Modelling Role-Playing Games Using PEPA Nets

Stephen Gilmore[1], Leila Kloul[1⋆], and Davide Piazza[2]

[1] LFCS, The University of Edinburgh, Edinburgh EH9 3JZ, Scotland
{stg,leila}@inf.ed.ac.uk
[2] OMNYS Wireless Technology, via Frassini, 35, 36100 Vicenza, Italy
davide.piazza@omnys.it

Abstract. We present a performance modelling case study of a distributed multi-player game expressed in the PEPA nets modelling language. The case study provides a modern complex distributed application programming problem which has many inherent communication and synchronisation complexities which are subtle to model accurately. We put forward the position that a high-level performance modelling language is well-suited to such a task. The structure of the model and the performance index which is most significant for the problem match well a solution method which has previously been applied in Petri net modelling. We apply this method to a PEPA net model for the first time in this paper.

1 Introduction

Performance models of computer and telecommunication systems are used to gain insights into the behaviour of the system under expected workload on the available hardware. The state-of-the-art in the design of computer systems grows ever more sophisticated as programming languages become more complex; application programs increasingly use additional layers of middleware and infrastructure; and software developers deploy complex patterns and idioms to structure their application code. Similarly, the computing platform on which these applications execute becomes more complex. A wide range of computing devices may be deployed, from high-end servers to battery-powered handheld devices. Other layers of interpretation may also be used: virtual machines such as the JVM or hardware emulators as in the Transmeta Crusoe processor. Each such layer adds complexity and degrades performance.

Complex software necessitates the use of a sophisticated modelling language which provides direct support for high-level configuration and re-configuration in addition to offering a behavioural specification language to be used to capture the application logic. The PEPA nets modelling language [5] is one such language, a high-level coloured stochastic Petri net formalism where the tokens of the net

⋆ On leave from PRiSM, Université de Versailles, 45, Av. des Etats-Unis 78000 Versailles, France.

C. Aykanat et al. (Eds.): ISCIS 2004, LNCS 3280, pp. 523–532, 2004.

are themselves programmable stochastically-timed components. The modelling language which is used for the tokens of a PEPA net is Jane Hillston's Markovian process algebra PEPA (Performance Evaluation Process Algebra) [6]. The PEPA nets formalism is a recent research development and we are currently exploring its possibilities in tandem with developing its underlying theory [5]. To this end we have undertaken a number of case studies including the Jini discovery service and a mobile telephony scenario [5]; and the Mobile IP protocol [3]. The PEPA nets formalism is not described in this paper, all details of the PEPA nets and PEPA languages are found in [5] and [6].

In this paper we apply the PEPA nets language to the problem of modelling a complex distributed application, a multi-player online role-playing game. The game is one of the case studies from one of our industrial partners on the EC-funded DEGAS project (Design Environments for Global ApplicationS). The game is a characteristic "global computing" example, encompassing distribution, mobility and performance aspects. The representational challenges in modelling the game accurately include capturing location-dependent collaboration, multi-way synchronisation, and place-bounded locations holding up to a fixed number of tokens only. All of these are directly represented in the PEPA nets formalism.

In Section 2 we provide a high-level description of our modelling study, a role-playing game. In Section 3 we present the PEPA net model of the system. In Section 4 we make an analysis of the model, discuss solution methods for PEPA net models, and present results from our study. Conclusions follow at the end of the paper.

2 A High-Level Description of the Game

The Massive Multi-Player Online Role-playing Game (MMPORG) consists of a succession of game levels of increasing complexity. Each level is composed of a starting point and a certain number of rooms among which is a secret room.

In the game, a player is seen as an active entity who may interact with objects, locations (rooms), playing and non-playing characters of the virtual environment. Objects (weapons, medicine, food, ...) are one of the basic elements of the game environment that a player can collect and reuse later. The player has to explore the rooms to collect as many experience points as possible to improve character features such as strength, skill, luck, etc. Each obstacle or test successfully passed increases the number of experience points. Conversely, each failure decreases the number of points. If this number reaches zero, the player vanishes from the current room and is transferred back to the starting point of the current level. To progress to the next level, a player must find the secret room and pass the tests of this room. If they fail, they are once again transferred to the starting point of the level. The secret room can hold one player only, that is, at most one player can be inside at a time. A player may be in competition with one or several other players in the same room to acquire objects. The winner of a fight between two players earns experience points from the defeated player.

