PEPA nets in practice: modelling a decentralised peer-to-peer emergency medical application

Stephen Gilmore, Valentin Haenel, Jane Hillston, and Leïla Kloul

Laboratory for Foundations of Computer Science
The University of Edinburgh, Scotland.

Abstract. We apply the PEPA nets modelling language to modelling a peer-to-peer medical informatics application, the FieldCare PDA-based medical records system developed by SINTEF Telecom and Informatics, Norway. Medical data on accident victims is entered by medics on handheld devices at the crash site and propagated wirelessly from peer to peer in order to improve information flow and reduce the potential for data loss. The benefits of such a system include improved reliability in patient care and the ability for hospitals to prepare better for incoming trauma patients. The effectiveness and usefulness of the system in practice depends upon both reliability and performance issues. We analyse the functioning of the application through a high-level model expressed in the PEPA nets modelling language, a coloured stochastic Petri net in which the tokens are terms of Hillston's Performance Evaluation Process Algebra (PEPA). We use the PRISM probabilistic model checker to solve the model and evaluate probabilistically quantified formulae which quantify the responsiveness of the system.

1 Introduction

Public medical emergencies occur when trains are derailed or when aeroplanes or other multi-passenger vehicles crash. Doctors and paramedics are dispatched to the crash site in order to assess the severity of injuries, dispense medication and provide best-effort treatment for the victims in situ. Typically the majority of the most seriously-injured patients are transported by medical ambulance to a hospital for further treatment. With them is sent a transcript of the treatment administered at the crash site together with the assessment of the attending physician of the severity of the injuries. The record of medicines dispensed is a vital piece of documentation, preventing potentially injurious overdosing with another drug. Similarly the assessment of the on-site physician contains very important data, facilitating speedy prioritisation of the incoming patients.

At present this documentation is most often sent as paper records or cards which are attached to the patient's clothing. The intention is that when the patient reaches hospital they, and their records, can be transferred into the care of another doctor. In practice the process of documenting treatment in this way does not always work well. Records and cards can be lost or damaged or unreadable and the only copy of important medical data is compromised. This usually leads to a predictably poor outcome—overdosing a patient with an already-administered drug, or incorrectly prioritising patients when determining treatment order and thereby lessening the chances of survival of a badly-injured patient.

Medical informatics researchers are working to replace these systems with technologically superior ones. One such effort is the *FieldCare* project, initiated by SINTEF Telecom and Informatics, Norway [1]. The intention of this project is to equip emergency rescue medics in Norway with handheld PDAs on which is recorded the relevant patient information. This information is then replicated from PDA to PDA in peer-to-peer fashion in order to improve the robustness of the information storage. The functioning principle of the system is that all data should be replicated to every potential medical site. Patient care is often handed on from one doctor to another rapidly and poor information flow can be a cause of errors. The propagation of information through the system allows hospital staff to make appropriate preparations in advance of the arrival of patients at hospitals.

The timely operation of a mobile computing system such as this is a significant concern. In this paper we present a model of the FieldCare system implemented as a PEPA net [2], and present an analysis of the system based on this model. PEPA nets are ideally suited to represent a system in which the components of the system find themselves working in a changing context due to mobility. Our analysis is based on the PRISM stochastic model checking tool [3] which allows us to express desirable features of the model in Continuous Stochastic Logic (CSL) [4], a logical specification language which incorporates time- and probability-bounded operators.

The remainder of the paper is organised as follows. In the following section we give a description of the PEPA nets formalism, and its tool support. This includes an explanation of how a PEPA net model can be used as input to the PRISM model checker. In Section 3 we describe the FieldCare system in more detail and present the PEPA net representation of it. The analysis of this system, is given in Section 4, together with a brief overview of the CSL stochastic logic which we use to express desirable properties of the model. Finally, in Section 5, we discuss our conclusions.

2 PEPA nets

In this section we provide a brief overview of PEPA nets and the PEPA stochastic process algebra. A fuller description, together with supporting theory and proofs is available in [2] and [5]. The purpose of this summary is to provide enough information about the modelling language to make the present paper self-contained.

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 the interactions between them. 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*(). A PEPA model shows the order in which such methods can be invoked.

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

Every activity in the model 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).

Such a description 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”.

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 *cooperation set* L in this case is $\boldsymbol{\mathcal{A}}(\mathit{File})$, the *complete action type set* of the component, ($\mathit{openRead}$, $\mathit{openWrite}$, $\ldots$). This place has a *File*-type cell and a static component, *FileReader*, which can process the file when it arrives. When components cooperate in this way it will usually be the case that one is the active participant (which determines the rate at which the activity is performed) and the other is the passive participant (who is delayed until the activity completes, and cannot hurry its completion). The PEPA notation to denote the passive participant in a cooperation is to use the distinguished symbol $\top$ in place of their rate variable. Thus (a, r) and $(a, \top)$ can cooperate over a to produce a *shared activity* (a, r). The case where two active participants cooperate is defined in [5].

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 is denoted 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.

