, and a countable set, $\mathcal{Y}$, of all possible action types. We denote by $\mathcal{A}ct \subseteq \mathcal{Y} \times \mathbb{R}^+$, the set of activities, where $\mathbb{R}^+$ is the set of positive real numbers together with the symbol $\top$.

As the firings, on one hand, and the transitions, on the other hand, are special cases of PEPA activities, we differentiate the action types associated with each of these. We denote by $\mathcal{Y}_f$ the set of action types at the net level and by $\mathcal{Y}_t$ the set of action types inside the places such that $\mathcal{Y} = \mathcal{Y}_f \cup \mathcal{Y}_t$. Similarly, we denote by $\mathcal{A}ct_t \subseteq \mathcal{Y}_t \times \mathbb{R}^+$ the set of activities undertaken by the components inside the places and by $\mathcal{A}ct_f \subseteq \mathcal{Y}_f \times \mathbb{R}^+$ the set of activities at the net level such that $\mathcal{A}ct = \mathcal{A}ct_f \cup \mathcal{A}ct_t$.

A PEPA net is made up of PEPA *contexts*, one at each place in the net. A context consists of a number of *static* components (possibly zero) and a number of *cells* (at least one). Like a memory location in an imperative program, a cell is a storage area to be filled by a datum of a particular type. In particular in a PEPA net, a cell is a storage area dedicated to storing a PEPA component of the specified type. The components which fill cells are the *mobile* components and can circulate as the *tokens* of the net. In contrast, the static components cannot move.

The mobile components or tokens of a PEPA net are terms of the PEPA stochastic process algebra which define the behaviour of components via the activities they undertake and the interactions between them. Thus each token has a type given by its definition. This type determines the transitions and firings which a token can engage in; it also restricts the places in which it may be, since it may only enter a cell of the corresponding type.

We assume a countable set (possibly empty) of static components $\mathcal{C}_S$ and a countable set of mobile components or tokens $\mathcal{C}_M$ such that $\mathcal{C}_S \cup \mathcal{C}_M = \mathcal{C}$.

Definition 3.1 *A PEPA net $\mathcal{V}$ is a tuple $\mathcal{V} = (\mathcal{P}, \mathcal{T}, I, O, \ell, \pi, \mathcal{F}_P, K, M_0)$ such that*

- *$\mathcal{P}$ is a finite set of places;*
- *$\mathcal{T}$ is a finite set of net transitions;*
- *$I : \mathcal{T} \rightarrow \mathcal{P}$ is the input function;*
- *$O : \mathcal{T} \rightarrow \mathcal{P}$ is the output function;*
- *$\ell : \mathcal{T} \rightarrow (\mathcal{Y}_f, \mathbb{R}^+ \cup \{\top\})$ is the labelling function, which assigns a PEPA activity ((type, rate) pair) to each transition. The rate determines the negative exponential distribution governing the delay associated with the transition;*
- *$\pi : \mathcal{Y}_f \rightarrow \mathbb{N}$ is the priority function which assigns priorities (represented by natural numbers) to firing action types;*
- *$\mathcal{F}_P : \mathcal{P} \rightarrow P$ is the place definition function which assigns a PEPA context, containing at least one cell, to each place;*
- *K is the set of token component definitions;*
- *M_0 is the initial marking of the net.*

The syntax of PEPA nets is given in Figure 9. In that grammar S denotes a *sequential component* and P denotes a *concurrent component* which executes in parallel. I stands for a constant which denotes either a sequential or a concurrent component, as bound by a definition.

In PEPA, the behaviour of an expression is given by structured operational semantic rules [7]. These give rise to a labelled transition system which can be regarded as a derivation graph for the term: each syntactic form is a node of the graph and the possible activities give the arcs or transitions of the graph. For a component $C \in \mathcal{C}$, the set of reachable states, termed the *derivative set*, is denoted $ds(C)$. Regarding the graph as a state transition diagram gives rise to the CTMC underlying a PEPA expression (see [7] for more details). The CTMC can be solved to obtain a steady-state probability distribution from which performance measures can be derived.

Similarly, PEPA net behaviour is governed by structured operational semantic rules. These consist of the original rules for PEPA and some additional rules capturing the meaning of a cell, as well as the enabling and firing rules of the net level structure [4]. Now the states of the model are the marking vectors, which have one entry for each place of the PEPA net. As previously the semantic rules govern the possible evolution of a state, giving rise to a labelled transition system or derivation graph. Now nodes of the graph of the marking vectors and the activities (individual, shared or firing activities) give the arcs of the graph.

As in PEPA the conflicts may be solved by the race policy but it is also possible to assign different priorities to different Petri net transitions, giving some firings priority over others [4]. However in this paper we restrict consideration to PEPA nets in which all net level transitions have the same priority.

$$
\begin{array}{lcll}
 & & N \quad ::= \quad K^{+}M & \text{(net)} \\
 & & \text{(definitions and marking)} & \\
 & & & \\
M & ::= & (M_{\mathbf{P}}, \ldots) & \text{(marking)} \\
M_{\mathbf{P}} & ::= & \mathbf{P}[X, \ldots] & \text{(place marking)} \\
 & & \text{(marking vectors)} & \\
 & & & \\
K & ::= & I \stackrel{def}{=} S & \text{(component defn)} \\
 & | & \mathbf{P}[X] \stackrel{def}{=} P[X] & \text{(place defn)} \\
 & | & \mathbf{P}[X, \ldots] \stackrel{def}{=} P[X] \underset{L}{\bowtie} P & \text{(place defn)} \\
 & & \text{(identifier declarations)} & \\
 & & & \\
S & ::= & (\alpha, r).S & \text{(prefix)} \\
 & | & S + S & \text{(choice)} \\
 & | & I & \text{(identifier)} \\
 & & \text{(sequential components)} & \\
 & & & \\
P & ::= & P \underset{L}{\bowtie} P & \text{(cooperation)} \\
 & | & P/L & \text{(hiding)} \\
 & | & P[X] & \text{(cell)} \\
 & | & I & \text{(identifier)} \\
 & & \text{(concurrent components)} & \\
 & & & \\
X & ::= & \text{`_'} & \text{(empty)} \\
 & | & S & \text{(full)} \\
 & & \text{(cell term expressions)} &
\end{array}
$$

Figure 9: The syntax of PEPA nets

4 A Formal IOD Model

In this section we describe IODs and IOD nodes formally. The formal model is then used in the next section to define the language of an IOD.

Definition 4.1 *An IOD $\mathcal{D}$ is a tuple defined by*
$\mathcal{D} = (\mathcal{N}, \mathcal{S}, \mathcal{T}, \mathcal{P}, \mathcal{A}ct, \mathcal{L}_O, \mathcal{L}_I, \mathcal{F}, \mathcal{C}, \mathcal{B})$ *where*

- *$\mathcal{N}$ is a finite set of nodes;*
- *$\mathcal{S}$ is a finite set of fork nodes;*
- *$\mathcal{T}$ is a finite set of transitions;*
- *$\mathcal{P}$ is a set of pin types such that $\mathcal{P} = \mathcal{P}_I \cup \mathcal{P}_O$ and $\mathcal{P}_I \cap \mathcal{P}_O = \emptyset$, where $\mathcal{P}_I$ is the set of input pin types and $\mathcal{P}_O$ is the set of output pin types in $\mathcal{D}$;*

- *$\mathcal{A}ct$ is a set of activities such that $\mathcal{A}ct = \mathcal{A}ct_n \cup \mathcal{A}ct_p$ where $\mathcal{A}ct_n$ is the set of activities in the nodes and $\mathcal{A}ct_p$ is the set of activities at the IOD level. Each activity in $\mathcal{A}ct$ is a pair (a, r) consisting of an action type a and a rate $r \in \mathbb{R}^+ \cup \{\top\}$;*
- *$\mathcal{L}_O$: $\mathcal{T} \rightarrow \{\mathcal{P}_O, \mathcal{A}ct_p\}$ is a total labelling function which assigns an output pin type and an activity to the source pin of a transition;*
- *$\mathcal{L}_I$: $\mathcal{T} \rightarrow \{\mathcal{P}_I, \mathbb{N}^+\}$ is a total labelling function which assigns an input pin type and an upper bound to the target pin of a transition;*
- *$\mathcal{F}$: $\mathcal{T} \rightarrow \mathcal{N} \times \mathcal{N}$ is a total function which assigns a pair of nodes (a source node and a target node) to a transition;*
- *$\mathcal{C}$: $\mathcal{S} \rightarrow 2^{\mathcal{T}}$ is a total function which assigns a set of transitions to a fork node;*
- *$\mathcal{B} : \mathcal{P} \rightarrow \mathbb{N}$ is the initial marking of the IOD.*

An IOD $\mathcal{D}$ is described by a set of nodes $\mathcal{N}$ and edges $\mathcal{T}$, here called transitions, between the nodes. In general, IODs can have forks (to split the control flow and indicate parallelism), joins (to join the control flow), and decision points (to indicate guarded choice). We can model the behaviour of joins and decision points with our transitions (we omit details here), and they are thus not included in the definition. We only consider an additional set of fork nodes $\mathcal{S}$.