The MMPORG also features non-playing characters. Like the players, non-playing characters are active entities that may move from one room to another but they are confined to a single level and cannot access the secret room. These characters are generated by the rooms themselves. Non-playing characters like monsters are obstacles which a player will have to pass. Fighting is a direct interaction between characters within a room. These interactions are based on system of "cards" which can cause or neutralise some damage. The effect depends on the card features and on the features of the characters involved. Defeating a non-playing character allows the player to increase their current features. The player may acquire new cards and therefore increase their offensive or defensive skills.

All the computations resulting from the different interactions are performed by the rooms. Moreover, when a player selects the next room to visit, this room clones itself and sends its image to the player.

3 The PEPA Net Model

Assuming that L is the number of levels in the game and N_j is the number of rooms at level j, the PEPA net model consists mainly of three types of places: $ROOM_{ji}$, $SECRET\_R_j$ and $INIT\_R_j$ where $j = 1 \dots L$ and $i = 1 \dots N$. Respectively, these model room i, the secret room and the starting point at level j (see Figure 1). We use place OUT to stand for the environment outside the game.

Moreover we consider components *Player*, *NPlayer* and *Room* to model the behaviour of respectively the playing character, the non-playing character and the room.

3.1 The Components

– **Component Player.** Once connected (firing action **connect**), the player starts by choosing one of the rooms of the current level. This is modelled using firing action $\mathbf{select_i}$ with rate $p_i \times r_0$, i being the room number at the current level and p_i the probability to select this room number.
Once the player receives an image of the room, they may do different things: observe, walk, talk to another character (playing or non-playing). They may also try to use one of the objects they have with action type use_{obj} or to take a new one ($take_{obj}$) from the room. In this last case, the system, through the room character, may accept or refuse to let the player take the object using action type $accept_{obj}$ or $refuse_{obj}$. Here the rate of these actions is not specified by the player because the decision is made by the room.
When the player is in the room, they may be attacked by another player (*fightP*) or a non-playing character (*fightNP*). The result of the fight may be either a defeat of the player (*PlossP* or *PlossNP*) or its victory (*PwinP* or *PwinNP*). In the former case, it loses points ($less_{pts}$) if the fight is against another player. If it has no more points ($zero_{pts}$), it is transferred to the starting point of the current level. This is modelled using firing action **failure**.