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

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.

$$
\begin{array}{rlr}
N & ::= D^{+}M & \text{(net)} \\[4pt]
M & ::= (M_{\mathbf{P}}, \ldots) & \text{(marking)} \\
M_{\mathbf{P}} & ::= \mathbf{P}[C, \ldots] & \text{(place marking)} \\[4pt]
D & ::= I \stackrel{\mathit{def}}{=} S & \text{(component defn)} \\
 & \mid \ \mathbf{P}[C] \stackrel{\mathit{def}}{=} P[C] & \text{(place defn)} \\
 & \mid \ \mathbf{P}[C, \ldots] \stackrel{\mathit{def}}{=} P[C] \underset{L}{\bowtie} P & \text{(place defn)}
\end{array}
\qquad
\begin{array}{rlr}
P & ::= P \underset{L}{\bowtie} P & \text{(cooperation)} \\
 & \mid \ P/L & \text{(hiding)} \\
 & \mid \ P[C] & \text{(cell)} \\
 & \mid \ I & \text{(identifier)} \\[4pt]
C & ::= \text{`\_'} & \text{(empty)} \\
 & \mid \ S & \text{(full)} \\[4pt]
S & ::= (\alpha, r).S & \text{(prefix)} \\
 & \mid \ S + S & \text{(choice)} \\
 & \mid \ I & \text{(identifier)}
\end{array}
$$

Fig. 1. The syntax of PEPA nets

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 and $\mathcal{T}$ is a finite set of net transitions;*
- *$I : \mathcal{T} \to \mathcal{P}$ is the input function and $O : \mathcal{T} \to \mathcal{P}$ is the output function;*
- *$\ell : \mathcal{T} \to (\mathcal{A}_f, \mathbb{R}^+ \cup \{\top\})$ is the labelling function, which assigns a PEPA activity to each transition. The rate determines the negative exponential distribution governing the delay associated with the transition;*
- *$\pi : \mathcal{A}_f \to \mathbb{N}$ is the priority function which assigns priorities (represented by natural numbers) to firing action types;*
- *$\mathcal{C} : \mathcal{P} \to P$ 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, given in [2], gives a precise definition of the possible evolution of a PEPA net, and shows how a Continuous-Time Markov Chain (CTMC) can be derived, treating each marking as a distinct state.

The firing rule for PEPA nets is a natural extension of the classical firing rule. A transition may fire if there is an input token corresponding to each input arc which can perform a firing activity of the appropriate type. However in addition we require that there is a vacant cell of the correct type corresponding to each output arc of the transition (see [6] for details).

2.1 Tool support for PEPA nets

We see the provision of tool support for modelling languages as being analogous to providing tools for programming languages. A software development kit used by a software engineer typically provides a range of tools (compilers, debuggers, profilers, perhaps even model checkers) which perform various types of analysis or conversion on the program. Similarly a model development kit used by a performance engineer contains steady-state and transient solvers, passage-time analysers, model-checkers and other tools which implement a range of analyses for models in the modelling language used.

PEPA nets are directly supported by the PEPA Workbench for PEPA nets, ML Edition [2]. This extends the PEPA Workbench [7] to implement the semantics of the PEPA nets language directly.

Since there already exists comprehensive tool support for PEPA [7, 8, 3, 9] we have recently developed a compiler for PEPA nets which carries out a translation from a PEPA net to a PEPA model [6]. This has the effect of removing the locations within the model and encoding them implicitly within state and activity names. The benefit of this approach is that it allows us to exploit existing tools for PEPA, such as the PRISM probabilistic model checker [3].

In order to define and analyse a model, PRISM requires two input files: a *description of the system* under investigation expressed in a language of reactive modules and a *set of properties* to be checked against it. To integrate PEPA, we built a compiler which compiles PEPA into the reactive modules language. Additionally, PRISM was extended to support PEPA's combinators (parallel and hiding). The PRISM implementation is based on the CUDD [10] binary decision diagram (BDD) library which provides multi-terminal BDD data structures and algorithms. In PRISM, analysis of CTMCs is performed through model checking specifications in the probabilistic temporal logic CSL (see Section 4).

3 The FieldCare system and its PEPA net model

3.1 Description

As previously explained, the FieldCare system has been developed in [1] to address the need of medical teams dealing with accidents to share the information about the status of individual patients. The medical teams include the staff providing first help at the accident scene, the coordination staff at the scene, the ambulance staff and dispatchers in central control rooms, and the local hospitals' staff preparing for the arrival of the casualties. The team members record the information about the patients using hand-held and other computers linked together in a wireless network. The network and team will be very dynamic with members joining and leaving at very short notice.

The particular concern of the authors was to investigate how to achieve reliable data replication between the nodes in a network where some users are very mobile, the radio links may suffer intermittent failures and the connection topology between nodes may vary as team members move around, including moving out of range of a base station. To quote [1], "The goal is that all data, about all patients, should be stored on all nodes.". All nodes have the same status: there is no central server. Fortunately, the lifetime of the database is rather short and the volume of data entered by each user will not be great. Because PDAs are diskless, supporting record-based storage, the entire database will have to be stored in main memory in each PDA. The PDAs are not multi-function; they are specialised only to supporting the FieldCare application. There is one such PDA for each team.

Transaction name	Sequence no.	Neighbour 1	...	Neighbour n
(n, 1)	1	*true*	...	*false*
(n, 2)	2	*true*	...	*true*
(m, 1)	3	*false*	...	*true*
⋮	⋮	⋮	...	⋮

Table 1. Transaction table

The reliable data replication approach: basic idea The approach developed consists of replicating everything to everyone. The objective is that all data about all patients should be stored on all nodes. Thus, data entered on one node must be replicated to all nodes including those that are added to the network afterwards. To achieve that each node has a copy of the database and maintains its own transaction table.