In order to capture object mobility, a transition in an IOD is associated with a unique object and indicates how it moves from one node to another. To indicate which object is associated with a transition we use a set of pin types $\mathcal{P}$ distinguishing between input pin types $\mathcal{P}_I$ and output pin types $\mathcal{P}_O$. We use a set of activities $\mathcal{A}ct_p$ to indicate the action and rate associated with the object move and thus to a transition. All transitions are associated with two pin types: one output pin type (the source pin of the transition) and one input pin type (the target pin of the transition). We use functions $\mathcal{L}_O$ and $\mathcal{L}_I$ to associate the specific pins to a transition. The source of the transition also carries the activity associated with the object move. The target of the transition also has a natural number indicating the number of tokens allowed in the target pin (given by the *upperBound* constraint in the IOD - see Figure 8). If the target pin has reached its maximum number of tokens the transition is not enabled. A fork node in $\mathcal{S}$, which acts as a synchronisation bar, cuts across several transitions to synchronise the objects associated with the transitions. The set of transitions affected by a fork is given by the function $\mathcal{C}$. Finally, the initial marking $\mathcal{B}$ of the IOD defines how many tokens are available at pin types. When a transition fires one token from the source pin type of the transition is removed and placed at the associated target pin type.

Take the example IOD of Figure 7. Formally, the IOD is given by the set of nodes $\mathcal{N} = \{sd1, sd2\}$, one fork node $\mathcal{S} = \{s1\}$, transitions $\mathcal{T} = \{t_1, t_2, t_3, t_4\}$, input pins $\mathcal{P}_I = \{o_{1isd1}, o_{2isd1}, o_{3isd1}, o_{4isd1}, o_{1isd2}, o_{2isd2}\}$, output pins $\mathcal{P}_O = \{o_{1osd1}, o_{2osd1}, o_{3osd1}, o_{4osd1}\}$, set of activities Act (not given as the example does not show activities), and for instance $\mathcal{L}_O(t_1) = (o_{1osd1}, act_{o_1})$, $\mathcal{L}_I(t_1) = (o_{1isd2}, 1)$, $\mathcal{F}(t_1) = (sd1, sd2)$, $\mathcal{C}(s1) = \mathcal{T}$, $\mathcal{B}(o_{1isd1}) = 1$, $\mathcal{B}(o_{1osd1}) = 0$, and so on. Notice that we encode in the pin information whether it is an input pin, the object associated and which node it belongs to (e.g., o_{1isd1} is the input pin for object `o1` in node `sd1`).

The IOD defines the overall behaviour of the system whereas each individual node (sequence diagram) in the IOD describes the behaviour of a location in the system. A node is defined as follows.

Definition 4.2 *A node $\mathcal{A}$ for an IOD $\mathcal{D}$ where $\mathcal{A} \in \mathcal{N}$ is a tuple $\mathcal{A} = (\mathcal{O}, \mathcal{E}, <, \mathcal{M}_A, \mathcal{T}_A, \mathcal{P}_A, \mu_{\mathcal{A}}, \mathcal{I}_A, \mathcal{U}_A)$ such that*

- *$\mathcal{O}$ is a finite set of object types such that $\mathcal{O} = \mathcal{O}_M \bigcup \mathcal{O}_S$ where $\mathcal{O}_M$ is the set of mobile object types and $\mathcal{O}_S$ is the set of static object types;*
- *$\mathcal{E}$ is a set of events such that:*
  - *$\mathcal{E} = \mathcal{E}_S \bigcup \mathcal{E}_R$ where $\mathcal{E}_S$ is the set of send events and $\mathcal{E}_R$ is the set of receive events,*
  - *$\mathcal{E} = \bigcup_{o \in \mathcal{O}} \mathcal{E}_o$ such that for any $o_1, o_2 \in \mathcal{O}$, if $o_1 \neq o_2$ then $\mathcal{E}_{o_1} \bigcap \mathcal{E}_{o_2} = \emptyset$,*
- *$<$ is a set of partial orders $<_o \subseteq \mathcal{E}_o \times \mathcal{E}_o$ with $o \in \mathcal{O}$;*
- *$\mathcal{M}_A$ is a finite set of local labels (messages). Each label $m \in \mathcal{M}_A$ is defined as $m = a/r_1; r_2$ where $(a, r_1) \in \mathcal{A}ct_n$ and $(a, r_2) \in \mathcal{A}ct_n$.*
- *$\mathcal{T}_\mathcal{A}$ is the set of local transitions such as $\mathcal{T}_\mathcal{A} \subseteq \mathcal{E}_S \times \mathcal{M}_A \times \mathcal{E}_R$;*
- *$\mathcal{P}_\mathcal{A}$ is the set of pin types of $\mathcal{A}$ such that $\mathcal{P}_\mathcal{A} \subseteq \mathcal{P}$;*
- *$\mu_\mathcal{A} : \mathcal{P}_\mathcal{A} \rightarrow \mathcal{O}_M$ is a total function which associates a mobile object type with a pin type;*
- *$\mathcal{I}_\mathcal{A}$ is the set of inputs to $\mathcal{A}$ such that each input $I \in \mathcal{I}_\mathcal{A}$ is a set of pairs $\{(p, n)/p \in \mathcal{P}_{I_\mathcal{A}}, n \in \mathbb{N}^+\}$ where $\mathcal{P}_{I_\mathcal{A}}$ is the set of input pin types to $\mathcal{A}$ and n is a number of tokens.*
- *$\mathcal{U}_\mathcal{A}$ is the set of ouputs from $\mathcal{A}$ such that each output $U \in \mathcal{U}_\mathcal{A}$ is a set of pairs $\{(p, n)/p \in \mathcal{P}_{O_\mathcal{A}}, n \in \mathbb{N}^+\}$ where $\mathcal{P}_{O_\mathcal{A}}$ is the set of output pin types of $\mathcal{A}$ and n is a number of tokens.*

A node $\mathcal{A}$ in an IOD is a sequence diagram describing an interaction between objects $\mathcal{O}$. Some of the objects enter/leave the node through input/output pins and are the *mobile objects* given by the set $\mathcal{O}_M$ (the exact mapping of pin types to object types is given by the total function $\mu_\mathcal{A}$). Additional objects involved in the interaction described by the diagram are *static* and given by the set $\mathcal{O}_S$. Static objects reside in an IOD node and do not participate in any other interaction (node) elsewhere in the IOD. The behaviour of the node is described by a set of events $\mathcal{E}$ corresponding to the sending and receiving of messages ($\mathcal{E}_S$ and $\mathcal{E}_R$ respectively). Each event is associated with one unique object. A partial order $<$ is defined over the set of events and based on the local partial orders, i.e., the partial orders defined over the events of an object. Given a set of events and message labels, transitions in the node correspond to triples of the form (e_1, m, e_2) whereby e_1 is an event associated with the sending of message m and e_2 corresponds to the receipt of the same message.

Each message m consists of an action type a, and two rates r_1 and r_2. If one of the rate is unspecified, that is $r_1 = \top$ or $r_2 = \top$, then the rate is omitted leading to a message of the form $m = a/r$ where $r = r_1$ or $r = r_2$. In this case the rate is associated with the object sending the message. Note that, at least one rate must be specified giving the frequency at which the activity is to be performed.