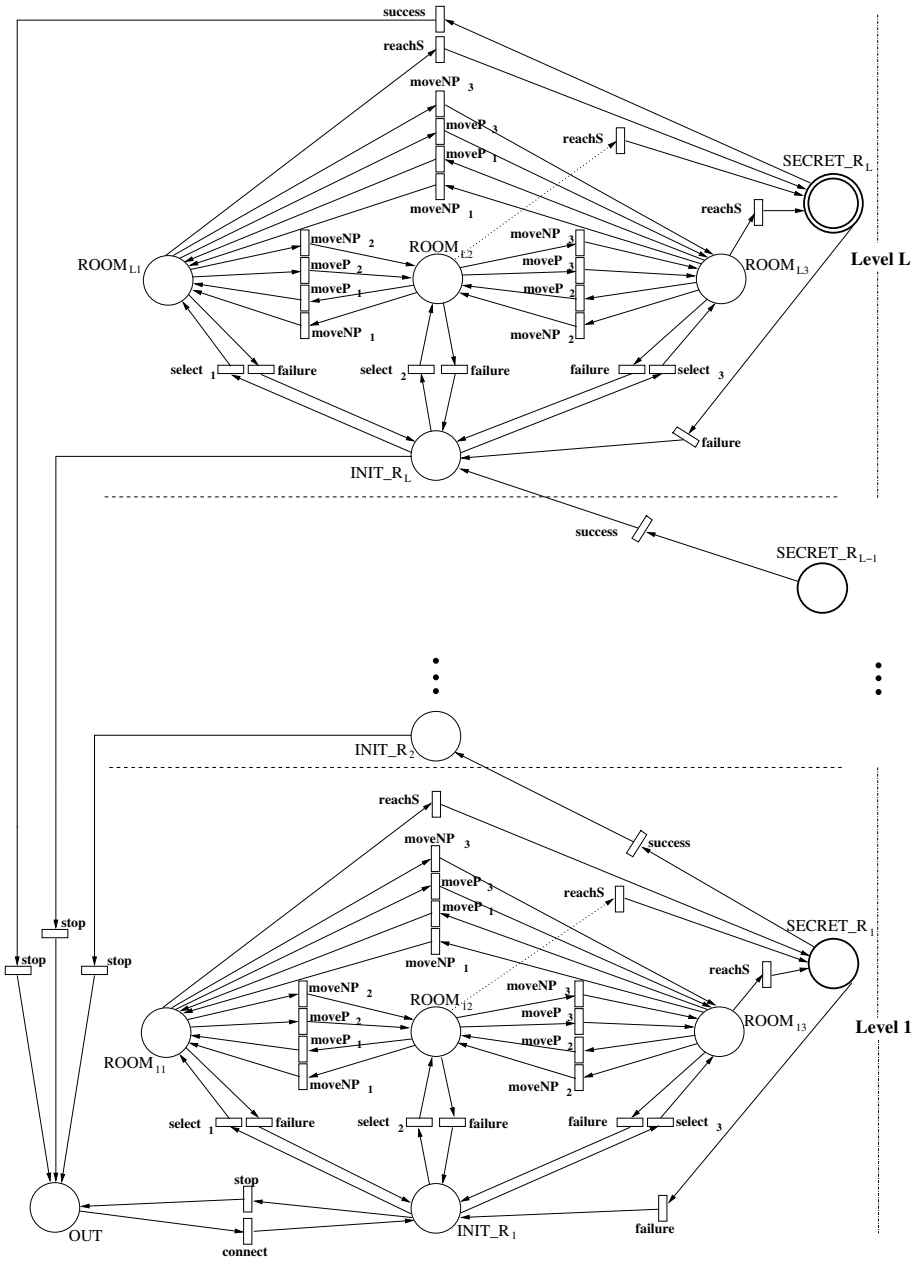

Fig. 1. PEPA net model for $N_j = 3,\ j = 1 \ldots L$

In the latter case, the player gets cards (new_{crd}) if it defeated a non-playing character.
The player may decide to move to another room i with action type $\mathbf{moveP_i}$ and probability q_i, or **reachS** if it finds the secret room. The player may also decide to stop the game at any moment as long as it is in the stating point $INIT\_R_j$ of a level. This is modelled using activity $(\mathbf{stop}, s)$.

$$
\begin{aligned}
Player &\stackrel{def}{=} (\mathbf{connect}, r).Player_0\\
Player_0 &\stackrel{def}{=} \textstyle\sum_{i=1}^{N_j}(\mathbf{select_i}, p_i \times r_0).(RImage, \top).Player_1 + (\mathbf{stop}, s).Player\\
Player_1 &\stackrel{def}{=} (observe, \alpha_1).Player_1 + (walk, \alpha_2).Player_1 + (talk, \alpha_3).Player_1\\
&\quad +(fightNP, \beta_1).Player_{21} + (fightP, \beta_2).Player_{31}\\
&\quad +(use_{obj}, \delta_1).Player_4 + (take_{obj}, \delta_2).Player_5\\
&\quad + \textstyle\sum_{i=1}^{N_j-1}(\mathbf{moveP_i}, q_i \times r_1).Player_1 + (\mathbf{reachS}, r_2).Player_{70}
\end{aligned}
$$

$$
\begin{aligned}
Player_{21} &\stackrel{def}{=} (PlossNP, \top).Player_{22} + (PwinNP, \top).Player_{23}\\
Player_{22} &\stackrel{def}{=} (less_{pts}, \gamma_1).Player_1 + (zero_{pts}, \gamma_2).Player_6\\
Player_{23} &\stackrel{def}{=} (new_{crd}, \gamma_3).Player_1
\end{aligned}
$$

