Modelling Mobility with PEPA Nets

Jane Hillston[1]* and Marina Ribaudo[2]**

[1] Laboratory for Foundations of Computer Science, The University of Edinburgh
jeh@inf.ed.ac.uk
[2] Dipartimento di Informatica e Scienze dell'Informazione, Università di Genova
ribaudo@disi.unige.it

Abstract. We explain the use of PEPA nets in documenting high-level designs of mobile code and mobile computing systems. This modelling language (which allows the modeller to differentiate between location-changing and state-changing operations in the system, and to quantify their computational expense for use in predictive performance analysis) is applied to the problem of modelling the canonical mobile code design paradigms which are in practical application today, as described in [2].

1 Introduction

Mobile code design paradigms have received heightened attention with the advent of secure and portable programming languages such as Java. A mobile code infrastructure allows application designers to distribute computational effort across networked hosts which can have many benefits. For example it may be possible to reduce the workload on central server components by moving computation from the server to its clients.

Selecting the optimum design from a suite of alternative designs for a distributed mobile code system may involve ranking the alternatives on the basis of key performance metrics such as response time, throughput and utilisation. Statements of required performance measures such as these are increasingly being used in service-level agreements meaning that the efficient and reliable computation of performance measures from a high-level model of a mobile code system is becoming a critical part of state-of-the-art application development.

It is imperative that the modelling formalism used to express these high-level performance models should reliably capture notions of *location*, *context* and *evaluation environment*. Additionally, it should clearly distinguish local computation at one of the locations from the movement of code-containing objects from one location to another. The latter brings about a change in the communication topology of the system, allowing patterns of communication which had previously not been possible because the partners in the communication were separated by being on different hosts in different address spaces, or behind firewalls or other administrative domain boundaries.

* Jane Hillston is supported by the DEGAS (Design Environments for Global ApplicationS) IST-2001-32072 project in the FET Proactive Initiative on Global Computing.
** Marina Ribaudo is supported by the FIRB project WEBMINDS (Wide-scalE, Broadband, MIddleware for Network Distributed Services).

To this end we have developed a high-level Petri net formalism which we believe naturally captures performance models of mobile code systems meeting the requirements suggested above. The *PEPA nets* formalism uses Hillston's Performance Evaluation Process Algebra [6] as the inscription language for coloured stochastic Petri nets. We have implemented software tools to facilitate the processing and performance analysis of PEPA nets [5] and created a hierarchical drawing tool for representing them graphically on screen [3].

In Section 2 we present the description and formal definitions of the PEPA nets modelling language. The definitions are illustrated by simple examples. Section 3 discusses the key design paradigms for code mobility, presenting PEPA net models of each, and some conclusions and directions for future work are presented in Section 4.

2 PEPA Nets

In this section we provide a brief overview of PEPA nets and the PEPA stochastic process algebra. A fuller description is available in [4] and [6].

The 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 their interactions. One example of a PEPA component would be a *File* object which can be opened for reading or writing, have data read (or written) and closed. Such an object would understand the methods *openRead*(), *openWrite*(), *read*(), *write*() and *close*().

$$\begin{aligned} \mathit{File} &\stackrel{\mathit{def}}{=} (\mathit{openRead}, r_o).\mathit{InStream} + (\mathit{openWrite}, r_o).\mathit{OutStream} \\ \mathit{InStream} &\stackrel{\mathit{def}}{=} (\mathit{read}, r_r).\mathit{InStream} + (\mathit{close}, r_c).\mathit{File} \\ \mathit{OutStream} &\stackrel{\mathit{def}}{=} (\mathit{write}, r_w).\mathit{OutStream} + (\mathit{close}, r_c).\mathit{File} \end{aligned}$$

This PEPA model documents a high-level protocol for using *File* objects, from which it is possible to derive properties such as "it is not possible to write to a closed file" and "read and write operations cannot be interleaved: the file must be closed and re-opened first".