An IOD node $\mathcal{A}$ has a set of pin types $\mathcal{P}_\mathcal{A}$ which is a subset of the pin types of the IOD, and as such consists of a disjoint set of input and output pin types.

For a node to execute, it needs to have a set of tokens available at its input pins. This is given by $\mathcal{I}_\mathcal{A}$. In particular, a node can have *alternative* inputs and $\mathcal{I}_\mathcal{A}$ is a family of sets of inputs to the node. For example, $\mathcal{I}_\mathcal{A} = \{\{(p_1, 1), (p_2, 2)\}, \{(p_3, 1)\}\}$ indicates that node $\mathcal{A}$ has three input pin types p_1, p_2 and p_3, but p_1, p_2 are an alternative input to p_3. Further, for the node to execute, we need one token of type p_1 and two tokens of type p_2 or alternatively one token of type p_3. Similarly, once a node has executed, it generates a set of tokens at its output pins. The outputs correspond to a family of sets of output pins $\mathcal{U}_\mathcal{A}$.

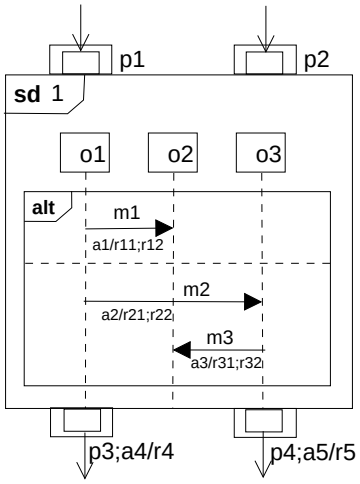

Figure 10: An IOD node.

Consider the node in Figure 10. Formally, node $sd1$ is given by objects $\mathcal{O}_M = \{o_1, o_3\}$ and $\mathcal{O}_S = \{o_2\}$, events $\mathcal{E} = \{e_1, \ldots e_6\}$ where local object events are $\mathcal{E}_{o_1} = \{e_1, e_3\}$, $\mathcal{E}_{o_2} = \{e_2, e_6\}$, $\mathcal{E}_{o_3} = \{e_4, e_5\}$ and the partial order is such that $e_4 <_{o_3} e_5$. The set of messages is given by $\mathcal{M}_{sd3} = \{m_1, m_2, m_3\}$, and the local transitions $\mathcal{T}_{sd3} = \{t_1, t_2, t_3\}$ are such that $t_1 = (e_1, m_1, e_2)$, $t_2 = (e_3, m_2, e_4)$ and $t_3 = (e_5, m_3, e_6)$. The pins to the node are $\mathcal{P}_{sd3} = \{p_1, p_2, p_3, p_4\}$ and $\mu_{sd3}(p_1) = \mu_{sd3}(p_3) = o_1$, $\mu_{sd3}(p_2) = \mu_{sd3}(p_4) = o_3$. The activities are such that for instance $(a_4, r_4) \in Act_p$ and $(a_1, r_{11}) \in Act_n$. Finally, there is one possible input and one possible output given by $\mathcal{I}_{sd3} = \{\{(p_1, 1), (p_2, 1)\}\}$ and $\mathcal{U}_{sd3} = \{\{(p_3, 1), (p_4, 1)\}\}$.

Since we can describe a variety of behaviour in a node using interaction fragments such as parallel behaviour, alternative behaviour, and loops, we need to capture the fragments associated with an IOD node. In what follows, we consider a *region* as a subset of events. We define a so-called *basic region* next.

Definition 4.3 *$\mathcal{R}$ is a basic region over $(\mathcal{E}, <)$ if $\mathcal{R} \subseteq \mathcal{E}$ and $\mathcal{R} = \bigcup_{o \in \mathcal{O}} \mathcal{R}_o$, where $\mathcal{R}_o$ is a totally ordered set of events for object o. The minimal and the maximal events of $\mathcal{R}_o$ are denoted $first_{\mathcal{R}_o}$ and $last_{\mathcal{R}_o}$ respectively.*

A basic region is a subset of events where all events belonging to the same object are totally ordered and hence we can refer to the first and last events for that object. One example of a basic region is a node wihtout any interaction fragments in it. The nodes $sd1$ and $sd2$ from the example of Figure 7 are basic regions. Notice that according to our definition a basic region can be empty.

Definition 4.4 *Let $\mathcal{R}$ be a basic region over $(\mathcal{E}, <)$ defined over IOD node $\mathcal{A}$. $\mathcal{R}$ is closed iff for any $e \in \mathcal{R}$ if there is a local transition $t \in \mathcal{T}_{\mathcal{A}}$ with $t = (e, m, e^{'})$ or $t = (e^{'}, m, e)$ then $e^{'} \in \mathcal{R}$.*

A closed basic region does not cut across local transitions. In other words, if one event involved in a transition belongs to a basic region so does its corresponding send/receive event. In the sequel, we assume that all our basic regions are closed.

Once we include alternative or parallel behaviour, we are no longer able to characterise the events of an object as being totally ordered. This is the case in our previous example of Figure 10. Indeed, the

whole set of events $\mathcal{E} = \{e_1, \ldots e_6\}$ does not define a region as for instance $e_1 \not<_{o_1} e_3$. In this example, we have two basic regions which correspond to the operands of the alt fragment, namely $\mathcal{R}_1 = \{e_1, e_2\}$ and $\mathcal{R}_2 = \{e_3, e_4, e_5, e_6\}$.

Definition 4.5 *Let $\mathcal{R}$ be a basic region over $(\mathcal{E}, <)$ defined over IOD node $\mathcal{A}$. The associated set of local transitions for $\mathcal{R}$ is given by $\mathcal{T}_{\mathcal{A}_\mathcal{R}}$ and is such that for each $l \in \mathcal{T}_{\mathcal{A}_\mathcal{R}}$, $l = (e_1, m, e_2)$ with $e_1, e_2 \in \mathcal{R}$.*

The local transitions associated with the regions defined for our example are given by $\mathcal{T}_{sd3_{\mathcal{R}\,1}} = \{t_1\}$ and $\mathcal{T}_{sd3_{\mathcal{R}\,2}} = \{t_2, t_3\}$.

Definition 4.6 *Let $\mathcal{R}$ be a basic region over $(\mathcal{E}, <)$ defined over IOD node $\mathcal{A}$, and $t_1, t_2 \in \mathcal{T}_{\mathcal{A}_\mathcal{R}}$. t_1 preceedes t_2 in the set of local transitions (written $t_1 \ll t_2$) iff at least one of the following holds for $t_1 = (e_{11}, m_1, e_{12})$, $t_2 = (e_{21}, m_2, e_{22})$, and some $o \in \mathcal{O}$*

1. $e_{11} <_o e_{21}$, *where* $e_{11}, e_{21} \in \mathcal{E}_o$
2. $e_{12} <_o e_{22}$, *where* $e_{12}, e_{22} \in \mathcal{E}_o$
3. $e_{12} <_o e_{21}$, *where* $e_{12}, e_{21} \in \mathcal{E}_o$
4. $e_{11} <_o e_{22}$, *where* $e_{11}, e_{22} \in \mathcal{E}_o$

In other words the transitions share at least one object, and the associated events for each object are ordered. Back in our example, for basic region $\mathcal{R}_2$ with $t_2, t_3 \in \mathcal{T}_{sd3_{\mathcal{R}\,2}}$, $t_2 \ll t_3$ as $e_4 <_{o_3} e_5$.

Definition 4.7 *Let $\mathcal{R}$ be a basic region and $\mathcal{T}_{\mathcal{A}_\mathcal{R}}$ be the associated set of local transitions over $\mathcal{R}$. The concurrency level of $\mathcal{T}_{\mathcal{A}_\mathcal{R}}$ is l, if $\mathcal{T}_{\mathcal{A}_\mathcal{R}}$ contains l totally ordered subsets of transitions $\mathcal{T}_{\mathcal{A}_\mathcal{R}} = \mathcal{T}_{\mathcal{A}_{\mathcal{R}1}} \cup \ldots \cup \mathcal{T}_{\mathcal{A}_{\mathcal{R}l}}$ such that, for two arbitrary distinct transitions t_1 and t_2, $t_1 \in \mathcal{T}_{\mathcal{A}_{\mathcal{R}i}}$, $t_2 \in \mathcal{T}_{\mathcal{A}_{\mathcal{R}j}}$ and $i \neq j \in [1, \ldots, l]$, $t1 \not\ll t_2$ and $t_2 \not\ll t_1$.*

Notice that for a basic region $\mathcal{R}$ where $\mathcal{T}_{\mathcal{A}_\mathcal{R}}$ has concurrency level 1, $\mathcal{T}_{\mathcal{A}_\mathcal{R}}$ is a totally ordered set of local transitions. In our example, $\mathcal{T}_{sd3_{\mathcal{R}\,2}}$ is a totally ordered set of local transitions and has concurrency level 1. For the example of Figure 7, the basic region in node $sd1$ containing both transitions labelled m_1 and m_2 has concurrency level 2 as m_1 and m_2 are completely independent. We can thus consider two basic regions one associated with a set of transitions containing m_1 and the other associated with a set of transitions containing m_2 (both singletons and hence trivially totally ordered).

