A Fluid Flow Approach to Usability Analysis of Multi-user Systems*

Full Version

Mieke Massink[1], Diego Latella[1], Maurice H. ter Beek[1], Michael Harrison[3], and Michele Loreti[2]

[1] Istituto di Scienza e Tecnologie dell'Informazione 'A. Faedo', CNR
Area della Ricerca, Via G. Moruzzi 1, Pisa, Italy
{mieke.massink,diego.latella,maurice.terbeek}@isti.cnr.it
[2] Università di Firenze, Dipartimento di Sistemi e Informatica
Viale Morgagni 65, Firenze, Italy
loreti@dsi.unifi.it
[3] School of Computing Science, Newcastle University
Newcastle upon Tyne, UK
Michael.Harrison@ncl.ac.uk

Abstract. The analysis of usability aspects of multi-user systems, such as cooperative work systems and pervasive systems, pose particular problems because group behavior of their users may have considerable impact on usability. Model-based analysis of such features leads to state-space explosion because of the sheer number of entities to be modeled when automatic techniques such as model checking are used. In this paper we explore the use of a recently proposed scalable model-based technique based on solving sets of Ordinary Differential Equations (ODEs). Starting from a formal model specified using the Performance Evaluation Process Algebra (PEPA), we show how different groupware usage patterns may be modeled and analyzed using this approach. We illustrate how the approach can explore different design options and their impact on group behavior by comparing file access policies in the context of the industrial groupware application thinkteam.

1 Introduction

Tools for usability analysis in relation to one (or at most a few) users are by now relatively mature. However, to date, systematic techniques for analyzing systems where there are many users and where the collective behavior of these users has an influence on the usability of the system are currently undeveloped. Such techniques are becoming more necessary as the variety of cooperative work systems, multi-player games, shared virtual spaces and pervasive systems grows.

Collective behavior may have an impact on the usability of a system as it is perceived by an individual. The effect of the behavior of other users may be to change the individuals user interaction. Consider for example a groupware system that offers exclusive access to files by allowing users to get and lock files when files are available. If the lock is already given to another user, and the file is currently in use, then the user will not be able to access the file until the other user has finished with it. In such situations users devise strategies to ensure that they will have the editing rights that they need when they need them. Alternatively they will schedule their work so that there is always something else that they can do in such circumstances. For example, a strategy that might be feasible in this example would be to get hold of the file some time before it is needed. This greedy strategy would be effective for the individual, making it possible for them

* This work has been partially funded by the Italian MIUR/FIRB project tocai.it, by the EU project Resist/Faerus (IST-2006-026764), by the EU project Sensoria (IST-2005-016004) and by the Italian CNR/RSTL project XXL. thinkteam, think3 and thinkPLM are registered trademarks of think3 Inc. For details: http://www.think3.com.

to carry out their work effectively, but it is not likely to be effective for the whole collaborative activity.

Not only will the individual behavior of a user be affected by changes to the system through its collective use, but the system can also have an effect on the collective behavior of the users. Indeed a system may be designed to achieve precisely this, consider for example a dynamic signage system such as [13] designed to facilitate evacuation of a building. The displays showing where people should go could be designed to change depending on volumes of people within different spaces in the building at any given moment. The displays will together modify the behavior of those in the spaces and thereby, if effective, achieve the most efficient and calm movement of people.

Other factors may affect the usability of these multi-user systems. Usage patterns in relation to technology may also be induced by external factors. For example, in a collaborative design environment it is often the case that the collaboration takes place in a way that reflects project-oriented organization of the work. Projects tend to have different phases: creative phases in which artifacts are developed, which may require longer periods of file creation and modification; fine-tuning phases characterized by frequent but short accesses to a number of critical files. These different phases may lead to a shift between typical usage patterns of the system with a potential impact on its usability characteristics.

Techniques are required that will enable an understanding of both qualitative and quantitative performance aspects of collective usability. In practice few studies have addressed collective behavior. Empirical studies either focus on individual interactions within a system, for example exploring how a group of individuals use flight strips in air traffic management. These studies tend to use ethnographic techniques to provide a rich contextualized account of behavior (see [12] for example) or more anecdotal accounts of social behavior (see [16] in relation to social behavior using the Flickr photo-sharing service). On the other hand detailed statistical analyses of systems have been used to detect biases in their individual use (see for example [20] in relation to a mammography system). These studies are important in exploring patterns of behavior that arise from use of the system. They are time and resource intensive and require a live system. The question of the paper is how to analyze collective behavior of users in relation to a system prior to fielding the system.