Every activity incurs an execution cost which is quantified by an estimate of the (exponentially-distributed) rate at which it can occur (r_o, r_r, r_w, r_c). Activities may be *passive*, i.e., they can be executed only in cooperation with corresponding active ones. The rate of a passive activity is denoted by $\top$.

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, such as the *File* object described above. The components which fill cells can circulate as the tokens of the net. In contrast, the static components cannot move. A typical place might be the following:

$$\mathit{File}[\_] \underset{L}{\bowtie} \mathit{FileReader}$$

where the *synchronisation set* L in this case is $\boldsymbol{\mathcal{A}}(\mathit{File})$, the *complete action type set* of the component, ($\mathit{openRead}$, $\mathit{openWrite}$, ...). This place has a *File*-type cell and a static component, *FileReader*, which can process the file when it arrives.

A PEPA net differentiates between two types of change of state. We refer to these as *firings* of the net and *transitions* of PEPA components. Each are special cases of PEPA activities. Transitions of PEPA components will typically be used to model small-scale (or *local*) changes of state as components undertake activities. 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. The set of all firings is denoted by $\mathcal{A}_f$, the set of all transitions by $\mathcal{A}_t$. We distinguish firings syntactically by printing their names in boldface.

Continuing our example, we introduce an instant message as a type of transmissible file.

$$\mathit{InstantMessage} \overset{\mathit{def}}{=} (\mathbf{transmit}, r_t).\mathit{File}$$

Part of a definition of a PEPA net which models the passage of instant messages is shown below. An instant message *IM* can be moved from the input place on the left to the output place on the right by the **transmit** firing. In doing so it changes state to evolve to a *File* derivative, which can be read by the *FileReader*.

$$\boxed{\mathit{InstantMessage}[\mathit{IM}]} \xrightarrow{(\mathbf{transmit}, r_t)} \boxed{\mathit{File}[\_] \underset{L}{\bowtie} \mathit{FileReader}}$$

The syntax of PEPA nets is given in Figure 1. S denotes a *sequential component* and P a *concurrent component* which executes in parallel. I stands for a constant denoting either a sequential or a concurrent component, as bound by a definition.

Definition 1 (PEPA Net). *A PEPA net* $\mathcal{N}$ *is a tuple* $\mathcal{N} = (\mathcal{P}, \mathcal{T}, I, O, \ell, \pi, \mathcal{C}, D, M_0)$ *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;*
- $\ell : \mathcal{T} \rightarrow (\mathcal{A}_f, \mathbb{R}^+ \cup \{\top\})$ *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;*
- $\pi : \mathcal{A}_f \rightarrow \mathbb{N}$ *is the priority function which assigns priorities (represented by natural numbers) to firing action types;*
- $\mathcal{C} : \mathcal{P} \rightarrow P$ *is the place definition function which assigns a PEPA context, containing at least one cell, to each place;*
- D *is the set of token component definitions;*
- M_0 *is the initial marking of the net.*

The structured operational semantics, defined in [4], give a precise definition of the possible evolution of a PEPA net, and shows how a CTMC can be derived, treating each marking as a distinct state.

$$
\begin{array}{rlr}
N ::= & D^{+} M & \text{(net)} \\
& & \\
M ::= & (M_{\mathbf{P}}, \ldots) & \text{(marking)} \\
M_{\mathbf{P}} ::= & \mathbf{P}[C, \ldots] & \text{(place marking)} \\
& & \\
D ::= & I \overset{\mathit{def}}{=} S & \text{(component defn)} \\
\mid & \mathbf{P}[C] \overset{\mathit{def}}{=} P[C] & \text{(place defn)} \\
\mid & \mathbf{P}[C, \ldots] \overset{\mathit{def}}{=} P[C] \underset{L}{\bowtie} P & \text{(place defn)}
\end{array}
\qquad
\begin{array}{rlr}
P ::= & P \underset{L}{\bowtie} P & \text{(cooperation)} \\
\mid & P/L & \text{(hiding)} \\
\mid & P[C] & \text{(cell)} \\
\mid & I & \text{(identifier)} \\
& & \\
C ::= & \text{`\_'} & \text{(empty)} \\
\mid & S & \text{(full)} \\
& & \\
S ::= & (\alpha, r).S & \text{(prefix)} \\
\mid & S + S & \text{(choice)} \\
\mid & I & \text{(identifier)}
\end{array}
$$