Definition 4.8 *For basic regions $\mathcal{R}_1$ and $\mathcal{R}_2$, $\mathcal{R}_1.\mathcal{R}_2$ is a basic region denoting the sequential composition of the regions satisfying $last_{\mathcal{R}_{1_o}} <_o first_{\mathcal{R}_{2_o}}$ for any $o \in \mathcal{O}$.*

The sequential composition of regions is as expected a way of ordering the events of the respective regions sequentially.

Definition 4.9 *$\mathcal{G}$ is a basic arbitrary region if :*

- *$\mathcal{G}$ is a basic alt region, that is $\mathcal{G} = \mathcal{R}_1 \cup \mathcal{R}_2 \cup \ldots \cup \mathcal{R}_N$, $N \in \mathbb{N}$, where each $\mathcal{R}_n$, $n = 1, \ldots, N$, is a basic region, or*
- *$\mathcal{G}$ is a basic loop region, that is $\mathcal{G} = \mathcal{R}^N$ where $\mathcal{R}$ is a basic region and N, $N \in \mathbb{N}$, is the loop index,or*

- *$\mathcal{G}$ is a basic par region, that is $\mathcal{G} = [\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_N]$, $N \in \mathbb{N}$, where each $\mathcal{R}_n$, $n = 1, \ldots, N$, is a basic region and the associated set of local transitions $\mathcal{T}_{\mathcal{A}_{\mathcal{R}_n}}$ is of concurrency level one.*

If an alternative fragment does not have further nesting it is called basic. A basic alternative fragment, called basic alt region, can be seen as a finite union $\mathcal{G}$ of regions where each region corresponds to one of the operands in the alternative fragment and these regions are basic. We have already seen this for our example of Figure 10. Similarly, a basic loop region and a basic par region are made from basic regions: one in the case of the loop where there is an iteration over that basic region, and as many as there are operands in the case of the par.

The careful reader may notice that if we have a basic region with concurrency level greater than 1 (say l) we can see it as a basic par fragment with l operands where each operand is a basic region of concurrency level 1 and given by one of the subsets of transitions. This is stated in the following lemma.

Lemma 4.1 *Let $\mathcal{R}$ be a basic region such that the associated set of local transitions $\mathcal{T}_{\mathcal{A}_{\mathcal{R}}}$ has concurrency level l. Then there is an equivalent basic par region $\mathcal{G}$ with l operands such that $\mathcal{G} = [\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_l]$ and where each $\mathcal{R}_p$, for $p \in [1, ..., l]$, is a basic region with associated set of local transitions $\mathcal{T}_{\mathcal{A}_{\mathcal{R}_p}}$ of concurrency level* 1.

The idea is that a basic region with concurrency level greater than one can always be replaced by a basic par region where the level of concurrency gives us the number of operands of the par region. The proof is straightforward and we omit it here.

Given the lemma and without loss of generality, from now on we only consider basic regions with associated set of local transitions of concurrency level 1 (hence totally ordered).

The next definition deals with more general alternative fragments where nesting is allowed but restricted to a finite number of times given by k.

Definition 4.10 *$\mathcal{G}_k$ is an alt region of level k with N operands, $k > 1$, $N \in \mathbb{N}^+$, if $\mathcal{G}_k = \bigcup_{n=1}^{N} \mathcal{R}_n$, where each $\mathcal{R}_n$ is either a basic region or, for at least one value n_1 of n, $\mathcal{R}_{n_1} = \mathcal{P}re.\mathcal{G}_{k-1}.\mathcal{P}ost$. Both $\mathcal{P}re$ and $\mathcal{P}ost$ are basic regions and $\mathcal{G}_{k-1}$ is an arbitrary region of level $k-1$. If $k = 1$ then $\mathcal{G}_1$ is a basic arbitrary region.*

If there is nesting in one of the operands of an alternative fragment, the operand is not basic and can be seen as the sequential composition of three regions given by $(\mathcal{P}re.\mathcal{P}.\mathcal{P}ost)$ where the first and the last are basic and $\mathcal{P}$ is again an interaction fragment of some kind (alternative, parallel, or loop).

Consider the example of Figure 11. This example describes an alt region of level $k = 3$ with two operands ($N = 2$). Thus we have $\mathcal{G}_3 = \mathcal{R}_1 \cup \mathcal{R}_2$ where $\mathcal{R}_1 = \mathcal{P}re.\mathcal{G}_2.\mathcal{P}ost$ and $\mathcal{R}_2$ is a basic region with associated set of local transitions $\mathcal{T}_{\mathcal{A}_{\mathcal{R}_2}} = \{m_5, m_6\}$. In $\mathcal{R}_1$, $\mathcal{G}_2 = \mathcal{R}'_1 \cup \mathcal{R}'_2$, and both $\mathcal{P}re$ and $\mathcal{P}ost$ are empty. At the second level of nesting $\mathcal{R}'_1$ is a basic region with associated set of local transitions $\mathcal{T}_{\mathcal{A}_{\mathcal{R}'_1}} = \{m_1, m_2\}$ whereas $\mathcal{R}'_2 = \mathcal{P}re'.\mathcal{G}_1.\mathcal{P}ost'$. Both $\mathcal{P}re'$ and $\mathcal{P}ost'$ are empty and $\mathcal{G}_1$ is a basic alt region. Finally, $\mathcal{G}_1 = \mathcal{R}''_1 \cup \mathcal{R}''_2$ where $\mathcal{R}''_1$ and $\mathcal{R}''_2$ are basic regions, and $\mathcal{T}_{\mathcal{A}_{\mathcal{R}''_1}} = \{m_3\}$ and $\mathcal{T}_{\mathcal{A}_{\mathcal{R}''_2}} = \{m_4\}$.

Definition 4.11 *$\mathcal{G}$ is a loop region if $\mathcal{G} = \mathcal{P}re.(\mathcal{R})^n.\mathcal{P}ost$ where n, $n \in \mathbb{N}$, is the loop index, $\mathcal{P}re$ and $\mathcal{P}ost$ are basic regions and $\mathcal{R}$ is an arbitrary region.*

Notice that an arbitrary region can be any possible region, that is, a basic region, an alt region of some level or a par region of some level as defined next.

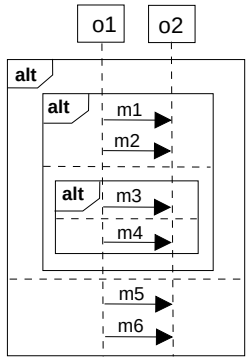

Figure 11: Alt region of level $k = 3$.

Definition 4.12 *$\mathcal{G}_k$ is a par region of level k with N operands, $k > 1$, $N \in \mathbb{N}^+$, if $\mathcal{G}_k = [\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \ldots, \mathcal{R}_N]$ where $\mathcal{R}_n$, $n = 1, \ldots, N$, is either a basic region or, for at least one value n_1 of n, $\mathcal{R}_{n_1} = [\mathcal{P}re.\mathcal{G}_{k-1}.\mathcal{P}ost]$. Both $\mathcal{P}re$ and $\mathcal{P}ost$ are basic regions and $\mathcal{G}_{k-1}$ is an arbitrary region of level $k-1$. If $k = 1$ then $\mathcal{G}_1$ is a basic arbitrary region.*

Consider the example of Figure 12. This example describes a par region of level $k = 2$ with two operands ($N = 2$). Thus we have $\mathcal{G}_2 = [\mathcal{R}_1, \mathcal{R}_2]$ where $\mathcal{R}_1 = [\mathcal{P}re.\mathcal{G}_1.\mathcal{P}ost]$ and $\mathcal{R}_2 = [\mathcal{P}re'.\mathcal{G}'_1.\mathcal{P}ost']$. $\mathcal{P}re$ is a basic region with one message m_0 whereas $\mathcal{P}ost$, $\mathcal{P}re'$ and $\mathcal{P}ost'$ are empty basic regions. Both $\mathcal{G}_1$ and $\mathcal{G}'_1$ are basic par regions.