While formal models have been developed and explored that are relevant to modeling the interaction between an individual user and device in context (see e.g., [8,9]) and general behavior of users have been captured through normative task models (see e.g. [10,19]) the impact modeling of collective behaviors within interactive systems have not been studied. This issue becomes particularly important in ubiquitous systems, providing smart environments in which many users are immersed and which can have an important impact on the collective behavior of those involved. This paper focuses on the role that modeling approaches can take in enabling the analysis of collective behavior during the early stages of design. The aim is that these techniques should be capable of providing a basis for usability evaluation in the face of different user strategies, when in different phases of collaboration and given different technology designs. A groupware system similar to the one used already for illustration, provides an example of the use of the particular technique.

The fundamental problem with formal modeling in relation to analyses of collective behaviors is how to deal with the state explosion that arises through attempts to model multiple instances of processes required to define the collective behavior. The paper explores a recently proposed scalable model-based technique, Fluid Flow Analysis [15]. This technique supports the analysis of many replicated entities with autonomous behavior that collaborate by means of forms of synchronization. It builds upon a process-algebraic approach and adds techniques for quantitative analyses to those for behavioral analysis. The technique has been successfully applied in areas such as large-scale Web Services [11,15], Service-Oriented Computing [22] and Grid applications [5,6], but also in Systems Biology [7].

The technique consists in deriving automatically a set of Ordinary Differential Equations (ODEs) from a specification defined using Performance Evaluation Process Algebra (PEPA) [14]. The solution of the set of ODEs, by means of standard numerical techniques, gives insight into

the dynamic change over time of aggregations of components that are in particular states. The approach abstracts away from the identity of the individual components. The derivation of sets of ODEs from PEPA specifications, the algorithms to solve ODE equations and the generation of the numerical results are supported by the PEPA workbench [21].

The problem addressed in the paper is to explore different user strategies and groupware designs for a simplified version of a groupware system called thinkteam. Two different file access policies are analyzed and compared. thinkteam is part of the Product Lifecycle Management system of think3. The Fluid Flow technique can be used in this situation because the system being analyzed involves many replicated components that can be abstracted to relatively few states. The approach can be seen as complementary with model checking in general and stochastic model checking in particular. Stochastic model checking techniques have already been applied to the same example in earlier work [1,2,3,4]. While this approach allows a richer analysis of specific properties of smaller sets of processes, Fluid Flow allows broader analysis of larger aggregations.

The paper introduces PEPA in Section 2 and briefly explains the Fluid Flow interpretation of PEPA models in Section 3. In Section 4 the thinkteam example is introduced, followed in Section 5 by a specification of the example. Section 6 describes the analysis and Section 7 outlines briefly future directions. The formal semantics of PEPA are recalled in Appendix A while the actual PEPA models used in the paper are given in detail in Appendix B. Further details on the result of the translation from PEPA models to ODEs are shown in Appendix C and Appendix D.

2 PEPA: A Process Algebra for Performance Evaluation

In PEPA, systems can be described as interactions of components that may engage in activities in much the same way as in other process algebras. Components reflect the behavior of relevant parts of the system, while activities capture the actions that the components perform. A component may itself be composed of components. The specification of a PEPA activity consists of a pair *(action type, rate)* in which *action type* denotes the type of the action, while *rate* characterizes the negative exponential distribution of the activity duration. A positive real-valued random variable X is exponentially distributed with rate r if the probability of X being at most t, i.e. $Prob\{X \leq t\}$, is $1 - e^{r \cdot t}$ if $t \geq 0$ and is 0 otherwise, where t is a real number. The expected value of X is $1/r$. Exponentially distributed random variables are more tractable because they have a memoryless property, i.e. $Prob\{X > t + t' | X > t\} = Prob\{X > t\}$ for $t, t' \geq 0$. Exponential distributions are widely used in the modeling of the dependability and performance of real systems where they form the basis for *Continuous Time Markov Chains* (CTMC), see e.g. [20].