$$
\begin{aligned}
Player_{31} &\stackrel{def}{=} (PlossP, \top).Player_{32} + (PwinP, \top).Player_{33}\\
Player_{32} &\stackrel{def}{=} (less_{pts}, \gamma_1).Player_1 + (zero_{pts}, \gamma_2).Player_6\\
Player_{33} &\stackrel{def}{=} (get_{pts}, \gamma_4).Player_1
\end{aligned}
$$

$$
\begin{aligned}
Player_4 &\stackrel{def}{=} (less_{pts}, \gamma_1).Player_1 + (get_{pts}, \gamma_4).Player_1 + (zero_{pts}, \gamma_2).Player_6\\
Player_5 &\stackrel{def}{=} (accept_{obj}, \top).Player_1 + (refuse_{obj}, \top).Player_1\\
Player_6 &\stackrel{def}{=} (\mathbf{failure}, f).Player_0
\end{aligned}
$$

$$
\begin{aligned}
Player_{70} &\stackrel{def}{=} (RImage, \top).(test, \beta_3).Player_7\\
Player_7 &\stackrel{def}{=} (win, \top).Player_8 + (lose, \top).Player_6\\
Player_8 &\stackrel{def}{=} (get_{pts}, \gamma_4).(\mathbf{success}, c).Player_0
\end{aligned}
$$

– **Component NPlayer.** Once a non-playing character has been created by a room ($generateNP$), it may walk, use its own objects and meet a playing character. A fight may then follow and as explained before, if the non-playing character is defeated ($PwinNP$), it vanishes from the system (the room), via action type $destroyNP$. If it wins, it just continues its progression in the rooms of the current game level.

$$
\begin{aligned}
NPlayer &\stackrel{def}{=} (generateNP, \top).NPlayer_1\\
NPlayer_1 &\stackrel{def}{=} (walk, \delta_1).NPlayer_1 + (talk, \top).NPlayer_1 + (fightNP, \delta_2).NPlayer_2\\
&\quad + \textstyle\sum_{i=1}^{N-1}(\mathbf{moveNP_i}, q_i \times v_1).NPlayer_1\\
NPlayer_2 &\stackrel{def}{=} (PlossNP, \top).NPlayer_1 + (PwinNP, \top).NPlayer_3\\
NPlayer_3 &\stackrel{def}{=} (destroyNP, \top).NPlayer
\end{aligned}
$$

– **Component Room.** The room creates and makes vanish the non-playing characters using respectively *generateNP* and *destroyNP*. When it is chosen by a player, the room clones itself and sends an image to them (*RImage*). The room makes also the acceptance ($accept_{obj}$) or the rejection ($refuse_{obj}$) of any attempt of a player to take an object from the location. Moreover it makes all computations related to the fights and sends the results to the characters using action types *PlossP* or *PwinP* and also *PlossNP* and *PwinNP*.

$$\begin{aligned}
Room &\stackrel{def}{=} (generateNP, \sigma_1).Room + (RImage, \sigma).Room + (fightP, \top).Room_2 \\
&\quad +(fightNP, \top).Room_3 + (take_{obj}, \top).Room_1 + (use_{obj}, \top).Room \\
Room_1 &\stackrel{def}{=} (accept_{obj}, \rho_1).Room + (refuse_{obj}, \rho_2).Room \\
Room_2 &\stackrel{def}{=} (PlossP, \phi_1).(PwinP, \phi_2).Room \\
Room_3 &\stackrel{def}{=} (PlossNP, \phi_3).Room + (PwinNP, \phi_4).Room_4 \\
Room_4 &\stackrel{def}{=} (destroyNP, \sigma_2).Room
\end{aligned}$$

– **Component SRoom** models the secret room. It is similar to the other rooms except that at most one player can be inside and non-playing characters are not allowed to get in. Once inside, the player has to pass a different test to get to the higher level.

$$\begin{aligned}
SRoom &\stackrel{def}{=} (RImage, \sigma).(test, \top).SRoom_1 \\
SRoom_1 &\stackrel{def}{=} (lose, \phi_3).SRoom + (win, \phi_4).SRoom
\end{aligned}$$