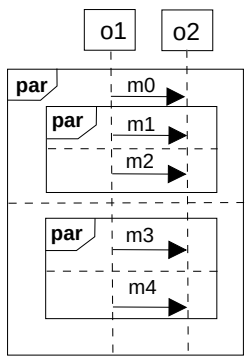

Figure 12: Par region of level $k = 2$.

Figure 13 describes an example of nested alt and par regions of level $k = 3$ with two operands ($N = 2$). As the region at the third level is an alt region we have $\mathcal{G}_3 = \mathcal{R}_1 \cup \mathcal{R}_2$ where $\mathcal{R}_1$ is a par region of level 2 and thus $\mathcal{R}_1 = [\mathcal{R}'_1, \mathcal{R}'_2]$ and $\mathcal{R}_2$ is a basic region with associated set of local transitions $\mathcal{T}_{\mathcal{A}_{\mathcal{R}_2}} = \{m_4, m_5\}$. $\mathcal{R}'_1$ is a basic region with $\mathcal{T}_{\mathcal{A}_{\mathcal{R}'_1}} = \{m_1\}$ and $\mathcal{R}'_2 = [\mathcal{P}re.\mathcal{G}_1.\mathcal{P}ost]$ where $\mathcal{P}re$ and $\mathcal{P}ost$ are empty and $\mathcal{G}_1$ is a basic par region such that $\mathcal{G}_1 = [\mathcal{R}''_1, \mathcal{R}''_2]$ with $\mathcal{T}_{\mathcal{A}_{\mathcal{R}''_1}} = \{m_2\}$ and $\mathcal{T}_{\mathcal{A}_{\mathcal{R}''_2}} = \{m_3\}$.

Given the framework described above to deal with interaction fragments, we are now able to define an IOD node fragment specification. Assume given a set of interaction fragment operators Ω such that $par, alt, loop \in \Omega$.

Definition 4.13 *An IOD node fragment specification for $\mathcal{A}$ is given by $Spec_{\mathcal{A}} = (Int_{\mathcal{A}}, f_{\mathcal{A}}, g_{\mathcal{A}})$ where*

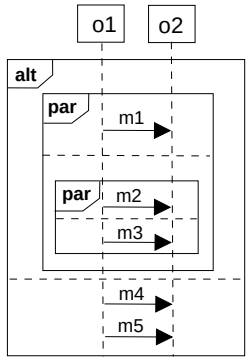

Figure 13: Nested alt and par region of level $k = 3$.

- *$Int_{\mathcal{A}}$ is a set of interaction fragment identifiers in node $\mathcal{A}$;*
- *$f_{\mathcal{A}} : Int_{\mathcal{A}} \rightarrow \Omega \times \mathbb{N} \times \mathbb{N}$ is a total function that assigns a triple (o, l, N) to an interaction fragment identifier where o is an operator, l a natural number indicating the level of the fragment and N the number of operands.*
- *$g_{\mathcal{A}} : Int_{\mathcal{A}} \times \Omega \times \mathbb{N} \times \mathbb{N} \rightarrow 2^{\mathcal{E}}$ is an injective function that takes a tuple of the form (i, o, l, N) and if $f_{\mathcal{A}}(i) = (o, l, N)$ then returns an o region of level l with N operands, otherwise it is undefined.*

5 Languages

In the following, we first define the associated language with interaction overview diagrams and then the language associated with PEPA nets.

5.1 The language of an IOD

Given the formal model of an IOD as given above, we now define its associated language. $\mathcal{L}(D)$ where $\mathcal{D}$ is an IOD, corresponds to the legal set of traces of $\mathcal{D}$. The traces are defined by the ordering of the events in the IOD nodes and respecting the ordering given by the transitions at the IOD level. In other words, a trace of an IOD is given by the union of the traces for all tokens in the IOD.

Definition 5.1 *A trace of IOD node $\mathcal{A} = (\mathcal{O}, \mathcal{E}, <, \mathcal{M}_A, \mathcal{T}_A,$ $\mathcal{P}_A, \mathcal{I}_A, \mathcal{U}_A)$ is a (possibly infinite) word $w = c_1.c_2 \ldots$ over the alphabet $\mathcal{M}_A$ iff there is a sequence of local transitions $t_1.t_2 \ldots$ over $\mathcal{T}_A$, such that $t_1 \ll t_2 \ll \ldots$, $t_i = (e_{si}, a_i/r_{i1}; r_{i2}, e_{ri})$ and $c_i = (a_i, min(r_{i1}, r_{i2}))$ for $0 < i \leq |w|$, $e_{si} \in \mathcal{E}_S$ and $e_{ri} \in \mathcal{E}_R$.*

We define L_1 as the IOD alphabet such that $L_1 = \mathcal{A}ct_p \cup \mathcal{A}ct_t$.

Definition 5.2 *A trace of IOD $\mathcal{D} = (\mathcal{N}, \mathcal{S}, \mathcal{T}, \mathcal{P}, Act, \mathcal{L}_O,$ $\mathcal{L}_I, \mathcal{F}, \mathcal{C}, \mathcal{B})$ is a (possibly infinite) word $W = w_1.c_1.w_2.c_2 \ldots$ over the alphabet L_1 iff there is a sequence of transitions $t_1.t_2 \ldots$ over $\mathcal{T}$ such that, for $0 < i \leq |W|$,*

- *w_i is a trace of IOD node $\mathcal{A}_i$,*
- *$\mathcal{L}_O(t_i) = (p_i, c_i)$ where $p_i \in \mathcal{P}_O$ and $c_i = (a_i, r_i) \in Act_p$,*

- $\mathcal{F}(t_i) = (\mathcal{A}_i, \mathcal{A}_{i+1})$ *where* $\mathcal{A}_i, \mathcal{A}_{i+1} \in \mathcal{N}$*, and*
- $t_1 \in \mathcal{T}_\mathcal{B}$ *where* $\mathcal{T}_\mathcal{B}$ *is the set of possible initial transitions obtained from the inital marking* $\mathcal{B}$*.*

Definition 5.3 *Let a maximal trace be a trace which is not a proper prefix of any other trace. The language of IOD* $\mathcal{D}$ *is the set* $L_1(\mathcal{D})$ *of words over the alphabet* L_1 *where* $L_1(\mathcal{D}) = \{W \mid W$ *is a maximal trace of* $\mathcal{D}\}$*.*

5.2 The language of a PEPA net

Let V be the labelled transition system or derivation graph of a place $P \in \mathcal{P}$ and let $\mathcal{T}_V$ be the set of all transitions in that graph. We define h as the labelling function which assigns a PEPA activity to each transition in $\mathcal{T}_V$.

Definition 5.4 *Let* $t_1, t_2 \in \mathcal{T}_V$*.* t_1 *preceedes* t_2 *in the set of transitions (written* $t_1 \ll t_2$*) iff there is a sequence of activities* $h(t_1).h(t_2)$ *where* $h(t_1) = (a_1, r_1)$ *and* $h(t_2) = (a_2, r_2)$*,* $r_1, r_2 \in \mathbb{R}^+ \cup \{\top\}$*.*

In order to define the language of a PEPA net $\mathcal{V}$, we first define the trace of a PEPA net place $P \in \mathcal{P}$ as follows.