The database is assumed to have a very simple data model where each record consists of patient identification number, time stamp, name of the team, and details of the medical observation. All transactions in the database have an identifier which is unique throughout the network. The identifier is generated on the node at which the transaction is originated. Only INSERT operations are allowed on the database; DELETE and MODIFY are not allowed. Furthermore, full database transaction consistency is not necessary; temporary inconsistency between the local copies of the database during data replication is acceptable [1].

The transaction table is unique to the node. It contains one row for each transaction entered in the local copy of the database and shows, for each current neighbour of the node, which transactions it is known to have. To achieve this a column is created and initialised to *false* for each new neighbouring node. Two additional columns are used, one for the unique transaction identifier and one for the transaction sequence number. The identifier is assigned to the transaction by the node where it is originated and the sequence number is local to each node. This corresponds to the position in which the transaction has been entered in the table. Table 1 shows an example.

Transactions available protocol When a new transaction is generated in a node, an identifier is assigned and it is inserted in the local database. A new row is added to the transaction table and the columns corresponding to all current neighbouring nodes are initialised to *false* for this new row. The node sends a *Transaction offer* message to each of its neighbours, to which it expects to receive a *Please send* answer. The node then sends the new transaction using a *Data update* command. When the neighbouring node receives the new patient record, it sends an *I've got it* message. In the original node this triggers a change in the corresponding column value to *true*.

Frequent communication failures are anticipated and because of this expectation there are sporadic updates between neighbouring nodes. Thus each node checks periodically if its current neighbours have all the transactions reported in its own transaction table. The node sends a *Transaction offer* message to those neighbours whose corresponding column for the transaction offered is still set to *false*. The contacted node may respond either with an *I've got it* message because it already has the transaction in its local database or a *Please send* message. In the latter case it receives a *Data update* command to which it answers with an *I've got it* message. Both nodes have then to set the column of the neighbour to *true*.

Moreover, when a node establishes a new node communication after changing its position in the network, both nodes send each other a *"Pleased to meet you"* message and add a new column for each other in their respective transaction table. These nodes have then to offer each other all the transactions in their table using the exchanges of messages described above.

The periodic check allows the network to provide a solution to the problem of possible losses, like the loss of an aknowledgement message *I've got it*.

3.2 The PEPA net model

Consider two geographical areas, AREA 1 and AREA 2. The former being, for example, the area where an accident took a place and the latter the location of the hospital. Moreover, consider three medical teams A, B and C moving from one area to the other. The PEPA net model corresponding to this situation is described in Figure 2.

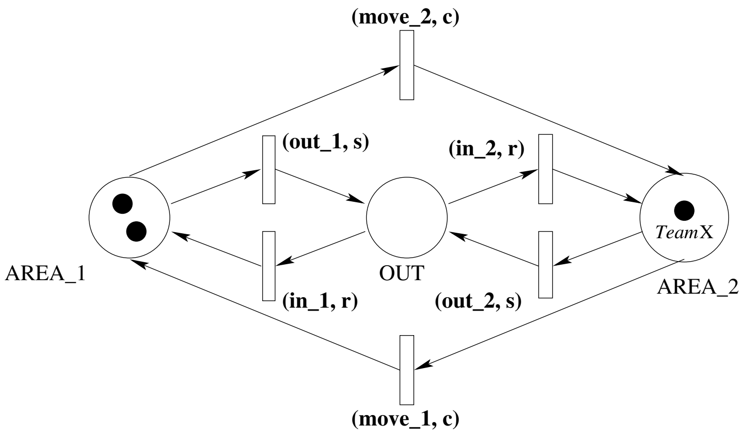

Fig. 2. The PEPA net Model

The PEPA net model consists of three places, namely *AREA*_1, *AREA*_2 and *OUT*. Place *OUT* is used to model the area where a team is out of range of the base station of either AREA 1 or AREA 2. To model the behaviour of the different teams, we use components *Team*X where X stands for A, B or C. Moreover a static component *Neighbour* is associated with each place of the PEPA net network. This component allows us to keep track of the local copies of the database which have already been updated and the ones which still have to be updated.

Component *Team*X This component models the behaviour of a node X representing a medical team in the wireless network. Initially a node may generate a transaction, receive a transaction offer or a *"Pleased to meet you"* message from another node Y. These are modelled using activities *generate_trans*, *trans_offer*$_{YX}$ and *pleased*$_{YX}$ respectively. The two latter activities have an unspecified rate ($\top$) as they are initiated by another node (component). Furthermore a node may move from one area to another, behaviour that is captured by the firing activity **move_i** where i is the area number where it moves to.

Once a new transaction is generated and inserted (*insert_trans*) into the local transaction table of the node, it is offered, with activity *trans_offer*, and then sent (with activity *please_send*) to all immediate neighbours of the node, that is all the nodes in the same area (place). Activity *end_neighbours*_X allows us to know when all neighbours of node X have been updated with the new transaction. This activity has an unspecified rate as it is controlled by component *Neighbour* presented later.

The periodic check for changes which is done by a node is modelled using the activity *periodic_check*. During this phase of the protocol, the node checks its transaction table using the *check_table* activity and offers a transaction (*trans_offer*) to all its neighbours whose corresponding column has `false` as value.