3.2 The Places

The places of the PEPA net are defined as follows:

$$\begin{aligned}
ROOM_{ji}\ [_, \ldots, _] &\stackrel{def}{=} \left(Room \underset{\mathcal{K}_1}{\bowtie} \left(Player\ [_] \underset{\mathcal{K}_2}{\bowtie} \ldots \underset{\mathcal{K}_2}{\bowtie} Player\ [_] \right) \right) \underset{\mathcal{K}_3}{\bowtie} \\
&\qquad \left(NPlayer\ [_] \parallel \ldots \parallel NPlayer\ [_] \right)
\end{aligned}$$

$$SECRET\_R_j\ [\_, \ldots, \_] \stackrel{def}{=} SRoom \underset{\mathcal{K}_4}{\bowtie} Player\ [\_]$$

$$INIT\_R_j\ [\_, \ldots, \_] \stackrel{def}{=} Player\ [\_] \parallel \ldots \parallel Player\ [\_]$$

$$OUT\ [\_, \ldots, \_] \stackrel{def}{=} Player\ [Player] \parallel \ldots \parallel Player\ [Player]$$

where $i = 1 \ldots N$ is the room number and $j = 1 \ldots L$ is the game level number. The synchronising sets are defined as follows

$$\begin{aligned}
\mathcal{K}_1 &= \{take_{obj}, use_{obj}, accept_{obj}, refuse_{obj}, RImage, fightP, PlossP, PwinP, \\
&\quad fightNP, PlossNP, PwinNP\} \\
\mathcal{K}_2 &= \{fightP\} \\
\mathcal{K}_3 &= \{generateNP, fightNP, PlossNP, PwinNP, destroyNP, talk\} \\
\mathcal{K}_4 &= \{RImage, test, lose, win\}
\end{aligned}$$

4 The Model Analysis

One of the stated aims of the PEPA nets modelling language is to allow analysis techniques and solution methods developed for Petri nets to be used either directly or in adaptation for PEPA nets. The structure of the present model allows us to do exactly this by deploying *flow-equivalent replacement* to bring about a dramatic reduction in the state-space of the model to be solved. The method works in the following fashion for this example. The purpose of the application logic encoded in the PEPA nets model of the MMPORG is to specify in detail the necessary steps to take in order to succeed at each level of the game (and thereby progress to the next level). Once this model has been solved to find the rate at which players progress from one level to the next then the application logic has served its purpose and the sub-model can be replaced by a suitably exponentially-distributed delay, eliding all of the (now) unnecessary detail.

We consider the abstraction of the PEPA net model of Figure 2 where each level j has one input and two output parameters. The input parameter denoted by λ_j represents the arrival rate of the players to the level. The first output parameter denoted by λ_{j+1} is nothing other than the input to the next level $j+1$. This represents the rate of successful players of level j. The second output parameter, noted μ_j, represents the rate of the players leaving the game.

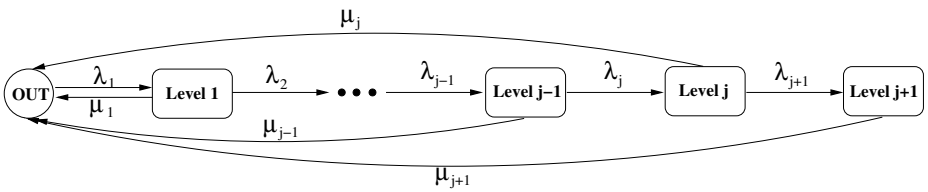

Fig. 2. Abstraction of the PEPA net model

This technique is very well suited to this application because it allows us to evaluate one of the key performance indices of a game application: *difficulty of completion*. If it is possible to progress too quickly from commencing playing to completing the final level of the game then the application may be considered unchallenging. Conversely, if it is very arduous to make progress in the game then the developers risk losing their target audience and finding their game consigned to being suited to hardcore game enthusiasts only.

4.1 Model Solution

We compiled the PEPA net model to an equivalent PEPA model [4]. When processed, this PEPA model will generate the same CTMC as the given PEPA net.