Definition 5.5 *A trace of a PEPA net place* P *is a (possibly infinite) word* $w = c_1.c_2\ldots$ *over the alphabet* $\mathcal{A}ct_t$ *iff there is a sequence of transitions* $t_1, t_2, \ldots$ *over* $\mathcal{T}_V$ *such that, for* $0 < i \leq |w|$*,* $t_1 \ll t_2 \ll \ldots$ *and* $c_i = h(t_i) = (a_i, r_i)$ *where* c_i *is either:*

- *an individual activity, or*
- *a shared activity, between two components* C_1 *and* C_2*, which rate is* $r_i = min(r_{i1}, r_{i2})$ *where* r_{i1} *and* r_{i2} *are the rates of the activity in components* C_1 *and* C_2 *respectively.*

We define L_2 as the PEPA net alphabet such that $L_2 = \mathcal{A}ct_t \cup \mathcal{A}ct_f$. Using the definition of the trace w_i of each place $P_i \in \mathcal{P}$ in the net, the trace of a PEPA net $\mathcal{V}$ is defined as follows.

Definition 5.6 *A trace of a PEPA net* $\mathcal{V} = (\mathcal{P}, \mathcal{T}, I, O, \ell, \pi,$
$\mathcal{C}, K, M_0)$ *is a (possibly infinite) word* $W = w_1.c_1.w_2.c_2\ldots$ *over the alphabet* L_2 *iff there is a sequence of transitions* $t_1.t_2\ldots$ *over* $\mathcal{T}_f$ *such that, for* $0 < i \leq |W|$*,*

- w_i *is a trace of the PEPA net place* $P_i \in \mathcal{P}$*,*
- $\mathcal{O}(t_i) = P_i$*,*
- $\mathcal{I}(t_i) = P_i'$ *where* $P_i' \in \mathcal{P}$*,*
- $c_i = l(t_i) = (a_i, r_i)$ *where* $c_i \in \mathcal{A}ct_f$*, and*
- $t_1 \in \mathcal{T}_{\mathcal{M}_0}$ *where* $\mathcal{T}_{\mathcal{M}_0}$ *is the set of possible initial transitions obtained from the inital marking* $\mathcal{M}_0$*.*

Now, we define the language of a PEPA net $\mathcal{V}$, noted $L_2(\mathcal{V})$, as follows:

Definition 5.7 *Let a maximal trace be a trace which is not a proper prefix of any other trace. The language of the PEPA net* $\mathcal{V}$ *is the set* $L_2(\mathcal{V})$ *of words over the alphabet* L_2 *such that* $L_2(\mathcal{V}) = \{W \mid W$ *is a maximal trace of* $\mathcal{V}\}$*.*

6 The Transformation

In this section, we describe the algorithm behind the IOD-to-PEPA net model transformation and prove that the algorithm is correct by proving the equivalence between the language of an IOD and the one of a PEPA net, that is, the language for an IOD $\mathcal{D}$ given by $L_1(\mathcal{D})$ is equivalent to the language for a PEPA net $\mathcal{V}$ given by $L_2(\mathcal{V})$.

6.1 The Algorithm

We can build a direct correspondence between the IOD nodes and the objects in the UML model, with, respectively, the places and the components in the PEPA net model. Both models use activities and there is a one-to-one correspondence between activities in IOD edges or IOD node messages, and PEPA net firing transitions or PEPA transitions respectively.

In other words, an IOD can be viewed as a PEPA net model where each IOD node corresponds to a place in the PEPA net. An edge or transition between two IOD nodes is transformed into a firing transition between two places in the net with the same label. Table 6.1 describes the correspondence between the elements of an IOD and those of a PEPA net, in accordance with our definition of an IOD (definition 4.1), an IOD node (definition 4.2) and a PEPA net (definition 3.1).

IODs	**PEPA nets**
IOD $\mathcal{D}$ (def. 4.1)	PEPA net $\mathcal{V}$ (def. 3.1)
IOD node $\mathcal{A} \in \mathcal{N}$	Place $P \in \mathcal{P}$
IOD transition $t \in \mathcal{T}$	Firing transition $t \in \mathcal{T}_f$
IOD activity $c \in Act_p$	Firing activity $c \in Act_f$
IOD node local transition $t \in \mathcal{T}_\mathcal{A}$	Transition $t \in \mathcal{T}_t$
Static object $O \in \mathcal{O}_S$	Static component $C \in \mathcal{C}_S$
Mobile object, token $O' \in \mathcal{O}_M$	PEPA net token $C' \in \mathcal{C}_M$
IOD node activity $c \in Act_n$	PEPA activity $c \in Act_t$
Set of inputs to IOD node $\mathcal{A}$ $(p, n) \in \mathcal{I}_A$	Number of cells n in place P for corresponding token
IOD fork node $s \in \mathcal{S}$	PEPA component synchronisation in the source place

Table 1: Translation of IOD elements into PEPA net elements

A static object inside an IOD node ($O \in \mathcal{O}_S$) corresponds to a static PEPA component ($C \in \mathcal{C}_S$). In UML, an object is defined by its name and its type with the following syntax: `name:type` where the name of an object is optional. Both in the formal IOD model and the PEPA net model we only consider the type of the object. Inside an IOD node, the behaviour of a static object is described by a sequence diagram. From this diagram, we can derive the complete behaviour of the corresponding PEPA static component.

A mobile object or UML token $O' \in \mathcal{O}_M$ is translated into a PEPA net token $C' \in \mathcal{C}_M$. The behaviour of the mobile component C' can be derived from both the sequence diagram inside each IOD node object O' visits and the information on the pins of these IOD nodes. The information on a pin is translated in the PEPA net model as the activity $(action_type, rate)$ of the firing transition between the places representing the nodes. Moreover, this activity is added to component C' behaviour

as $(action_type, \top)$ showing that the rate of this activity will be specified when the net transition with label $(action_type, rate)$ is fired.

The local activity (a, r) to a PEPA component is the translation of a message a/r on the sequence diagram that the corresponding UML object sends to itself. A cooperation activity between two PEPA components C_1 and C_2 in a place $P \in \mathcal{P}$ is the translation of a message, in the sequence diagram of IOD node $\mathcal{A} \in \mathcal{N}$, that an object of type O_1 sends to an object of type O_2. This message, which is noted $b/r_1; r_2$, consists of the action type b, and two rates r_1 and r_2. This action type will be the one on which both PEPA components C_1 and C_2 will have to cooperate with rates r_1 and r_2 respectively.

We can distinguish between an active component and a passive one by considering which corresponding object sends the message as follows:

- If an object O_1 sends a message of the form b/r_1 to another object O_2, then this message is equivalent to $b/r_1; \top$ and that means that, in the context of the PEPA net, C_1 is an active component for action type b whereas C_2 is a passive one.

- Similarly if an object O_1 receives a message of the form b/r_2 from an object O_2, then O_1 should be translated as a passive component regarding action type b. Indeed this form of message is equivalent to $b/\top; r_2$.

In the following, we show how we translate an alt region, a par region and a loop region given definitions 4.10, 4.12 and 4.11 respectively.

The behaviour of an object O in an alt region of level k with N operands $\mathcal{G}_k = \bigcup_{n=1}^{N} \mathcal{R}_n$ can be translated into a PEPA component behaviour C_k such that $C_k \stackrel{def}{=} C_{k,1} + C_{k,2} + \ldots + C_{k,N}$. If $\mathcal{R}_n$ is a basic region which associated set of local transitions $\mathcal{T}_{\mathcal{A}_{\mathcal{R}_n}}$ is of size Z, then its corresponding derivative $C_{k,n}$ is defined as $C_{k,n} \stackrel{def}{=} (a_1, r_1).(a_2, r_2).\ldots.(a_Z, r_Z)$ where a_i and r_i are respectively the action type and one of the rates in message $m_i = a_i/r_{i_1}; r_{i_2}$, $m_i \in \mathcal{T}_{\mathcal{A}_{\mathcal{R}_n}}$ and $i = 1, \ldots, Z$. $r_i = r_{i_1}$ if O is the sender of m_i and $r_i = r_{i_2}$ if O is the receiver. Now, if $\mathcal{R}_n$ is not a basic region, that is $\mathcal{R}_n = \mathcal{P}re.\mathcal{G}_{k-1}.\mathcal{P}ost$ then $C_{k,n} \stackrel{def}{=} Q_1.C_{k-1}.Q_2$ where $Q_1 \stackrel{def}{=} (a_1, r_1).\ldots.(a_{Z_1}, r_{Z_1})$ and $Q_2 \stackrel{def}{=} (a'_1, r'_1).\ldots.(a'_{Z_2}, r'_{Z_2})$. Q_1 and Q_2 translate $\mathcal{P}re$ and $\mathcal{P}ost$ respectively. Z_1 and Z_2 are the number of messages in $\mathcal{P}re$ and $\mathcal{P}ost$ respectively.