When a node X moves to a new area it has to establish communication with all nodes already in the area. This is modelled by the activity *pleased*$_{\mathrm{XY}}$. As for the periodic check, the node has then to check its transaction table and to offer the transactions with a `false` value in its line to the corresponding immediate neighbours. Note that in this case the node has to offer all the transactions in its table to all its new immediate neighbours. Indeed when a node moves from one area to a new one, its transaction table does not keep track of its previous neighbours.

When a node is within an area, it may become out of range of the base station of the area. This is captured in the model by the firing activity **out_i** where $i(=1,2)$ is the node's current area.

$$
\begin{aligned}
\mathit{Team}\mathrm{X} &\stackrel{\mathit{def}}{=} (\mathit{generate\_trans}, r).\mathit{Team}\mathrm{X}_1 + \textstyle\sum_{\mathrm{Y}\neq\mathrm{X}}(\mathit{pleased}_{\mathrm{YX}}, \top).\mathit{Team}\mathrm{X}_0 \\
&\quad + (\mathit{periodic\_check}, t_1).\mathit{Team}\mathrm{X}_0 + \textstyle\sum_{\mathrm{Y}\neq\mathrm{X}}(\mathit{trans\_offer}_{\mathrm{YX}}, \top).\mathit{Team}\mathrm{X}_6 \\
&\quad + (\mathbf{move\_1, c}).\mathit{Team}\mathrm{X}_M + (\mathbf{move\_2, c}).\mathit{Team}\mathrm{X}_M
\end{aligned}
$$

$$
\begin{aligned}
\mathit{Team}\mathrm{X}_M &\stackrel{\mathit{def}}{=} \textstyle\sum_{\mathrm{Y}\neq\mathrm{X}}(\mathit{pleased}_{\mathrm{XY}}, w).\mathit{Team}\mathrm{X}_M + (\mathit{end\_neighbours\_X}, \top).\mathit{Team}\mathrm{X}_0 \\
&\quad + (\mathbf{out\_1, s}).\mathit{Team}\mathrm{X}_{M_{\mathit{out}1}} + (\mathbf{out\_2, s}).\mathit{Team}\mathrm{X}_{M_{\mathit{out}2}} \\
\mathit{Team}\mathrm{X}_0 &\stackrel{\mathit{def}}{=} (\mathit{check\_table}, c_2).\mathit{Team}\mathrm{X}_2 + \textstyle\sum_{\mathrm{Y}\neq\mathrm{X}}(\mathit{trans\_offer}_{\mathrm{YX}}, \top).\mathit{Team}\mathrm{X}_{6a} \\
&\quad + (\mathbf{out\_1, s}).\mathit{Team}\mathrm{X}_{0_{\mathit{out}1}} + (\mathbf{out\_2, s}).\mathit{Team}\mathrm{X}_{0_{\mathit{out}2}}
\end{aligned}
$$

$$
\mathit{Team}\mathrm{X}_1 \stackrel{\mathit{def}}{=} (\mathit{insert\_trans}, r_1).\mathit{Team}\mathrm{X}_2
$$

$$
\begin{aligned}
\mathit{Team}\mathrm{X}_2 &\stackrel{\mathit{def}}{=} \textstyle\sum_{\mathrm{Y}\neq\mathrm{X}}(\mathit{trans\_offer}_{\mathrm{XY}}, r_2).\mathit{Team}\mathrm{X}_3 + (\mathit{end\_neighbours\_}\mathrm{X}, \top).\mathit{Team}\mathrm{X} \\
&\quad + (\mathbf{out\_1, s}).\mathit{Team}\mathrm{X}_{2_{\mathit{out}1}} + (\mathbf{out\_2, s}).\mathit{Team}\mathrm{X}_{2_{\mathit{out}2}} \\
\mathit{Team}\mathrm{X}_3 &\stackrel{\mathit{def}}{=} \textstyle\sum_{\mathrm{Y}\neq\mathrm{X}}(\mathit{please\_send}_{\mathrm{YX}}, \top).\mathit{Team}\mathrm{X}_4 + \textstyle\sum_{\mathrm{Y}\neq\mathrm{X}}(\mathit{I've\_got\_it}_{\mathrm{YX}}, \top).\mathit{Team}\mathrm{X}_5 \\
&\quad + (\mathbf{out\_1, s}).\mathit{Team}\mathrm{X}_{3_{\mathit{out}1}} + (\mathbf{out\_2, s}).\mathit{Team}\mathrm{X}_{3_{\mathit{out}2}}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{Team}\mathrm{X}_4 &\overset{\mathit{def}}{=} \textstyle\sum_{\mathrm{Y}\neq\mathrm{X}}(\mathit{database\_update}_{\mathrm{XY}}, r_3).\mathit{Team}\mathrm{X}_3 + (\mathbf{out\_1, s}).\mathit{Team}\mathrm{X}_{4_{out}1} \\
&\quad + (\mathbf{out\_2, s}).\mathit{Team}\mathrm{X}_{4_{out}2} \\
\mathit{Team}\mathrm{X}_5 &\overset{\mathit{def}}{=} (\mathit{set\_true\_receiver}, r_4).\mathit{Team}\mathrm{X}_2
\end{aligned}
$$