Fig. 1. The syntax of PEPA nets

We define the firing rule of PEPA nets to respect the net structure in the usual way (one token from each input place, one token to each output place) but also to take into consideration the ability of tokens to participate in the firing (can they perform an activity of the correct type?), and the availability of vacant cells of the appropriate type in the output places. Note that we require that the net is *balanced* in the sense that, for each transition, the number of input places is equal to the number of output places. In classical Petri nets tokens are identitiless, and can be viewed as being consumed from input places and created into output places for each firing. In contrast, in PEPA nets our tokens have state and identity, and we view them as *passing through* net-level transitions. For each firing there must be as many output tokens as there were input tokens.

Definition 2 (Enabling). *An enabling is a mapping of places to tokens. A net level transition t has an* enabling *of firing type α, $\mathrm{E}(t, \alpha)$, if for each input place $\mathbf{P}_i$ of t there is a token T in the current marking of $\mathbf{P}_i$, which has a one-step α-derivative, T'.*

Note that there may be several enablings for a given transition firing in any particular marking, as the enabling selects one token to fire from each input place, and there may be more than one eligible token at each input place.

Since it is important that each fired token has a vacant cell to go into after the firing, we define a corresponding notion of *output*. A transition has an output if, in the current marking, there is at least one vacant cell in each output place.

Definition 3 (Output). *For any net level transition t, an* output*, denoted $O(t)$, is a mapping from the output places of t to vacant cells in the current marking.*

Since each token passes through a net level transition when it fires, such a transition is enabled only when there is a bijective function between the chosen enabling and an output.

Definition 4 (Concession). *A net level transition t has* concession *for a firing of type α if there is an enabling $\mathrm{E}(t, \alpha)$ such that there is a bijective mapping ϕ from $\mathrm{E}(t, \alpha)$ to an output $O(t)$, which preserves the types of tokens.*

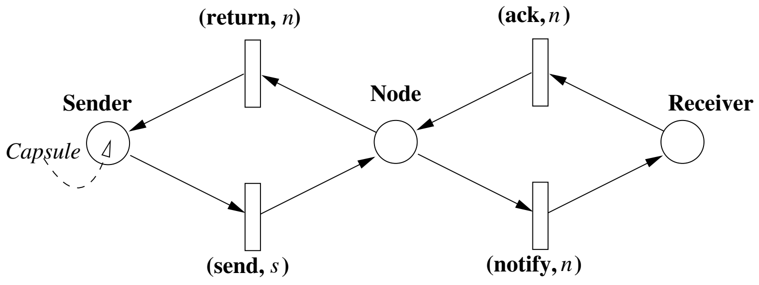

Fig. 2. PEPA net of the simple active network

As with classical Petri nets with priority, having concession identifies those transitions which could legally fire according to the net structure and the current marking. The set of transitions which *can* fire is determined by the priorities.

Definition 5 (Enabling Rule). *A net level transition t will be* enabled *for a firing of type α if there is no other net transition of higher priority with concession in the current marking.*

Definition 6 (Firing Rule). *When a net level transition t fires with type α on the basis of the enabling* $\mathrm{E}(t, \alpha)$*, and concession ϕ then for each* $(\mathbf{P}_i, T,)$ *in* $\mathrm{E}(t, \alpha)$*, $T[T]$ is replaced by $T[\_]$ in the marking of $\mathbf{P}_i$, and the current marking of each output place is updated according to ϕ.*

We assume that when there is more than one mapping ϕ from an enabling to an output, then they have equal probability and one is selected randomly. The rate of the enabled firing is determined using apparent rates, and the notion of bounded capacity, as usual for PEPA. We refer the reader to [5] for more details.