The behaviour of an object O in a par region of level k with N operands, $\mathcal{G}_k = [\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \ldots, \mathcal{R}_N]$, can be translated into a PEPA component behaviour C_k such that $C_k = C_{k,1} \parallel C_{k,2} \parallel \ldots \parallel C_{k,N}$ where each $C_{k,n}$ translates a region $\mathcal{R}_n$, $n = 1, \ldots, N$. If $\mathcal{R}_n$ is a basic region with Z messages then $C_{k,n} \stackrel{def}{=} (a_1, r_1).\ldots.(a_Z, r_Z)$ where a_i and r_i are respectively the action type and one of the rates in message $m_i = a_i/r_{i_1}; r_{i_2}$, $m_i \in \mathcal{T}_{\mathcal{A}_{\mathcal{R}_n}}$ and $i = 1, \ldots, Z$. $r_i = r_{i_1}$ if O is the sender of m_i and $r_i = r_{i_2}$ if O is the receiver. Now, if $\mathcal{R}_n$ is not a basic region that is $\mathcal{R}_n = [\mathcal{P}re.\mathcal{G}_{k-1}.\mathcal{P}ost]$, then $C_{k,n} \stackrel{def}{=} Q_1.C_{k-1}.Q_2$ where $Q_1 \stackrel{def}{=} (a_1, r_1).\ldots.(a_{Z_1}, r_{Z_1})$ and $Q_2 \stackrel{def}{=} (a'_1, r'_1).\ldots.(a'_{Z_2}, r'_{Z_2})$. Q_1 and Q_2 translate $\mathcal{P}re$ and $\mathcal{P}ost$ respectively. Z_1 and Z_2 are the number of messages in $\mathcal{P}re$ and $\mathcal{P}ost$ respectively.

Note that, in both the alt and the par, if an object O is not involved in all regions $\mathcal{R}_n$, $n = 1, \ldots, N$ then only the regions in which it is involved have a derivative in the PEPA component C_k. Similarly, if the object is involved partially in a region, that is not in all the messages in the region's associated set of local transitions, then only the messages in which it is involved are translated into PEPA activities and used in the corresponding derivative.

The behaviour of an object O in a loop region $\mathcal{G} = \mathcal{P}re.(\mathcal{R})^n.\mathcal{P}ost$ can be translated into a PEPA component behaviour $C \stackrel{def}{=} Q_1.C'.Q_2$ where Q_1 and Q_2 translate $\mathcal{P}re$ and $\mathcal{P}ost$ respectively, such that

$Q_1 \stackrel{def}{=} (a_1, r_1).\ldots.(a_{Z_1}, r_{Z_1})$ and $Q_2 \stackrel{def}{=} (a'_1, r'_1).\ldots.(a'_{Z_2}, r'_{Z_2})$. Z_1 and Z_2 are the number of messages in $\mathcal{P}re$ and $\mathcal{P}ost$ respectively. $C' \stackrel{def}{=} C''.C'$ where C'' translates the bahaviour in the arbitrary region $\mathcal{R}$. According to the type of this arbitrary region (alt, par, loop) we use its corresponding translation to PEPA given above. Note that the loop index n is taken into account using functional rates [8].

A definition of a formal semantics for UML sequence diagrams and operators we have used is given in [9]. This semantics is based on the structural operational semantics used in PEPA.

6.2 Equivalence of the languages

In section 5, we have described legal traces, or words, for an IOD node, an IOD, a PEPA place and a PEPA net. The set of legal traces determines the language of an IOD $\mathcal{D}$ given by $L_1(\mathcal{D})$ or a PEPA net $\mathcal{V}$ given by $L_2(\mathcal{V})$. Given the algorithm described in the previous section, we can prove that the languages are equivalent, also known as *strongly consistent*.

Theorem 6.1 *Let $\mathcal{D}$ be an IOD and $\mathcal{V}$ the PEPA net derived from $\mathcal{D}$. If $L_1(\mathcal{D})$ is the set of words over the alphabet L_1 of $\mathcal{D}$ and $L_2(\mathcal{V})$ is the set of words over the alphabet L_2 of $\mathcal{V}$ then*

1. *$L_1 = L_2$ and*
2. *$L_1(\mathcal{D}) = L_2(\mathcal{V})$*

Proof The first point is true by definition, given that $L_1 = Act_n \cup Act_p = Act_t \cup Act_f = L_2$.

The language equality can be proven in two steps: (1) $L_1(\mathcal{D}) \subseteq L_2(\mathcal{V})$ and (2) $L_2(\mathcal{V}) \subseteq L_1(\mathcal{D})$. We prove (1) by contradiction and assume there is a word W such that $W \in L_1(\mathcal{D})$ and $W \notin L_2(\mathcal{V})$. Since strong consistency is assumed by hypothesis, the trace violation occurs at length $i+1$, i.e., there is a trace $W = w_1.c_1.w_2.c_2 \ldots w_i.c_i.w_{i+1}.c_{i+1}$ such that $w_1.c_1.w_2.c_2 \ldots w_i.c_i \in L_2(\mathcal{V})$ but there is no trace in $L_2(\mathcal{V})$ which would contain the continuation $w_{i+1}.c_{i+1}$ and thus there is either no word w_{i+1} in the PEPA place P_{i+1} associated with node $\mathcal{A}_{i+1}$ or there is no net transition $c_{i+1} \in \mathcal{A}_f$. The first assumption contradicts the one-to-one correspondence between the event ordering in an IOD node (and thus the local transition ordering) and the sequences of activities possible for the components in place P_{i+1}. The second assumptiom contradicts the one-to-one correspondence between the IOD transitions and the PEPA net transitions (the languages $L_1 = L_2$). The proof for (2) is similar.

□

Another notion commonly available in synthesis methods is the notion of *weak consistency*, where the language of the target model contains the language of the source model and more. When only a result of weak consistency between languages can be guaranteed then we have a case of implied (unspecified or unacceptable) behaviour in the synthesised models. If this is the case, further methods have to be used to detect such additional behaviours.

7 Related work

With the advances in networking technology and the development of systems based on mobile code, an increased number of approaches have emerged for the design and/or formal verification of mobile systems. At the software design level this includes extensions of UML for mobility (e.g., [1, 6] among many

others). In [1], the authors extend UML activity diagrams to capture mobile systems. Their extension introduces concepts for representing mobile objects, locations, mobile locations, move actions and clone actions, making use of UML's extension mechanisms (stereotypes, tagged values and constraints). Further, the extension of activity diagrams is done in two variants: responsibility-centred (using swimlanes) and location-centred (using composite objects to visualise the hierarchy of locations). However, activity diagrams are not adequate to capture at the same time the structure of the system (locations), how objects move between locations, *and* how objects behave/interact within locations. By contrast, this is possible in our approach using interaction overview diagrams. Furthermore, our usage of IODs for modelling interactions and mobility enhanced with performance information offer a rich language for capturing mobile distributed systems for performance analysis. In particular, our extension of IODs are a natural UML counterpart for the underlying performance technique of PEPA nets [4].

There are several performance modelling approaches using UML and an underlying formal model for performance analysis including [11, 5, 3, 2] among others. Some of the work using UML for performance analysis has different motivation than ours. In this context [11] uses activity diagrams to refine `do` activities in state machines and then obtain predictive performance measures from the performance model obtained from these diagrams. Activity diagrams are annotated with rates and durations according to the UML profile for performance, schedulability and time. In [5] the authors introduce a mobility profile for the performance analysis domain, but do not focus on new notations available in UML2.0.

In [3] the authors report on a toolset for modelling systems with performance information using UML. In this approach, a UML state diagram with performance annotations is mapped onto a PEPA model for performance analysis. The outcomes of the analysis are given to the designer as additional annotations to their original UML model. However, this approach does not consider mobility, and assumes an underlying translation of mainly UML1.x notation into the process algebra PEPA. Our approach is different, because we use recent UML2.0 notation and PEPA nets as an underlying model. We are thus concerned with both mobility and performance information.