$$
\begin{aligned}
\mathit{Team}\mathrm{X}_6 &\overset{\mathit{def}}{=} \textstyle\sum_{\mathrm{Y}\neq\mathrm{X}}(\mathit{please\_send}_{\mathrm{XY}}, r_6).\mathit{Team}\mathrm{X}_7 + \sum_{\mathrm{Y}\neq\mathrm{X}}(\mathit{I've\_got\_it}_{\mathrm{XY}}, r_5).\mathit{Team}\mathrm{X} \\
&\quad + (\mathbf{out\_1, s}).\mathit{Team}\mathrm{X}_{6_{out}1} + (\mathbf{out\_2, s}).\mathit{Team}\mathrm{X}_{6_{out}2} \\
\mathit{Team}\mathrm{X}_7 &\overset{\mathit{def}}{=} \textstyle\sum_{\mathrm{Y}\neq\mathrm{X}}(\mathit{database\_update}_{\mathrm{YX}}, \top).\mathit{Team}\mathrm{X}_8 + (\mathbf{out\_1, s}).\mathit{Team}\mathrm{X}_{7_{out}1} \\
&\quad + (\mathbf{out\_2, s}).\mathit{Team}\mathrm{X}_{7_{out}2} \\
\mathit{Team}\mathrm{X}_8 &\overset{\mathit{def}}{=} (\mathit{set\_true\_sender}, r_7).\mathit{Team}\mathrm{X}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{Team}\mathrm{X}_{6a} &\overset{\mathit{def}}{=} \textstyle\sum_{\mathrm{Y}\neq\mathrm{X}}(\mathit{please\_send}_{\mathrm{XY}}, r_6).\mathit{Team}\mathrm{X}_{7a} + \sum_{\mathrm{Y}\neq\mathrm{X}}(\mathit{I've\_got\_it}_{\mathrm{XY}}, r_5).\mathit{Team}\mathrm{X}_{8a} \\
&\quad + (\mathbf{out\_1, s}).\mathit{Team}\mathrm{X}_{a6_{out}1} + (\mathbf{out\_2, s}).\mathit{Team}\mathrm{X}_{a6_{out}2} \\
\mathit{Team}\mathrm{X}_{7a} &\overset{\mathit{def}}{=} \textstyle\sum_{\mathrm{Y}\neq\mathrm{X}}(\mathit{database\_update}_{\mathrm{YX}}, \top).\mathit{Team}\mathrm{X}_{8a} + (\mathbf{out\_1, s}).\mathit{Team}\mathrm{X}_{a7_{out}1} \\
&\quad + (\mathbf{out\_2, s}).\mathit{Team}\mathrm{X}_{a7_{out}2} \\
\mathit{Team}\mathrm{X}_{8a} &\overset{\mathit{def}}{=} (\mathit{set\_true\_sender}, r_7).\mathit{Team}\mathrm{X}_0
\end{aligned}
$$

In component *Team*X the derivatives $\mathit{Team}\mathrm{X}_{k_{out}i}$ are defined as follows

$$\mathit{Team}\mathrm{X}_{k_{out}i} \overset{\mathit{def}}{=} (\mathbf{in\_i, r}).\mathit{Team}\mathrm{X}_k$$

indicating that for a node which has moved out of range, when it returns it does so in the same state. Firing activity **in_i** models the case where the node becomes reachable again. That is, it is no longer out of range.

Component *Neighbour* Component *Neighbour* allows us to keep track of the nodes which have received a transaction offer and the ones which have not. Similarly it allows us to identify the current node(s) of an area with which a newly arriving node has established communication. The complete behaviour of the component when there are three nodes or teams A, B and C, in the network is as follows:

$$
\begin{aligned}
\mathit{Neighbour} &\overset{\mathit{def}}{=} (\mathit{trans\_offer}_{AB}, \top).\mathit{Next}_{AC} + (\mathit{trans\_offer}_{AC}, \top).\mathit{Next}_{AB} \\
&\quad + (\mathit{trans\_offer}_{BA}, \top).\mathit{Next}_{BA} + (\mathit{trans\_offer}_{BC}, \top).\mathit{Next}_{BC} \\
&\quad + (\mathit{trans\_offer}_{CA}, \top).\mathit{Next}_{CB} + (\mathit{trans\_offer}_{CB}, \top).\mathit{Next}_{CA} \\
&\quad + (\mathit{pleased}_{AB}, \top).\mathit{Next}_{AC} + (\mathit{pleased}_{AC}, \top).\mathit{Next}_{AB} \\
&\quad + (\mathit{pleased}_{BA}, \top).\mathit{Next}_{BC} + (\mathit{pleased}_{BC}, \top).\mathit{Next}_{BC} \\
&\quad + (\mathit{pleased}_{CA}, \top).\mathit{Next}_{CB} + (\mathit{pleased}_{CB}, \top).\mathit{Next}_{CA} \\
&\quad + \textstyle\sum_{x=A,B,C}(\mathit{end\_neighbours\_}X, \rho).\mathit{Neighbour} \\
\mathit{Next}_{AB} &\overset{\mathit{def}}{=} (\mathit{trans\_offer}_{AB}, \top).\mathit{Next} + (\mathit{pleased}_{AB}, \top).\mathit{Neighbour} \\
&\quad + (\mathit{end\_neighbours\_}A, \rho).\mathit{Neighbour} \\
\mathit{Next}_{AC} &\overset{\mathit{def}}{=} (\mathit{trans\_offer}_{AC}, \top).\mathit{Next} + (\mathit{pleased}_{AC}, \top).\mathit{Neighbour} \\
&\quad + (\mathit{end\_neighbours\_}A, \rho).\mathit{Neighbour} \\
\mathit{Next}_{BA} &\overset{\mathit{def}}{=} (\mathit{trans\_offer}_{BA}, \top).\mathit{Next} + (\mathit{pleased}_{BA}, \top).\mathit{Neighbour} \\
&\quad + (\mathit{end\_neighbours\_}B, \rho).\mathit{Neighbour} \\
\mathit{Next}_{BC} &\overset{\mathit{def}}{=} (\mathit{trans\_offer}_{BC}, \top).\mathit{Next} + (\mathit{pleased}_{BC}, \top).\mathit{Neighbour} \\
&\quad + (\mathit{end\_neighbours\_}B, \rho).\mathit{Neighbour}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{Next}_{CA} &\stackrel{\mathit{def}}{=} (\mathit{trans\_offer}_{CA}, \top).\mathit{Next} + (\mathit{pleased}_{CA}, \top).\mathit{Neighbour} \\
&\quad + (\mathit{end\_neighbours\_C}, \rho).\mathit{Neighbour} \\
\mathit{Next}_{CB} &\stackrel{\mathit{def}}{=} (\mathit{trans\_offer}_{CB}, \top).\mathit{Next} + (\mathit{pleased}_{CB}, \top).\mathit{Neighbour} \\
&\quad + (\mathit{end\_neighbours\_C}, \rho).\mathit{Neighbour} \\
\mathit{Next} &\stackrel{\mathit{def}}{=} \textstyle\sum_{x=A,B,C}(\mathit{end\_neighbours\_X}, \rho).\mathit{Neighbour}
\end{aligned}
$$