Solution procedures are accessed via the Imperial PEPA Compiler (IPC) [2]. IPC is a functional program written in the Haskell lazy functional programming language [8]. Its function is to compile a PEPA model into the input language of Knottenbelt's DNAmaca analyser [7].

The use of a chain of tools in this way might give the reader concern that this process is laborious or time-consuming and delays progress towards the computation of the performance results which are the real content of the analysis process. This is not the case as the entire end-to-end time of the analysis is itself small (60.41 *sec* for the MMPORG model), making the method suitable for productive interactive performance model development. All measurements were made on a 1.60GHz Intel Pentium IV processor machine with 256 KB of memory, running Red Hat Linux 9.0.

4.2 Model Results

The average activity times associated with the activities of the MMPORG model are shown in Table 1. The rate of each activity is the reciprocal of the average time. A scaling factor, k, initially 0.2, is used to vary the number of repetitions of some of the activities. These distinguished activities double in difficulty in moving from one level to the level above it.

Figure 3 shows the results generated for a game with four levels ($L = 4$). Each graph depicts the cumulative passage-time distribution function for a level of the game. This function is the probability of playing a level, from its starting state to the point where the player succeeds the test of the secret room and reaches the next level, in a certain interval of time.

We used the method of *stochastic probes* [1] to mark the start and end times of the path through the system which we wished to observe. From the graph we can read off high-level performance results such as "50% of the players of the first level will be able to reach the next level in six (6) minutes" and "nearly all players will be able to reach the next level in twenty (20) minutes". Figure 3 shows that things start to become more complicated for the players when they reach the upper levels. It shows that it may take more than ten (10) minutes for 50% of the players of the second level to reach the third one, and less than 35% will be able to finish the last level in less than twenty (20) minutes.

Table 1. Average activity times in the four levels of the MMPORG model

Activity	Level 1	Level 2	Level 3	Level 4
α_i	k × 5 mins.	$2k$ × 5 mins.	$4k$ × 5 mins.	$8k$ × 5 mins.
b_i	6 secs.	6 secs.	6 secs.	6 secs.
β_i	k × 2 mins.	$2k$ × 2 mins.	$4k$ × 2 mins.	$8k$ × 2 mins.
c	1 min.	1 min.	1 min.	1 min.
δ_1	k × 2 mins.	$2k$ × 2 mins.	$4k$ × 2 mins.	$8k$ × 2 mins.
δ_2	k × 6 secs.	$2k$ × 6 secs.	$4k$ × 6 secs.	$8k$ × 6 secs.
f	1 min.	1 min.	1 min.	1 min.
γ_i	6 secs.	6 secs.	6 secs.	6 secs.
h	k mins.	$2k$ mins.	$4k$ mins.	$8k$ min.
$p_i \times r$	3 secs.	3 secs.	3 secs.	3 secs.
$q_i \times r_1$	k × 4 secs.	$2k$ × 4 secs.	$4k$ × 4 secs.	$8k$ × 4 secs.
r_0	1 min.	1 min.	1 min.	1 min.
ρ_i	6 secs.	6 secs.	6 secs.	6 secs.
σ	30 secs.	30 secs.	30 secs.	30 secs.
s	1 min.	1 min.	1 min.	1 min.
φ_i	2 mins.	2 mins.	2 mins.	2 mins.

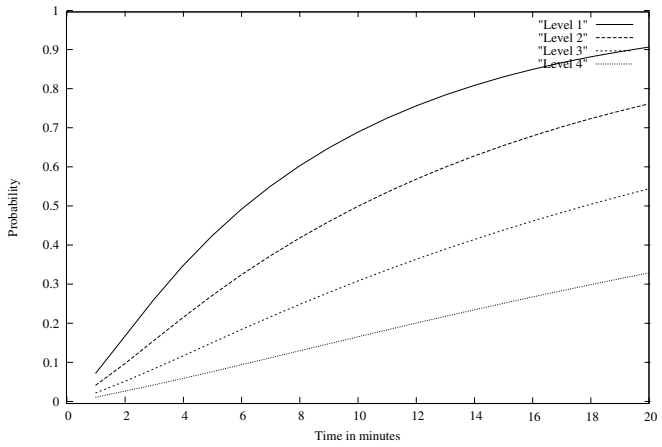

Fig. 3. Cumulative passage-time distribution function for completing each level

Figure 4 depicts the cumulative passage-time distribution function for the complete game, from the first level to the last one. It shows that, considering the parameters values used, less than 14% of the players will be able to go through the game tests in less than 1 hour and 20 minutes.