In [15] the authors are also concerned with mobility. However they propose a translation of UML1.x specifications made up of sequence and state diagrams into $\pi-$calculus processes. Our approach describes a mobile system at two levels. At the high level we describe the locations of the system and how objects move between locations which is given in UML by an IOD. At the lower level we describe how objects behave and interact locally. This is given by the individual nodes of the IOD, namely sequence diagrams. Both levels are enriched with performance related information (i.e., activities). This approach does in particular allow us to define an automatic transformation of IODs into PEPA nets.

As shown in this paper, our approach to synthesising PEPA nets from IODs is such that the underlying languages of both models are strongly consistent. This guarantees the absence of so-called implied scenarios at the PEPA net level. Implied scenarios are additional scenarios or behaviour that was not specified or intended. Other synthesis approaches, e.g.[18], often have this problem as the models used are very different in nature and essentially capture different views of the system. Transforming a model with a global system view into a model based on individual and local object views makes it impossible to prevent implied scenarios from existing. Such approaches then have to focus on mechanisms to detect such unwanted and unacceptable additional behaviours.

An interesting aspect of PEPA nets lies in the combination of Petri nets with a process algebra therefore combining strengths of two different formalisms. Using IODs we lift the benefits to the UML design level. The new sequence diagrams and IODs in UML2.0 have been largely influenced by Message Sequence Charts (MSCs)[12], a common approach used in the telecommunication domain to represent scenarios, and their higher-level language HMSCs. Motivated by the fact that HMSCs are not an executable model and have limited expressiveness, the authors in [13] introduced a new visual formalisms

called netcharts. Netcharts combine Petri nets with MSCs, but instead of having MSCs in places (as in IODs or PEPA nets) netcharts have MSCs in transitions. Another similar but reverse approach which combines MSCs with Petri nets is [17]. The approach differs from [13] as it uses Petri nets to describe the relations among scenarios including multiple concurrent processes. In that respect it follows a more similar approach to ours.The approach in [17] supports analysis and synthesis of MSC scenarios through available analysis methods for Petri nets but does not address performance analysis as our approach does. The MSCs used in [17] are also basic whereas we use sequence diagrams with (possibly nested) interaction fragments.

8 Conclusion

In this paper, we have shown how to formalise performance annotated IODs and IOD nodes taking into account complex behaviour within a node determined by several and possibly nested interaction fragments. We defined the languages associated with IODs and PEPA nets, and presented an algorithm to synthesise a PEPA net model from an IOD model. We further showed how the algorithm guarantees that the languages are *strongly consistent*. In other words, the set of legal traces of an IOD have a one-to-one correspondence to the set of legal traces of the underlying PEPA net model. As mentioned in the related work section, one crucial advantage of strongly consistent languages is the guaranteed absence of implied (unspecified or unacceptable) behaviours that can be observed in the synthesised model. The absence of implied scenarios in our approach facilitates an accurate performance analysis on the given UML design models.

We are currently completing the implemention of the algorithm for the IOD-to-PEPAnet transformation in C. The implementation currently takes a textual language for the IOD model and supports the formal IOD model as described in this paper. A future extension should take a (hopefully standardised) XMI file and do a model transformation following a model-driven development (MDD) approach. In other words, we aim to define the exact rules of the transformation and implement them using the Simple Transformer tool SiTRA[1]. One advantage is that possible extensions to the UML model and implementation are easier to integrate.

One extension we want to be able to bring to the UML level is the ability to express priorities of object moves by adding the information to IOD edges. As discussed in section 3, PEPA nets already incorporate the notion of priorities and we restricted them here to consider net level transitions with the same priority.

A more challenging aspect of our future work concerns the performance analysis itself. PEPA nets mainly rely on the performance techniques available for PEPA and these ignore the location or mobility information of the PEPA net. By contrast we want to exploit the design structure of our IOD and PEPA nets to enhance verification and scalability and thus have a more suitable approach for performance evaluation of complex mobile distributed applications.

9 Acknowledgements

This work was supported by *A Performance Engineering Technique for Mobile Distributed Systems (PETMoDS)*, a joint international project funded by the Royal Society and CNRS.

[1] `http://www.cs.bham.ac.uk/~bxb/SiTra.html`

References

[1] H. Baumeister, N. Koch, P. Kosiuczenko, and M. Wirsing. Extending activity diagrams to model mobile systems. In *Proc. of NetObjectDays 2002, Erfurt, Germany*, volume 2591 of *LNCS*, pages 278–293. Springer, 2003.

[2] C. Canevet, S. Gilmore, J. Hillston, L. Kloul, and P. Stevens. Analysing UML 2.0 activity diagrams in the software performance engineering process. In *WOSP 2004, Short papers*, pages 74–78. ACM, 2004.

[3] C. Canevet, S. Gilmore, J. Hillston, M. Prowse, and P. Stevens. Performance modelling with UML and stochastic process algebras. *IEE Proceedings: Computers and Digital Techniques*, 150(2):107–120, Mar. 2003.

[4] S. Gilmore, J. Hillston, L. Kloul, and M. Ribaudo. PEPA nets: a structured performance modelling formalism. *Performance Evaluation*, 54(2):79–104, 2003.

[5] V. Grassi, R. Mirandola, and A. Sabetta. UML based modeling and performance analysis of mobile systems. In *Proc. of ACM MSWIM 2004, 4-6 October 2004, Venezia, Italia*, pages 95–104. ACM, 2004.

[6] V. Grassi, R. Mirandola, and A. Sabetta. A UML profile to model mobile systems. In T. Baar, A. Strohmeier, A. Moreira, and S. J. Mellor, editors, *UML 2004 - The Unified Modeling Language. Model Languages and Applications. 7th International Conference, Lisbon, Portugal, October 11-15, 2004, Proceedings*, volume 3273 of *LNCS*, pages 128–142. Springer, 2004.

[7] J. Hillston. *A compositional approach to performance modelling*. Cambridge University Press, 1996.

[8] J. Hillston and L. Kloul. An efficient kronecker representation for pepa models. In *Proc. of the Joint International Workshop, PAPM-PROBMIV*, number 2165, pages 120–135. LNCS, Springer Verlag, 2001.

[9] L. Kloul. Blending UML2.0 and PEPA nets. Technical Report n.2006/102, PRiSM, Université de Versailles, http://wwwex.prism.uvsq.fr/recherche/rapports, 2006.

[10] L. Kloul and J. Küster-Filipe. Modelling Mobility with UML 2.0 and PEPA Nets. In K. Goossens and L. Petrucci, editors, *Sixth International Conference on Application of Concurrency to System Design*, pages 153–162. IEEE Computer Society, 2006.

[11] J. López-Grao, J. Merseguer, and J. Campos. From UML Activity Diagrams to Stochastic Petri Nets: Application to Software Performance Engineering. In *WOSP'04: Proceedings of the fourth international workshop on Software and Performance*, pages 25–36. ACM Press, 2004.

[12] *ITU-TS Recommendation Z.120 (11/99): MSC 2000. ITU-TS, Geneva*, 1999.

[13] M. Mukund, K. N. Kumar, and P. Thiagarajan. Netcharts: Bridging the gap between HMSCs and executable specifications. In R. Amadio and D. Lugiez, editors, *Proceedings of Concur 2003 - Concurrency Theory*, volume 2761 of *LNCS*, pages 296–310. Springer, 2003.

[14] OMG. *UML 2.0 Superstructure Specification*. OMG document ptc/05-07-04, available from www.uml.org, August 2005.

[15] K. Pokozy-Korenblat and C. Priami. Towards extracting $\pi-$calculus from UML sequence and state diagrams. *Electronical Notes in Theoretical Computer Science*, 101:51–72, 2004.

[16] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modelling Language Reference Manual*. Addison-Wesley, 2nd edition, 2005.

[17] M. Sgroi, A. Kondratyev, Y. Watanabe, L. Lavagno, and A. Sangiovanni-Vincentelli. Synthesis of petri nets from message sequence charts specifications for protocol design. In *Design, Analysis and Simulation of Distributed Systems Symposium (DASD '04)*, 2004.

[18] S. Uchitel, J. Kramer, and J. Magee. Detecting implied scenarios in message sequence chart specifications. In *Prooceedings of the 9th European Software Engineering Conference and 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 74–82, 2001.