2.1 Small Example

We present a small example PEPA net in Figure 2 considering an *active switch* within an *active network*. In an active network, in addition to straightforward routing, a switch may carry out some computational task, processing the packets of a message as they pass through the node. Several approaches to active networking have been suggested, one of which is the use of *capsules*, i.e., special packets which activate the active network with respect to the following message, or stream of packets. Our model represents a sender and receiver communicating through an active intermediate node. We assume that the sender supplies a continuous stream of packets which are processed by the switch (not explicitly represented). When the sender wants to take advantage of the processing capabilities of the switch for some message, it sends a capsule to activate the switch. The capsule also notifies the receiver, and then resides in the switch until processing the message is complete, at which point it returns to the sender.

There is one token type in the PEPA net (*Capsule*), representing the capsule which is sent to activate an active switch, with respect to the following packets. We assume that after this (activity *activate*), a notification is sent to the *Receiver* that the following packets will have been processed in transit. This is acknowledged by the *Receiver*, returning the *Capsule* to the switch. After a random delay, approximating the length of the message to be processed, the *Capsule* reverts the *Switch* to normal processing via the activity *deactivate*. A final report is sent back to the *Sender*, represented by the firing **return**.

$$\begin{aligned} \mathit{Capsule} &\overset{\mathit{def}}{=} (\mathit{generate}, \gamma).(\mathbf{send}, \top).(\mathit{activate}, \alpha).(\mathbf{notify}, \top).\mathit{Capsule}' \\ \mathit{Capsule}' &\overset{\mathit{def}}{=} (\mathbf{ack}, \top).(\mathit{deactivate}, \alpha).(\mathbf{return}, \top).\mathit{Capsule} \end{aligned}$$

In this model the places *Sender* and *Receiver* do not represent any functionality beyond the processing of the *Capsule*. Therefore each place simply contains a cell of type *Capsule*. We assume that in the initial marking the *Capsule* is with the *Sender* and that the *Receiver* has just an empty cell.

$$\mathit{Sender} \overset{\mathit{def}}{=} \mathit{Capsule}[\mathit{Capsule}] \qquad\qquad \mathit{Receiver} \overset{\mathit{def}}{=} \mathit{Capsule}[\_]$$

The remaining place, *Node*, representing the node hosting the switch, contains a static component. This component represents the functionality of the switch. Before activation it will repeatedly *route* packets arriving on the input stream. The activity *activate* must be carried out in cooperation with a *Capsule* component, and has the effect of introducing an additional step to packet processing: computation (activity *compute*) is done before routing. The switch continues in this mode until the *deactivate* activity is performed, again in cooperation with the *Capsule*. The place *Node* and the *Switch* component are defined as follows:

$$\begin{aligned} \mathit{Node} &\overset{\mathit{def}}{=} \mathit{Switch} \underset{\{\mathit{activate}, \mathit{deactivate}\}}{\bowtie} \mathit{Capsule}[\_] \\ \mathit{Switch} &\overset{\mathit{def}}{=} (\mathit{route}, \rho).\mathit{Switch} + (\mathit{activate}, \top).\mathit{Active\_Switch} \\ \mathit{Active\_Switch} &\overset{\mathit{def}}{=} (\mathit{compute}, c).(\mathit{route}, \rho).\mathit{Active\_Switch} + (\mathit{deactivate}, \alpha).\mathit{Switch} \end{aligned}$$