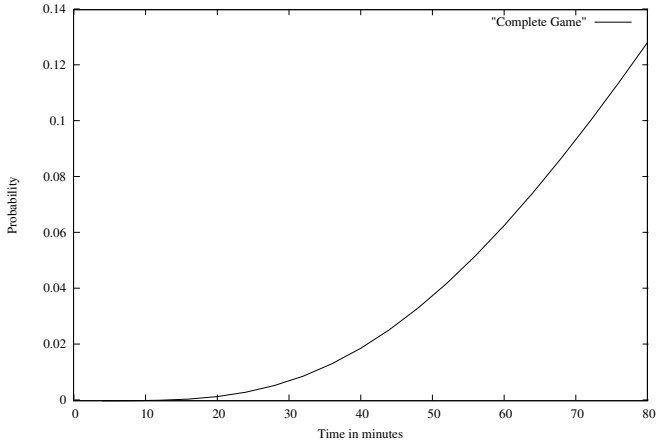

Fig. 4. Cumulative passage-time distribution function for the complete game

The degree of difficulty of completing tasks within the game (fighting other players, collecting objects, and similar tasks) may be changed using different parameters values. Moreover, other graphs of passage-time percentiles for different start and end points in the computation can be obtained by probing different starting points and end points similarly.

5 Conclusions

In this paper we have shown that complex, structured performance models of real-world computing applications can be efficiently solved to give insightful information about their performance behaviour at run-time. The application which we investigated was a mobile multi-player networked game for a handheld device with limited processing power and memory. There are many places for the performance of such an application to fall below the minimum standard expected by a customer who buys such a device. Repairing performance faults in software and firmware after the device has shipped may be cripplingly expensive. For this reason, developers who are concerned with developing marketable, popular products need to address performance-related problems as early as possible. Our methodology strongly supports this by bringing performance analysis capability back to the design phase of product development.

Acknowledgements

The authors are supported by the DEGAS (Design Environments for Global ApplicationS) IST-2001-32072 project funded by the FET Proactive Initiative on Global Computing.

References

1. Bradley, J.T., Dingle, N.J., Argent-Katwala, A.: Expressing Performance Requirements Using Regular Expressions to Specify Stochastic Probes over Process Algebra Models. In: Proceedings of the Fourth International Workshop on Software and Performance, Redwood Shores, California, USA. ACM Press (2004) 49–58
2. Dingle, N.J., Gilmore, S.T., Knottenbelt, W.J., Bradley, J.T.: Derivation of Passage-Time Densities in PEPA Models Using IPC: The Imperial PEPA Compiler. In: Kotsis, G. (ed.): Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems, University of Central Florida. IEEE Computer Society Press (2003) 344-351
3. Gilmore, S., Hillston, J., Kloul, L.: PEPA Nets. In: Calzarossa, M.C., Gelenbe, E. (eds.): Performance Tools and Applications to Networked Systems: Revised Tutorial Lectures. Lecture Notes in Computer Science, Vol. 2965. Springer, Berlin Heidelberg New York (2004) 311–335
4. Gilmore, S., Hillston, J, Kloul, L., Ribaudo, M.: Software Performance Modelling Using PEPA Nets. In: Proceedings of the Fourth International Workshop on Software and Performance, Redwood Shores, California, USA. ACM Press (2004) 13–24
5. Gilmore, S., Hillston, J., Ribaudo, M., Kloul, L.: PEPA Nets: A Structured Performance Modelling Formalism. Performance Evaluation **54** (2003) 79–104
6. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press (1996)
7. Knottenbelt, W.J.: Generalised Markovian Analysis of Timed Transition Systems. Master's thesis, University of Cape Town (1996)
8. Jones, S.P. (ed.): Haskell 98 Language and Libraries: The Revised Report. Cambridge University Press (2003)