The *Neighbour* component does not have a physical counterpart in the system but can be regarded as representing the knowledge embodied in the wireless LAN, for purposes such as broadcast messaging.

The initial marking The initial marking of the system we consider is the one corresponding to Figure 2.

$$
\begin{aligned}
\Big(\, &\mathit{Neighbour} \underset{\mathcal{L}}{\bowtie} ((\mathit{TeamA}[\mathit{TeamA}] \underset{\mathcal{K}}{\bowtie} \mathit{TeamB}[\mathit{TeamB}]) \underset{\mathcal{K}'}{\bowtie} \mathit{TeamC}[\_]), \\
&\mathit{Neighbour} \underset{\mathcal{L}}{\bowtie} ((\mathit{TeamA}[\_] \underset{\mathcal{K}}{\bowtie} \mathit{TeamB}[\_]) \underset{\mathcal{K}'}{\bowtie} \mathit{TeamC}[\mathit{TeamC}]), \\
&\qquad\qquad \mathit{TeamA}[\_] \underset{\emptyset}{\bowtie} \mathit{TeamB}[\_] \underset{\emptyset}{\bowtie} \mathit{TeamC}[\_] \Big)
\end{aligned}
$$

The cooperation sets $\mathcal{K}$, $\mathcal{K}'$ and $\mathcal{L}$ are defined as follows:

$$
\begin{aligned}
\mathcal{K} &= \{\mathit{trans\_offer}_{AB}, \mathit{trans\_offer}_{BA}, \mathit{please\_send}_{AB}, \mathit{database\_update}_{AB}, \mathit{I've\_got\_it}_{AB}\} \\
\mathcal{K}' &= \{\mathit{trans\_offer}_{AC}, \mathit{trans\_offer}_{BC}, \mathit{trans\_offer}_{CB}, \mathit{trans\_offer}_{CA}, \mathit{please\_send}_{AC}, \\
&\quad \mathit{I've\_got\_it}_{AC}, \mathit{database\_update}_{AC}, \mathit{database\_update}_{BC}, \mathit{please\_send}_{BC}, \mathit{I've\_got\_it}_{BC}\} \\
\mathcal{L} &= \{\mathit{trans\_offer}_{AB}, \mathit{trans\_offer}_{AC}, \mathit{trans\_offer}_{BA}, \mathit{trans\_offer}_{BC}, \mathit{trans\_offer}_{CA}, \\
&\quad \mathit{trans\_offer}_{CB}, \mathit{end\_neighbours\_A}, \mathit{end\_neighbours\_B}, \mathit{end\_neighbours\_C}, \mathit{pleased}_{AB}, \\
&\quad \mathit{pleased}_{AC}, \mathit{pleased}_{BA}, \mathit{pleased}_{BC}, \mathit{pleased}_{CA}, \mathit{pleased}_{CB}\}
\end{aligned}
$$

4 Model analysis

The PRISM model checker supports the analysis of probabilistic and stochastic systems by allowing a modeller to check a logical property against a model. Several logics and several types of model are supported. Recently the CSL logic (Continuous Stochastic Logic) [4] has gained some acceptance as a suitable vehicle for expressing performance and performability measures which can be model checked on a CTMC. A CSL formula expresses an assertion about the performance measures of a model which can then be checked to see whether it is true or not. The syntax of CSL is:

$$
\begin{aligned}
\phi &::= \mathit{true} \mid \mathit{false} \mid a \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \mathcal{P}_{\bowtie p}[\psi] \mid \mathcal{S}_{\bowtie p}[\phi] \\
\psi &::= \mathrm{X}\,\phi \mid \phi\ \mathrm{U}^{I}\ \phi \mid \phi\ \mathrm{U}\ \phi
\end{aligned}
$$

where a is an atomic proposition, $\bowtie\ \in \{<, \leq, >, \geq\}$ is a relational parameter, $p \in [0, 1]$ is a probability, and I is an interval of $\mathbb{R}$. An action-based version of CSL has been defined [11] but is not supported by PRISM.