3 Design Paradigms for Code Mobility

In this section we assess the expressiveness of PEPA nets for modelling mobility. We consider a classification of types of mobility found in [2], where Fuggetta *et al.* identify four key design paradigms for code mobility: *Client-Server*, *Remote Evaluation*, *Code-on-Demand* and *Mobile Agent*. Each is exemplified by a simple scenario involving a chocolate cake and two friends, Louise and Christine. Here we present the PEPA net model of each scenario. The infrastructure of our model, representing the locations in the scenario, is as shown in Figure 3. The two places **LH** and **CH** represent Louise's house and Christine's house respectively. The labelling function for the transitions depends on the design paradigm. At the end of the section we also create and model an additional scenario, depicting the increasingly popular *Web Service* paradigm.

Client-Server. *Louise would like to have a chocolate cake, but she doesn't know the recipe, and she does not have the ingredients or an oven. Fortunately, she*

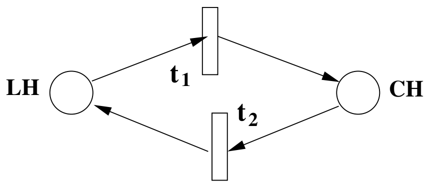

Fig. 3. Net structure for PEPA net models of all original chocolate cake scenarios

knows that Christine knows how to make a chocolate cake, and that she has a well supplied kitchen at her place. Since Christine is usually quite happy to prepare cakes on request, Louise phones her asking: "Can you make me a chocolate cake, please?" Christine makes the chocolate cake and delivers it back to Louise.

$$\begin{aligned}
&\mathit{Louise} \overset{\mathit{def}}{=} (\mathit{want}, w).(\mathit{eat}, e).\mathit{Louise} \\
&\mathit{Recipe} \overset{\mathit{def}}{=} (\mathit{mix}, m).\mathit{Recipe} \qquad\qquad \mathit{Oven} \overset{\mathit{def}}{=} (\mathit{bake}, b).\mathit{Oven} \\
&\mathit{Cake} \overset{\mathit{def}}{=} (\mathit{want}, \top).(\mathbf{phone}, p).(\mathit{mix}, \top).(\mathit{bake}, \top).(\mathbf{deliver}, d).(\mathit{eat}, \top).\mathit{Cake} \\
&\mathbf{LH} \overset{\mathit{def}}{=} \mathit{Louise} \underset{\{\mathit{want},\mathit{eat}\}}{\bowtie} \mathit{Cake}[\mathit{Cake}] \qquad \mathbf{CH} \overset{\mathit{def}}{=} \mathit{Cake}[\_] \underset{\{\mathit{mix},\mathit{bake}\}}{\bowtie} (\mathit{Recipe} \parallel \mathit{Oven}) \\
&\ell(t_1) = \mathbf{phone},\ \ell(t_2) = \mathbf{deliver}
\end{aligned}$$

This PEPA net has a single token which represents the cake, as it evolves from Louise's desire to an actual cake when Louise eats. There are three static components, representing Louise, the recipe and the oven respectively. The recipe is needed for the cake to be mixed and the oven for the cake to be baked: both are located, in this case, at Christine's house. Thus the cake moves from Louise's house to Christine's house for these stages of its evolution.

Remote Evaluation. *Louise wants to prepare a chocolate cake. She knows the recipe but she has at home neither the required ingredients nor an oven. Christine has both at her place, yet she doesn't know how to make a chocolate cake. Louise knows that Christine is happy to try new recipes, therefore she phones Christine asking: "Can you make me a chocolate cake? Here is the recipe: take three eggs..." Christine prepares the cake following Louise's recipe and delivers it back to her.*