Furthermore, proper compositions of exponential distributions can be used for the approximation of any non-negative distribution. The PEPA expressions used in this article have the following syntax [4]:

$$P ::= (\alpha, r).P \mid P + P \mid P \bowtie_L P \mid A$$

Behavioral expressions are constructed through prefixing. Component $(a, r).P$ carries out activity (a, r), with action type a and duration Δt determined by rate r. The average duration is given by $1/r$. It is defined that Δt is an exponentially distributed random variable with rate r. After performing the activity, the component behaves as P. Component $P+Q$ models a system that may behave either as P or as Q, representing a race condition between components. The cooperation operator $P \bowtie_L Q$ defines the set of action types L on which components P and Q must synchronize (or cooperate); both components proceed independently with any activity not occurring in L. The expected duration of a cooperation of activities a belonging to L is a function of the expected durations of the corresponding activities in the components. Typically, it corresponds to the longest one (see [14,15] for definition of PEPA). An important special case is the situation where one

[4] For technical reasons, actually, there are some restrictions on the nesting of parallel processes in the dialect of PEPA suitable for the translation to ODEs. For the sake of simplicity, we refrain from discussing the issue here and refer to [15] for details.

component is *passive* (a rate $\top$ indicates this) in relation to another component. Here the total rate is determined by that of the active component only. The behavior of process variable A is that of P, provided that a defining equation $A = P$ is available for A. The formal semantics of the subset of PEPA used in this paper can be found in Appendix A. We introduce two shorthand notations. If the set L is empty $P \bowtie_L Q$ is written as the parallel composition of P and Q: $P|Q$. If there are n copies of P in parallel cooperating with m parallel copies of Q this is written as: $P[n] \bowtie_L Q[m]$. In this paper we will often present PEPA specifications graphically as a kind of stochastic automata and provide the full textual PEPA specification in Appendix B.

3 ODE Semantics of PEPA

One of the advantages of a formal, high-level specification language with a fully formal semantics is that it lends itself to the application of different analysis and evaluation techniques while respecting its semantics. For example, PEPA specifications can be analysed by means of a stochastic model checker, such as PRISM [18] but it can also be used for simulation. Recently, a different form of analysis has been proposed in which PEPA specifications are translated into sets of Ordinary Differential Equations (ODEs) [15]. This technique makes it possible to analyse performance aspects of systems with a large number of repeated components.

Let us consider a small example how PEPA specifications can be transformed into sets of ODEs. Imagine a machine selling train tickets at the station and Clients buying tickets. For the sake of simplicity, let us assume that only return tickets are sold so that travellers are buying their tickets in the same station every time they need to travel and that the travellers return to the same station after travelling. Simplistically the ticket machine can be modelled as:

$$\begin{aligned} \text{TMready} &= (\text{ticket}, 1).\text{TMreset} \\ \text{TMreset} &= (\text{reset}, 1).\text{TMready} \end{aligned}$$

Initially the ticket machine is ready to sell a ticket to a traveller (TMready). Selling the ticket takes 1 minute on average. After the ticket has been sold, the machine needs 1 minute to reset itself (TMreset) and return to the initial state.

The traveller can be modelled as:

$$\begin{aligned} \text{TT} &= (\text{ticket}, 1).\text{TTtravel} \\ \text{TTtravel} &= (\text{travel}, 1/60).\text{TT} \end{aligned}$$

The traveller arrives at the ticket machine without ticket (TT) and buys one. This takes on average 1 minute; then he travels (and performs other activities not modelled here) for an average of 60 minutes (TTtravel). After that the traveller is again without a valid ticket and needs to buy a new one at the machine.

If we have 1000 travellers and 100 machines on a crowded morning their combined behaviour can be expressed in PEPA as:

$$\text{TMready}[100] \bowtie_{\text{ticket}} \text{TT}[1000]$$

So, initially we have 100 ticket machines ready to sell tickets and 1000 travellers without tickets. Their combined behaviour is synchronised on the buying/selling of a ticket, represented by activity "ticket". When translating the global behaviour into a set of differential equations, the idea is that we represent how many ticket machines (and how many travellers) are in a certain state at each time instant. Furthermore, this discrete number is approximated by a continuous value.

