Synthesising PEPA Nets from IODs for Performance Analysis[*]

Juliana Bowles
University of St-Andrews
School of Computer Science
North Haugh, St-Andrews KY16 9SX
jkfb@st-andrews.ac.uk

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

ABSTRACT

The object-based Unified Modeling Language (UML) is a popular medium for effective design of most systems. PEPA nets are a 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.

Categories and Subject Descriptors: D.2.2 [Design Tools and Techniques]: object-oriented design methods. I.6.5 [Model Development]: Modeling methodologies.

General Terms: Design, Languages, Performance

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.

There are several performance modelling approaches using UML and an underlying formal model for performance analysis including [9, 5, 3] among others. Some of the work using UML for performance analysis has different motivation than ours. In this context [9] 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. Similarly, in [3] the authors report on a toolset for modelling systems with performance information using UML but do not consider mobility and assume an underlying translation of mainly UML1.x notation into the process algebra PEPA. We use recent and new UML2 notation with PEPA nets as an underlying model, and are concerned with both mobility and performance information.

In [12] the authors propose a translation of UML1.x specifications made up of sequence and state diagrams into $\pi-$calculus processes. In [1], the authors extend UML activity diagrams to capture mobile systems. 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 performance annotated IODs and we obtain a rich language for capturing mobile distributed systems for performance analysis.

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 (IOD). At the lower level we describe how objects behave and interact locally (sequence diagrams). Both levels are enriched with performance related information (activities) as described in [8]. This approach does in particular allow us to define an automatic transformation of IODs into PEPA nets.

The main contribution of this paper is a constructive approach to synthesise a PEPA net which realises the same language as a given UML2 IOD. We prove the equivalence of both languages by showing that the corresponding models are strongly consistent, that is by establishing the correspondence between the traces of the languages. Our result 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. [13], often have this problem as the models used are very different in nature and essentially cap-

[*]This work is supported by the CNRS/Royal Society Project PETMoDS (CNRS:IKSLU85079, RS:JP0867690).

ture 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.

Structure of this paper: In the next section, we present the UML2 interaction diagrams. In Section 3, we describe PEPA nets. In Section 4, we give the formal model for IODs. Section 5 describes the languages associated with both models. Section 6 describes the synthesis algorithm and gives a proof for the equivalence of the languages. Finally, Section 7 concludes the paper.

2. INTERACTION DIAGRAMS IN UML2.0

To model interactions UML2 offers various diagrams. We consider sequence diagrams and interaction overview diagrams.

Sequence diagrams are the more commonly used diagrams for capturing inter-object behaviour. In UML2, sequence diagrams are more expressive and interactions can be structured using so-called interaction fragments such as **alt** (alternative behaviour), **par** (parallel behaviour) and **loop**. The semantics of all operators is described informally in the UML2 superstructure specification [11].

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 [10]. In UML2, IODs are a special and restricted kind of activity diagrams (ADs) where nodes are interactions and edges indicate the flow or order in which these interactions occur [11]. Semantically, however, IODs and ADs are given different interpretations. IODs follow a trace semantics whereas ADs in UML2 are understood as Petri nets.

The notation used for IODs incorporates notation from sequence diagrams (the nodes) with forks, joins, decision and merge nodes from ADs. Branching and joining of branches in an IOD must be properly nested. IOD edges denote control flow only and according to the UML specification [11] object flow cannot be represented.

Object flow in an AD uses the notion of a pin. After an AD node has been completed a token of a certain type is placed in the output pin of the node. As soon as the edge fires the token moves to the input pin of another AD node. 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. 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. 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.

Even though IODs only describe control flow 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.

We are primarily interested in explicitly modelling the mobility of objects, and we borrow the notation of object flow and pins from ADs as shown in Figure 1. In other words, we do not allow simple edges between nodes and we always have to indicate pins on edges.

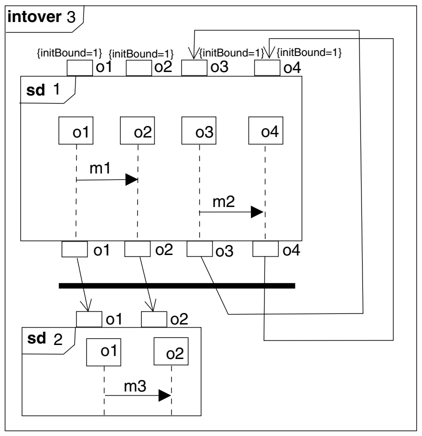

Figure 1: Object mobility in an IOD.

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. In Figure 1, 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. Here, the edges for objects `o1` and `o2` cannot fire independently and are 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`. In other words, a fork is used to synchronise the objects associated with the edges it cuts accross. Independent progression can be modelled without the fork.

The tagged value $\{initBound = n\}$ given next to a pin indicates 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 of the IOD. This gives the initial marking of the IOD.

In [8] 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. We add the explicit activity (an action type with its corresponding rate) to an IOD edge 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. 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*.

As shown in Figure 2, the textual label of a source pin is given by: `pin_type;action_type/rate`.

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 a shared activity between objects).

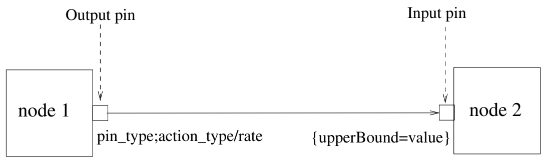

Figure 2: Input and output pins.