$$\begin{aligned}
&\mathit{Louise} \overset{\mathit{def}}{=} (\mathit{want}, w).(\mathit{eat}, e).\mathit{Louise} \\
&\mathit{Oven} \overset{\mathit{def}}{=} (\mathit{bake}, b).\mathit{Oven} \qquad \mathit{Recipe} \overset{\mathit{def}}{=} (\mathit{want}, \top).(\mathbf{phone}, p).(\mathit{mix}, m).\mathit{Recipe}' \\
&\mathit{Cake} \overset{\mathit{def}}{=} (\mathit{mix}, \top).(\mathit{bake}, \top).(\mathbf{deliver}, d).(\mathit{eat}, \top).\mathit{Cake}' \\
&\mathbf{LH} \overset{\mathit{def}}{=} \mathit{Louise} \underset{\{\mathit{want},\mathit{eat}\}}{\bowtie} (\mathit{Recipe}[\mathit{Recipe}] \parallel \mathit{Cake}[\_]) \\
&\mathbf{CH} \overset{\mathit{def}}{=} \mathit{Cake}[\mathit{Cake}] \underset{\{\mathit{mix},\mathit{bake}\}}{\bowtie} (\mathit{Recipe}[\_] \parallel \mathit{Oven}) \\
&\ell(t_1) = \mathbf{phone},\ \ell(t_2) = \mathbf{deliver}
\end{aligned}$$

In this case there are two tokens: one for the cake and one for the recipe. When Louise decides that she wants a cake she changes the state of the *Recipe* component so that it may be moved to Christine's house where it contributes to the evolution of the cake. Once ready the cake is delivered to Louise's house. As previously the oven is represented by a static component, resident at Christine's house and contributing to the baking of the cake at the appropriate time.

Code-on-Demand. *Louise wants to prepare a chocolate cake. She has at home both the ingredients and an oven, but she lacks the proper recipe. However, Louise knows that Christine has the recipe and she has already lent it to many friends. So, she phones Christine asking "Can you tell me your chocolate cake recipe?" Christine tells her the recipe and Louise prepares the chocolate cake at home.*

$$\begin{aligned}
\mathit{Louise} &\overset{def}{=} (\mathit{want}, w).(\mathit{eat}, e).\mathit{Louise}\\
\mathit{Oven} &\overset{def}{=} (\mathit{bake}, b).\mathit{Oven} \qquad \mathit{Recipe} \overset{def}{=} (\mathit{recipe}, \top).(\mathbf{relate}, s).(\mathit{mix}, m).\mathit{Recipe}'\\
\mathit{Cake} &\overset{def}{=} (\mathit{want}, \top).(\mathit{mix}, \top).(\mathit{bake}, \top).(\mathit{eat}, \top).\mathit{Cake}\\
\mathit{Request} &\overset{def}{=} (\mathit{want}, \top).(\mathbf{request}, q).(\mathit{recipe}, c).\mathit{Request}'\\
\mathbf{LH} &\overset{def}{=} \mathit{Louise} \underset{\{want, eat\}}{\bowtie} \Big((\mathit{Oven} \parallel \mathit{Recipe}\,[\_]) \underset{\{mix, bake\}}{\bowtie} \mathit{Cake}\Big) \underset{\{want\}}{\bowtie} \mathit{Request}[\mathit{Request}]\\
\mathbf{CH} &\overset{def}{=} \mathit{Request}[\_] \underset{\{recipe\}}{\bowtie} \mathit{Recipe}[\mathit{Recipe}]\\
\ell(t_1) &= \mathbf{request},\ \ell(t_2) \ = \ \mathbf{relate}
\end{aligned}$$

Here the recipe is a token as it must be communicated from one location (**CH**) to another (**LH**). First, however this movement must be triggered. This is achieved by having another token, of type *Request*, which is sent from Louise's house to Christine's house. The static component representing the resource, the oven, is located at Louise's house and the chocolate cake is also a static component here.