Each traveller without a ticket needs to find a free ticket machine, so if there are 1000 travellers without a ticket and 100 available ticket machines, each of which sells on average 1 ticket per minute, clearly 100 travellers per minute get their ticket and move on to their next state (as do the 100 ticket machines). This gives a reduction in the number of travellers that are without a ticket. On the other hand, when travellers come back from their trip and need another ticket this

causes an increase in the number of travellers without a ticket. The change of travellers buying tickets over time in general can be characterised by the following differential equation:

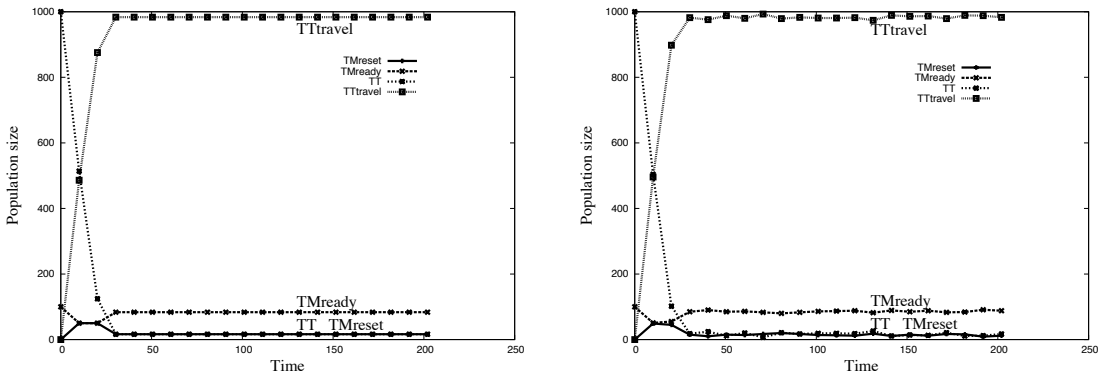
$$d\ \mathrm{TT}(t)/dt = -(1).min(\mathrm{TMready}(t), \mathrm{TT}(t)) + (1/60).\mathrm{TTtravel}(t)$$

This equation expresses the fact that the number of travellers without a ticket at time t decreases at a rate 1 per minute multiplied by the minimum of the number of available ticket machines at time t and the number of travellers without a ticket at time t. It increases at rate 1 per hour based on the simplifying assumption that the number of travellers coming back from their trip want to travel again.

The other quantities can be derived in a similar way:

$$\begin{aligned} d\ \mathrm{TTtravel}(t)/dt &= (1).min(\mathrm{TT}(t), TMready(t)) - (1/60).\mathrm{TTtravel}(t) \\ d\ \mathrm{TMready}(t)/dt &= -(1).min(\mathrm{TMready}(t), TT(t)) + (1).\mathrm{TMreset}(t) \\ d\ \mathrm{TMreset}(t)/dt &= (1).min(\mathrm{TMready}(t), TT(t)) - (1).\mathrm{TMreset}(t) \end{aligned}$$

Figure 1(a) has been obtained by solving the set of ODEs and plotting the results over a time period of around 200 minutes. It shows how, over time, starting from the initial number of travellers and ready ticket machines, the number of processes in certain states change and reach a stable situation. In fact, we can observe that after a while around 80 of the 100 ticket machines are not used. So, perhaps, under these circumstances the number of machines could be reduced. On the other hand, a peak of 1000 travellers all wishing to buy a ticket takes still about 30 minutes to process. If such peaks happen frequently, it might be a good idea to keep, or even add more ticket machines. Figure 1(b) shows a similar result obtained by a simulation for a limited number of runs of the same PEPA specification. Such a simulation is useful to get additional feedback on the correctness of the results obtained by solving the set of ODEs because, as is well-known, sometimes the solutions of sets of ODEs may be unstable. In what follows we will not show simulation results though we have checked that are consistent with the ODE approach.



(a) 1000 travellers using 100 ticket machines over 200 minutes, ODE analysis.

(b) 1000 travellers using 100 ticket machines over 200 minutes, simulation results.