Paths of interest through the states of the model are characterised by the *path formulae* specified by $\mathcal{P}$. Path formulae either refer to the next state (using the X operator), or record that one proposition is always satisfied until another is achieved (the until-formulae use the U-operator). Performance information is encoded into the CSL formulae via the time-bounded until operator (U^I) and the steady-state operator, $\mathcal{S}$.

It is sometimes convenient to introduce some *derived operators* in order to help with the expression of a CSL formula. These operators do not add to the expressive power of the logic, they simply help to make the statements of some formulae more compact. One such operator is *implication* ($\Rightarrow$) which can be defined in the usual way, $\phi_1 \Rightarrow \phi_2 \equiv \neg\phi_1 \vee \phi_2$. Another useful operator is *time-bounded eventually* ($\Diamond^I$, where I is an interval of time) which is defined thus: $\Diamond^I \phi \equiv \mathit{true}\ \mathrm{U}^I\ \phi$. This operator is frequently used with intervals of the form [0, t] to capture formally the concept of "*within t time units*".

These derived operators are used in our present case study in the following way. We wish to capture the situation where information sent by one of the participants in the protocol is received by another within a suitable time frame. In a framework with only one sender and one receiver we can characterise the *sending* and *receiving* events by two atomic propositions, `s` and `r`. Then the property which we are interested in can be expressed in CSL as follows.

$$\Phi \equiv \mathtt{s} \ \Rightarrow \mathcal{P}_{\geq p}[\, \Diamond^{[0,\, t]}\mathtt{r}\,]$$

This is a typical CSL formula, combining time, probability and a path through the system evolution. The above formula characterises all states for which either a send event does not occur or it does occur and then a receive event occurs within time t with probability at least p.

Hence, the CSL model-checking problem which we want to check is that $\mathrm{Sat}(\Phi) = S$, where S is the complete state-space of the model.

4.1 Probabilistic model checking

In model-checking our CSL formula against our model of the FieldCare system we investigated the formula Φ when p held the value 0.9. We found that the critical time-frame where Φ returns significant information is for t between 40 and 54 seconds. For values of t less than 40 the formula is true in just those states where a send event does not occur (178432 of the 182002 states of the model are in this subspace). For values of t greater than 54 the formula is true even if a send event occurs because with probability at least 0.9 the information will be received within 54 seconds (all of the 182002 states of the model are in this subspace). Figure 3 presents these results.

By presenting the number of satisfying states instead of simply a single truth value PRISM guides the modeller through experimentation with the model in order to select subsequent formulae to verify. If the present formula is true in nearly all of the states in the model then the next values for the probabilistic or time bounds might be close to the present ones. If the formula is satisfied in only a small number of states the next values chosen might be further away.

In studying the growth of the satisfying states in the model we see that the curve is not totally smooth. At around 50 seconds there is a small kink in the curve as a slightly larger than usual subset of the state space moves into the satisfying set for the formula. Shortly after the growth of this set slows. Variations such as this are readily explained as being caused by variability in the connectivity of components in the model.

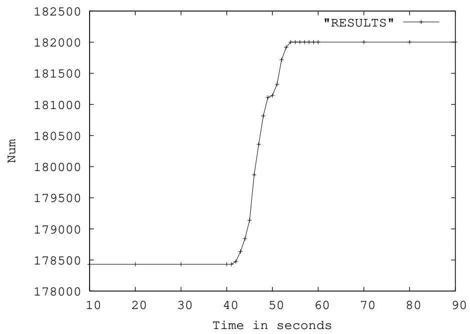

Fig. 3. Plot of satisfying states for the FieldCare model with $p = 0.9$

4.2 Performance analysis

In addition to model-checking the above formulae we undertook a classical performance analysis of the model. We used the sparse matrix engine of the PRISM tool with its implementation of the Jacobi over-relaxation numerical routine to solve the CTMC representation of the model for its stationary probability distribution. This is an efficient procedure, taking (approximately) 400 seconds to compute on a Pentium IV processor running at 1.8GHz with 256MB of physical RAM.

We then focussed on the probability of being in a distinguished subset of the state space, determined formally from our high-level PEPA net model. Figure 4 shows the impact of the increase in rate of movement of team *TeamB* from one area to another on the database update probability of *TeamA*. As the speed of movement increases, the probability P_{BA} of updating the database of *TeamA* decreases. This is the most typical case where one of the teams is more often required to move than the other one.

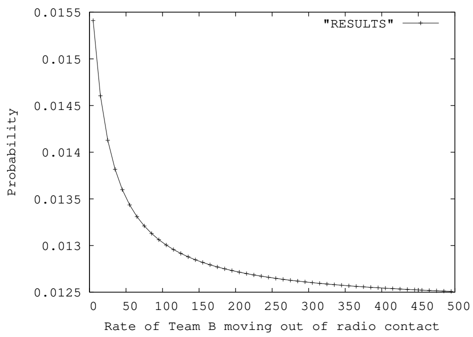

Fig. 4. Plot showing the probability of successfully updating the database decreasing as the rate of movement of teams increases

5 Conclusions

In this paper we have applied the PEPA nets modelling language to the analysis of a model of a real-world application, the FieldCare medical emergency system being developed by SINTEF Telecom and Informatics, Norway. This application will be used in a safety-critical setting and therefore deserves considered analysis and investigation before it is deployed in earnest.