Mobile Agent. *Louise wants to prepare a chocolate cake. She has the right recipe and ingredients, but she does not have an oven. However, she knows that Christine has an oven, and that she is happy to lend it. So, she prepares the cake batter and then goes to Christine's home, where she bakes the cake.*

$$\begin{aligned}
\mathit{Louise} &\overset{def}{=} (\mathit{want}, w).(\mathit{eat}, e).\mathit{Louise}\\
\mathit{Recipe} &\overset{def}{=} (\mathit{mix}, m).\mathit{Recipe} \qquad\qquad \mathit{Oven} \overset{def}{=} (\mathit{bake}, b).\mathit{Oven}\\
\mathit{Cake} &\overset{def}{=} (\mathit{want}, \top).(\mathit{mix}, \top).(\mathbf{take}, t).(\mathit{bake}, \top).(\mathbf{return}, u).(\mathit{eat}, \top).\mathit{Cake}\\
\mathbf{LH} &\overset{def}{=} (\mathit{Louise} \parallel \mathit{Recipe}) \underset{\{want, mix, eat\}}{\bowtie} \mathit{Cake}[\mathit{Cake}] \qquad \mathbf{CH} \overset{def}{=} \mathit{Cake}[\_] \underset{\{bake\}}{\bowtie} \mathit{Oven}\\
\ell(t_1) &= \mathbf{take},\ \ell(t_2) \ = \ \mathbf{return}
\end{aligned}$$

In this PEPA net, as in the first, the cake is the only token; the placement of the static components representing resources, however, differs. In this case the *Recipe* component is in the place representing Louise's house, while the *Oven* is located at Christine's house. As previously the *Cake* moves to fulfil its evolution. The difference is simply the locations in which the different stages are now exhibited.

We define a fresh scenario corresponding to the emerging paradigm of *Web services*, in which a service must first be located via a discovery service such as UDDI before invocations are made.

Web Service. *Louise would like to have a chocolate cake, but she doesn't know the recipe, and she does not have at home either the required ingredients or an*

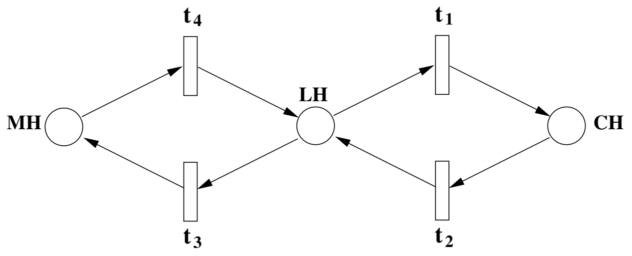

Fig. 4. Net structure for PEPA net model in the Web service chocolate cake scenario

oven, nor does she know anyone to ask. She asks her friend Marian, who knows everyone's capabilities. Marian tells her that Christine knows how to make a chocolate cake, and that she has a well-supplied kitchen at her place. Marian also gives Louise Christine's phone number and Louise phones Christine asking: "Can you make me a chocolate cake, please?" Christine makes the chocolate cake and delivers it back to Louise.

$$Louise \overset{def}{=} (want, w).(eat, e).Louise$$

$$Marian \overset{def}{=} (cake\_query, q).Marian$$

$$Recipe \overset{def}{=} (mix, m).Recipe \qquad Oven \overset{def}{=} (bake, b).Oven$$

$$Query \overset{def}{=} (want, \top).(\mathbf{ask}, a).(cake\_query, \top).(\mathbf{reply}, r).(who, w).Query$$

$$Cake \overset{def}{=} (who, \top).(\mathbf{phone}, p).(mix, \top).(bake, \top).(\mathbf{deliver}, d).(eat, \top).Cake$$

$$\mathbf{MH} \overset{def}{=} Marian \underset{\{cake\_query\}}{\bowtie} Query[\_] \qquad \mathbf{CH} \overset{def}{=} Cake[\_] \underset{\{mix, bake\}}{\bowtie} (Recipe \parallel Oven)$$

$$\mathbf{LH} \overset{def}{=} Louise \underset{\{want, eat\}}{\bowtie} \Big(Query[Query] \underset{\{who\}}{\bowtie} Cake[Cake]\Big)$$

$$\ell(t_1) = \mathbf{phone},\ \ell(t_2) = \mathbf{deliver},\ \ell(t_3) = \mathbf{ask},\ \ell(t_4) = \mathbf{reply}$$