Fig. 1. (a)-(b) Results of the ODE analyses and simulation.

In the ticket machine the 'bottleneck' that constrains the number of tickets sold per hour when there are many travellers is the rate at which the ticket machine effectively sells tickets. This is influenced both by the time it takes to sell a ticket and by the time taken to reset. The performance of a process can in some cases be better modelled by a passive rate representing that it is able to adapt its performance to any request rate. For example a model of an ideal ticket machine would sell as many tickets per minute as requested. Syntactically, such a passive rate is denoted by $\top$ and the specification of the first line of the ticket machine then becomes:

TMready = (ticket, $\top$).TMreset. The function *min* is then defined such that any rate λ is always smaller than $\top$: $min(\lambda, \top) = \lambda$. The use of $\top$ must be handled with care. The reason is that whenever there are no ready ticket machines available, then the rate of ticket selling should be zero and not λ. So, $\top$ can only be used if it is guaranteed that there is always a sufficient number of ready ticket machines, and this depends as is clear on the chosen rates in the model. A safer way to model the adaptive behaviour is to choose instead of $\top$ a sufficiently high rate in the ticket machine model, for example $\lambda \cdot N$ where N is the maximal number of travellers in the model. This is the approach followed in this paper, though for notational simplicity we will continue to use the symbol $\top$ in the specifications.

The generation of sets of ODEs from PEPA specifications can be performed in a fully automatic way as is explained in detail in [15] and has been implemented as part of the PEPA Workbench [21].

4 The thinkteam Groupware

The thinkteam system (http://www.think3.com/) is think3's Product Data Management (PDM) application. It is designed to deal with the document management needs of design processes in the manufacturing industry. Controlled storage and retrieval of documents in PDM applications is called vaulting, the vault being a file-system-like repository. The system is designed to be a secure and controlled storage environment, in which vaulting prevents inconsistent changes to the document base while still allowing maximal access compatible with business rules. A standard set of operations is supported (see Table 1).

Operation	Effect
get	extract a read-only copy of a file from the Vault
import	insert an external file into the Vault
checkOut	extract a copy of a file from the Vault with the intent of modifying it (exclusive, i.e. only one *checkOut* at a time is possible)
unCheckOut	cancel the effects of a preceding *checkOut*
checkIn	replace an edited file in the Vault (the file must previously have been checked out)
checkInOut	replace an edited file in the Vault, while at the same time retaining it as checked out

Table 1. thinkteam user operations

Access to files (via a *checkOut*) is based on the retrial principle: no queue or reservation system exists to handle the requests for editing rights. thinkteam typically handles some 100,000 files for 20-100 users. A user rarely checks out more than 10 files a day, but can keep a file checked out for periods from a few minutes to a few days. Log-file analysis of typical use indicated that only a small subset of the files are accessed regularly for editing. Files are typically shared by several users ranging from 2 to 5 with peaks of up to 17.

To maximize concurrency, a *checkOut* in thinkteam creates an exclusive lock for write access. An automatic solution of the write access conflict is not easy, as it is critically related to the type, nature, and scope of the changes performed on the file. Moreover, standard but harsh solutions—like maintaining a dependency relation between files and using it to simply lock all files depending on the file being checked out—are out of the question for think3 as they would cause these files to be unavailable for unacceptably long periods. In thinkteam the solution is to leave it to the users to resolve such conflicts. However, a publish/subscribe notification service would provide the means to supply the Clients with adequate information by (1) informing Clients checking out a

file of existing outstanding copies and (2) notifying the copy holders upon *checkOut* and *checkIn* of the file. [3] adds a lightweight and easy-to-use publish/subscribe notification service to thinkteam and verifies several correctness properties such as concurrency control, awareness, and denial of service. Denial-of-service is possible in this system in that one of the users can never get a turn to perform a *checkOut*. This may happen because the system is continuously kept busy by other users. Access to files is based on retrial. The usability aspects of the two file access policies need to be studied under different assumptions about how the group is using the system. In [1] two such usability aspects are studied; (1) how often, on average, users have to express their requests before they are satisfied and (2) under which system conditions (number of users, file editing time, etc.) such a reservation system would really improve usability. In that work a stochastic model-checking approach is used and a limited model with up to ten users competing for one file is analyzed. In this paper we investigate a complementary analysis based on the Fluid Flow approach were we study models with a much larger number of users and files.