3. PEPA NETS

PEPA nets [4] combine the process algebra PEPA (Performance Evaluation Process Algebra) with stochastic coloured Petri nets. This hybrid formalism can be regarded as using the 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 applications such 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 3. In the grammar S

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

Figure 3: The syntax of PEPA nets

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 definition.

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]. The states of the model are the marking vectors, which have one entry for each place of the PEPA net. The semantic rules govern the possible evolution of a state, giving rise to a labelled transition system or derivation graph. The nodes of the graph are the marking vectors and the activities (individual, shared or firing activities) give the arcs of the graph. This graph gives rise to a CTMC which can be solved to obtain a steady-state probability distribution from which performance measures can be derived.

4. A FORMAL IOD MODEL

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

DEFINITION 4.1. *$\mathcal{D} = (\mathcal{N}, \mathcal{S}, \mathcal{T}, \mathcal{P}, \mathcal{A}ct, \mathcal{L}_O, \mathcal{L}_I, \mathcal{F}, \mathcal{C}, \mathcal{B})$ is an IOD 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 $r \in \mathbb{R}^+ \cup \{\top\}$;*
- *$\mathcal{L}_O$: $\mathcal{T} \to \{\mathcal{P}_O, \mathcal{A}ct_p\}$ is a total labelling function which assigns an output pin type and an activity to the source pin of a transition;*
- *$\mathcal{L}_I$: $\mathcal{T} \to \{\mathcal{P}_I, \mathbb{N}^+\}$ is a total labelling function which assigns an input pin type and an upper bound to the target pin of a transition;*
- *$\mathcal{F}$: $\mathcal{T} \to \mathcal{N} \times \mathcal{N}$ is a total function which assigns a pair of nodes (a source node and a target node) to a transition;*
- *$\mathcal{C}$: $\mathcal{S} \to 2^{\mathcal{T}}$ is a total function which assigns a set of transitions to a fork node;*
- *$\mathcal{B} : \mathcal{P} \to \mathbb{N}$ 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 a 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. 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 1. 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. 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 `o1` in `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 $\mathcal{A} = (\mathcal{O}, \mathcal{E}, <, \mathcal{M}_A, \mathcal{T}_\mathcal{A}, \mathcal{P}_\mathcal{A}, \mu_\mathcal{A}, \mathcal{I}_\mathcal{A}, \mathcal{U}_\mathcal{A})$ 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, and $\mathcal{E} = \bigcup_{o \in \mathcal{O}} \mathcal{E}_o$ 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 $\mathcal{T}_\mathcal{A} \subseteq \mathcal{E}_S \times \mathcal{M}_A \times \mathcal{E}_R$;*
- *$\mathcal{P}_\mathcal{A}$ is the set of pin types of $\mathcal{A}$ such that $\mathcal{P}_\mathcal{A} \subseteq \mathcal{P}$;*
- *$\mu_\mathcal{A} : \mathcal{P}_\mathcal{A} \to \mathcal{O}_M$ 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 in $\mathcal{O}$. 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 $r = r_2$. Here 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 $\mathcal{U}_\mathcal{A}$.

We can describe a variety of behaviour in a node using interaction fragments for parallelism, alternatives and loops. To capture this behaviour we use a notion of *region* (subset of events). For space reasons we omit details and refer the interested reader to [2].

5. LANGUAGES

5.1 The language of an IOD

Given the formal model of an IOD as given above, we now define its associated language. We define $\mathcal{L}(D)$ as the legal set of traces of IOD $\mathcal{D}$. 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.

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 $\mathcal{B}$.*

Now we define the language of an IOD $\mathcal{D}$, noted $L_1(\mathcal{D})$.

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 with rate $r_i = min(r_{i1}, r_{i2})$ 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 $L_2(\mathcal{V})$.

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

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 corresponding languages.

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. An IOD can be viewed as a PEPA net model where each IOD node corresponds to a place in the PEPA net. A transition between IOD nodes is transformed into a firing transition between 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 definitions.

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

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 $\mathcal{A}$ $(p, n) \in \mathcal{I}_\mathcal{A}$	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

model and the PEPA net model we only consider the type of the object. Inside a 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 which the 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 that an object of type O_1 sends to an object of type O_2. This message, 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 and a passive component by considering which associated object sends the message. 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 $b/\top; r_2$ (see [7]).

6.2 Equivalence of the languages

With the algorithm described previously, 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) $L_1(\mathcal{D}) = L_2(\mathcal{V})$.*

The proof is straightforward and can be found in [2].

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. CONCLUSION

In this paper, we have shown how to formalise performance annotated IODs. For details on IOD nodes taking into account complex behaviour within a node determined by several and possibly nested interaction fragments see [2]. 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. 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.

8. REFERENCES

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

[2] J. Bowles and L. Kloul. Strongly Consistent Languages for Modelling Mobility, Technical Report, hal-00419934, http://hal.archives-ouvertes.fr/hal-00419934, 2009.

[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, March 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*, pages 95–104. ACM, 2004.

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

[7] 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.

[8] L. Kloul and J. Küster-Filipe. Modelling Mobility with UML 2.0 and PEPA Nets. In *ACSD 2006*, pages 153–162. IEEE Computer Society, 2006.

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

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

[11] OMG. *UML Superstructure Specification*. OMG document ptc/09-02-02, available from www.uml.org, February 2009.

[12] K. Pokozy-Korenblat and C. Priami. Towards extracting $\pi-$calculus from UML sequence and state diagrams. *ENTCS*, 101:51–72, 2004.

[13] S. Uchitel, J. Kramer, and J. Magee. Detecting implied scenarios in message sequence chart specifications. In *ESEC/FSE 2001*, pages 74–82, 2001.