The analysis which we applied to the system is undertaken by making a high-level model, abstracting away much unnecessary detail in order to focus more clearly on the salient aspects of the problem. Building performance models of realistic real-world systems is an activity which requires careful attention to detail in order to model correctly the intended behaviour of the system. Proceeding with care during this part of the modelling process is a wise investment of effort. If the initial performance model contains errors then all of the computational expense incurred in solving the model and all of the intellectual effort invested in the analysis and interpretation of the results obtained would at best be wasted. In general interpreting a model with errors could lead to making flawed decisions based on erroneous conclusions made from erroneous results. This can lead to perhaps classifying systems as being effectively reliable when a high-level model without these flaws would have demonstrated that they are not. For this reason we consider it important to work with structured, high-level modelling languages which directly support the concepts and idioms of the application domain, such as mobility of users and devices.

Using a suitable high-level modelling language, the PEPA nets notation, our analysis focussed on correctly capturing the behaviour of this dynamically-varying system in use and probing the timely behaviour of its key function: propagating critical data to all neighbours in a peer-to-peer application using a simple and robust protocol. Using the PRISM probabilistic model checker as our analysis tool we were able to ascertain how delays on the receiving of propagated medical information would impact on the system in use.

Other analysis tools for PEPA provide complementary analysis capabilities to those which are provided by PRISM. The Imperial PEPA Compiler (IPC) [9] uses the DNAmaca tool [12] to compute passage-time quantiles for passages through the model delimited by a set of starting states and a set of terminating states. This could be used to investigate other measures over the behaviour of the system including quantifying averages and extremes in response times. One of the reasons to model with a well-supported formal language such as PEPA is that a range of analysis options are available.

Many concerns and questions remain about peer-to-peer emergency medial systems such as FieldCare. Our analysis has said nothing about other very important aspects of the application such as resistance to malicious attack by hostile users of other mobile devices in the area. Similarly other analyses would be appropriate. The application is to be implemented using Java technology operating on hand-held devices under far from ideal conditions of use. We have thrown no light on whether or not the implementation technology can withstand this challenge. Aspects such as these remain to be considered carefully by the designers and developers of the system before the system is deployed in earnest in a life-critical situation.

Acknowledgements: The authors are supported by the Design Environments for Global ApplicationS project (DEGAS) IST-2001-32072 funded by the Future and Emerging Technologies Proactive Initiative on Global Computing. The PEPA to PRISM compiler was developed in collaboration with Gethin Norman and Dave Parker of The University of Birmingham. The authors wish to thank Paolo Ballarini for many helpful suggestions on CSL. Leïla Kloul is on leave from PRISM, Université de Versailles, Versailles, France.

References

1. J. Gorman, S. Walderhaug, and H. Kvålen. Reliable data replication in a wireless medical emergency network. In *Proceedings of the 22nd International Conference on Computer Safety, Reliability and Security (SAFECOMP'03)*, number 2788 in LNCS, pages 207–220, Edinburgh, Scotland, September 2003. Springer-Verlag.
2. S. Gilmore, J. Hillston, M. Ribaudo, and L. Kloul. PEPA nets: A structured performance modelling formalism. *Performance Evaluation*, 54(2):79–104, October 2003.
3. M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. In J.-P. Katoen and P. Stevens, editors, *Proc. 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, volume 2280 of *LNCS*, pages 52–66. Springer, April 2002.
4. A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Verifying continuous time Markov chains. In *Computer-Aided Verification*, volume 1102 of *LNCS*, pages 169–276. Springer-Verlag, 1996.
5. J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.
6. S. Gilmore, J. Hillston, L. Kloul, and M. Ribaudo. Software performance modelling using PEPA nets. In *Proceedings of the Fourth International Workshop on Software and Performance*, pages 13–24, Redwood Shores, California, USA, January 2004. ACM Press.
7. S. Gilmore and J. Hillston. The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In *Proceedings of the Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, number 794 in Lecture Notes in Computer Science, pages 353–368, Vienna, May 1994. Springer-Verlag.
8. G. Clark and W.H. Sanders. Implementing a stochastic process algebra within the Möbius modeling framework. In L. de Alfaro and S. Gilmore, editors, *Proceedings of the first joint PAPM-PROBMIV Workshop*, volume 2165 of *Lecture Notes in Computer Science*, pages 200–215, Aachen, Germany, September 2001. Springer-Verlag.
9. J.T. Bradley, N.J. Dingle, S.T. Gilmore, and W.J. Knottenbelt. Derivation of passage-time densities in PEPA models using IPC: The Imperial PEPA Compiler. In G Kotsis, editor, *Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems*, pages 344–351, University of Central Florida, October 2003. IEEE Computer Society Press.
10. F. Somenzi. *CUDD: CU Decision Diagram Package*. Department of Electrical and Computer Engineering, University of Colorado at Boulder, February 2001.
11. R. D. Nicola and F. W. Vaandrager. Action versus state based logics for transition systems. In *Proceedings* Ecole de Printemps *on Semantics of Concurrency*, volume 469 of *Lecture Notes in Computer Science*, pages 407–419. Springer Verlag, 1990.
12. W.J. Knottenbelt. Generalised Markovian analysis of timed transition systems. Master's thesis, University of Cape Town, 1996.