5 Modelling File Access Policies

A typical thinkteam user makes requests for edit rights on files using *checkOut* operations. After editing, the file is inserted back into the vault by a *checkIn* operation. Furthermore, a typical file manager is ready to receive a request from a Client and grants this request. It then locks the file for other Clients until it is returned to the vault. Two types of file manager will first be considered. The first supports retrial while the second supports a file reservation system based on a finite queue. It is assumed that the file manager is always able to provide a timely response to the Client on the availability of the file, be it positive or negative. This is modeled using passive actions as explained in Section. 3.

5.1 The Retry Policy

Figure 2 describes models of a Client and a FileManager supporting the Retry policy. This particular model will be called the "liberal retrial model" in what follows. PEPA specifications corresponding to all the stochastic state transition diagrams presented in this paper can be found in the Appendix. The Client initially tries to *checkOut* a file. This can be successful (cos) or fail (cof). The rate a denotes the access rate and characterizes the time that passes between the last *checkIn* of a file and the next access to a file. In other words, it represents the time that a Client is busy with activities other than requesting edit rights for a file and modifying it. If the Client has successfully received edit rights to the file, she works on it for a while and checks the file in. The time involved in this activity is modelled by the rate w. If the edit rights are not granted, the Client tries again repeatedly with time intervals characterised by rate r, the retry rate. The FileManager initially is in a state in which the file is free and can accept a *checkOut* request from a Client. It then moves to a state representing that the file is now locked (FMbusy) in which further Clients' requests result in a failed checkOut (cof) until the file is checked in (ci).

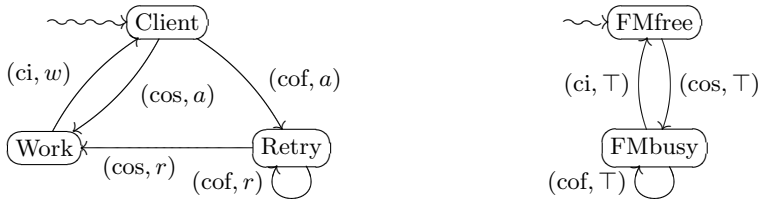


Fig. 2. From left to right: Stochastic Automata of Client and FileManager components.

All activities of the FileManager have a passive rate ($\top$), they adapt to any rate induced by the Clients. The model abstracts from the identity of the Clients by not keeping track of which

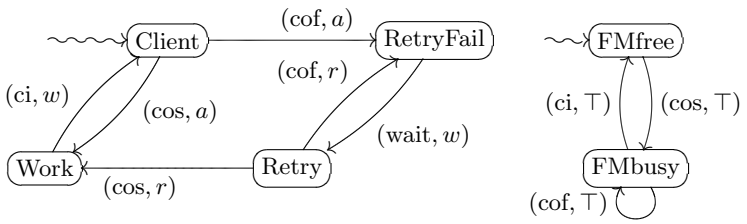


Fig. 3. From left to right: Stochastic Automata of Client and FileManager components.

Client exactly is requesting which file. The model of the Client behavior does not require that a Client's retry activity is aiming at obtaining the same file. In fact, it models Clients that try to obtain whatever file they want every time they are making a request. This can be a request for the same file or for any other file, free or occupied. In this sense the model differs from the one we presented in [1], where the fact that there was only one file implied that all three Clients are trying to get the same file. This abstraction can be achieved without loss of generality given the volumes of processes. A composed model with 90 Clients competing for 30 files can now be expressed using the PEPA cooperation operator:

$$\text{Client}[90] \bowtie_{\text{cos,ci,cof}} \text{FMfree}[30]$$

A modified specification of the Retry model (the Waiting Retry model) is given in Figure 3. Here when a *checkOut* attempt fails (cof), the Client, waits on average an amount of time equal to the length of a typical editing session ($1/w$) before trying again. This is modeled by the pair of states RetryFail and Retry and their related transitions. This model approximates a situation in which Clients keep on trying to obtain a particular file because, on average, they have to wait for such a file at least for the duration of one editing session. It could be argued that a Client may be lucky and wait less time when the Client that is currently editing has almost finished, but because the exponential distributions are memoryless the same rate w modeling the working time also models the remaining working time. As in the liberal Retry model we can express the composed model with 90 Clients and 30 FileManagers as