The scenario now involves three locations, as shown in Figure 4. In addition to the places for Louise's house and Christine's house, we have an additional place for the resource represented by the static component *Marian*. There are two tokens in the system: the *Cake* which follows a similar evolution to in the client-server scenario; and *Query* which carries out the discovery, initiated by Louise and satisfied by Marian. Once the query has been resolved, the two tokens synchronise (in the place **LH**) to pass on the information, which the cake can then use to determine its next move.

Whilst these scenarios are simple in terms of the number of components and locations involved, they do capture five key paradigms and we have shown that each can be readily modelled using PEPA nets. More realistic systems might be expected to entail a greater number of components and locations but the patterns of interaction, communication and movement would be the same, meaning that they could also be captured by PEPA nets.

4 Conclusions

Mobility and mobile agents pose interesting problems for modellers. New formalisms are emerging for modelling them. In this paper we have presented an introduction to one such formalism, PEPA nets. We have introduced PEPA nets in some detail and considered their expressiveness with respect to a published classification of mobile code design paradigms.

Several process calculi for modelling mobile computation, primarily for the purpose of functional verification, e.g., the π-calculus [8] (and its stochastic version [9]) and the calculus of mobile ambients [1] have appeared in the literature. There have also been extensions of Petri nets based on the use of "nets within nets", e.g., *Elementary Object Systems* [10] and *Reference nets* [7]. It is an area for future work to study the differences and similarities between these formalisms and PEPA nets.

References

1. Cardelli, L., Gordon, A.D.: Mobile Ambients. In: Nivat, M. (ed.): Foundations of Software Science and Computational Structures. Lecture Notes in Computer Science, Vol. 1378. Springer, Berlin Heidelberg New York (1998) 140–155
2. Fuggetta, A., Picco, G.P., Vigna, G.: Understanding Code Mobility. IEEE Transactions on Software Engineering **24** (1998) 342–361
3. Gilmore, S., Gribaudo, M.: Graphical Modelling of Process Algebras with DrawNET. In: Bause, F. (ed.): The Tools of the 2003 Illinois Multiconference on Measurement, Modelling, and Evaluation of Computer-Communication Systems. Research Report 781, Universität Dortmund, Informatik IV. (2003) 1–4
4. Gilmore, S., Hillston, J., Kloul, L., Ribaudo, M.: PEPA Nets: A Structured Performance Modelling Formalism. Performance Evaluation **54** (2003) 79–104
5. Gilmore, S., Hillston, J., Kloul, L., Ribaudo, M.: Software Performance Modelling using PEPA Nets. In: Proceedings of the 4th International Workshop on Software and Performance, California. (2004) 13–23
6. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press (1996)
7. Köhler, M., Moldt, D., Rölke, H.: Modelling the Structure and Behaviour of Petri Nets Agents. In: Proceedings of the International Conference on Applications and Theory of Petri Nets. Lecture Notes in Computer Science, Vol. 2075. Springer, Berlin Heidelberg New York (2001) 224–241
8. Milner, R.: Communicating and Mobile Systems: The π Calculus. Cambridge University Press (1999)
9. Priami, C.: Stochastic π Calculus. In: Gilmore, S., Hillston, J. (eds.): Proceedings of the Third International Workshop on Process Algebras and Performance Modelling. Special Issue of The Computer Journal **38** (1995) 578–589
10. Valk, R.: Petri Nets as Token Objects—An Introduction to Elementary Object Nets. In: Desel, J., Silva, M. (eds.): Proceedings of the 19th International Conference on Application and Theory of Petri Nets, Portugal. Lecture Notes in Computer Science, Vol. 1420. Springer, Berlin Heidelberg New York (1998) 1–25