$$\text{Client}[90] \bowtie_{\text{cos,ci,cof}} \text{FMfree}[30].$$

5.2 The Waiting-list Policy

Figure 4 models the Waiting-list policy. The model of the FileManager supporting this policy is given in Figure 5. The Client may initially achieve: (1) a successful *checkOut* of the requested file (cos), (2) an unsuccessful *checkOut*, but placement in the waiting list (cof), or (3) a complete failure because the waiting list for the file is full (qf). In the first case, the Client edits the file and checks it in as before. In the second case, the Client waits until a notification arrives saying that it is the Client's turn to edit the file (trn). In the third case the Client has to try again to get the file or to be put on the waiting list. The model of the FileManager that supports the Waiting-list policy includes a queue. In this specific case one Client can be editing the file and at most two other Clients may be in the queue. Initially the file is free and a *checkOut* request is successful (cos). If a further request arrives the request is placed in the waiting list (cof) modeled by state FMbusyW1. If yet a further request arrives before the file is checked in it is placed in the list as well, modeled by state FMfullW2, denoting that the list is now full and two Clients are waiting for getting write access. Any further requests are answered with a queue full message (qf). When the file is checked in while the FileManager is in state FMfullW2, it moves to state FMfullW2bis from which a notification is sent to the next Client that was waiting for the file (trn). We know

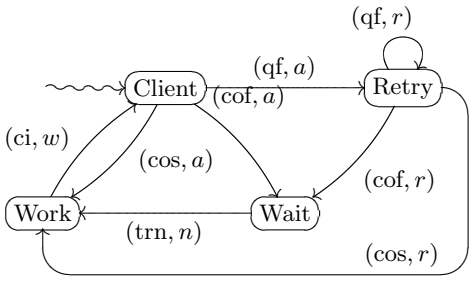


Fig. 4. Stochastic Automaton of Client component.

that such a Client exists because Clients that receive a (cof) are waiting for such a notification before they can do other things. The model

$$\text{Client}[90] \bowtie_{\text{cos,ci,cof,qf,trn}} \text{FMfree}[30]$$

now takes the new definitions for Client and FMfree. This model is not concerned with exactly

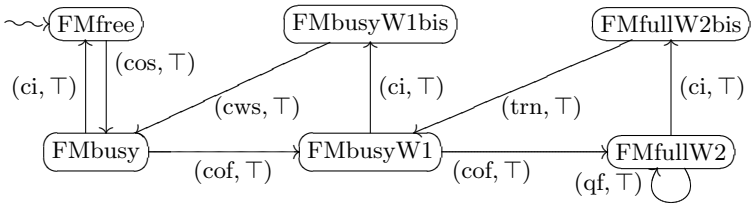


Fig. 5. Stochastic Automaton of FileManager component.

which Client gets the notification. In fact, when abstracting from identity, any Client that is waiting for a notification will do, because on average for every Client that in theory would have received the notification before its turn there is an equivalent one that receives it later than would be preferred. In daily life Clients do care about such a random assignment of turns, but note that for the purpose of the analysis, we only require that Clients wait until they receive a notification. We can correctly abstract from the identity of the Clients (and files) because we are only interested in the number of Clients that are in a certain state. This provides an indication of the performance of the overall system. To make this clearer consider the following example. If ten people stand in a queue, each with their numbered ticket, the length of the queue is not influenced by two people exchanging their tickets (or their places). If we have two queues, their length is also not influenced by the exchange of two people, one from each queue. In the case of our model, we therefore do not need to model in which queue which Client is. In this model it is necessary to synchronize also on the actions denoting queue full (qf) and next turn (trn).

6 Analysis of File Access Policies in thinkteam

The models in Section 5 can be used to explore the advantages and disadvantages of alternative strategies giving a perspective on the collective usability of these different strategies. Analysis using the PRISM stochastic model checker with a limited number of files and Clients is described in [1]. The specifications are also amenable to discrete event simulation. In this section we present the results of the Fluid Flow analysis. This analysis provides information about how many Clients are editing a file or are waiting in a queue over time. These numbers depend on the typical usage patterns of the system, which in their turn can be characterized by the values of the parameters of the model. The following assumptions are made about usage patterns, that

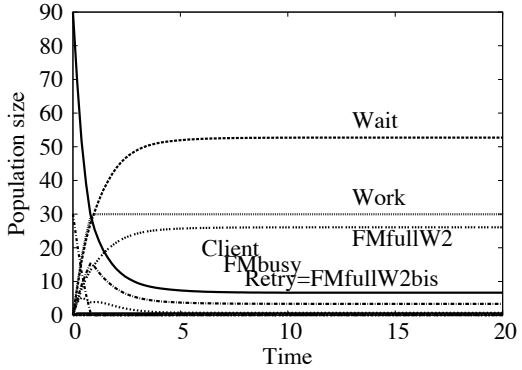
– the average time between a checkIn and the next request is 2 hours (i.e. rate $a = 0.5$);
– the system is used by 90 Clients that compete for 30 files;
– the retry rate r is $5 \cdot a$;
– editing sessions of different average duration $1/w$;
– each Client has at any moment at most one file checked out.

In addition in case of the Waiting-list model we assume that there can be at most one Client working on a file and that there can be at most two Clients in the queue before it is full.

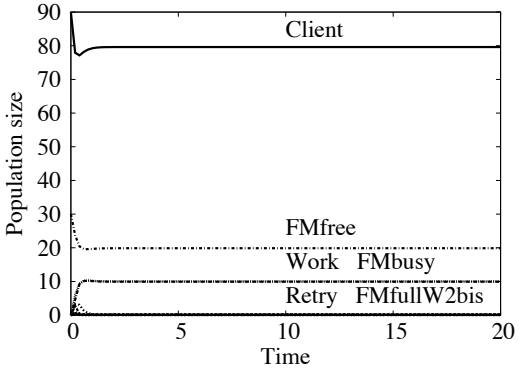
6.1 Analysis of the Waiting-list Policy

Results show average durations of editing sessions of 4 hours (Figure 6(a)) and 5 minutes (Figure 6(b)). All other assumptions are invariant. The graphs show how an initial situation of the Waiting-list model with 90 Clients and 30 free files evolves over 20 hours. Each curve shows the evolution of the number of processes in each state described in the specification of Section 5. A number of observations can be made about the number of Clients who are editing files, waiting in queues or busy trying to get a file. In all cases stability occurs within an hour or two. We can see in the longer sessions (Figure6(a)):

1. a steep decrease in the number of Clients involved in other activities, dropping from 90 initially to a stable 6.5;
2. a steep decrease in the number of free files from 30 to almost zero (arising for the fact that so many Clients are competing for files and are involved in relatively long editing sessions);
3. the number of Clients spending their time waiting in some queue is relatively high tending to approximately 52;
4. the queues themselves are quite full, i.e. approximately 26 of the 30 queues are full in the long run.



(a) Number of processes in each state in Waiting-list model with Clients editing files for 4 hours on average.



(b) Number of processes in each state in Waiting-list model with Clients editing files for 5 min. on average.

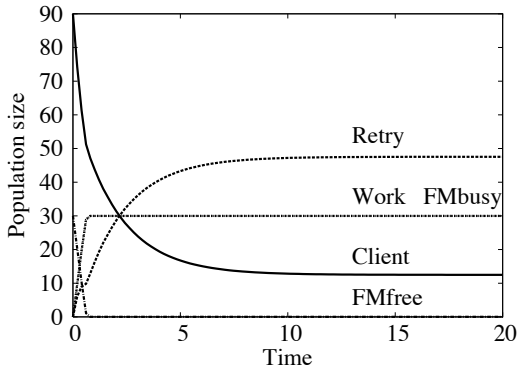
Fig. 6. (a)-(b) Results of the ODE analyses for the Waiting-list policy for long and short editing periods.

In the shorter sessions (Figure 6(b)): 10 files are actually being edited at any time and the Clients are hardly wasting any time in the queues obtaining the files they need. This situation may of course change rapidly when shorter editing times are combined with much more frequent requests for files.

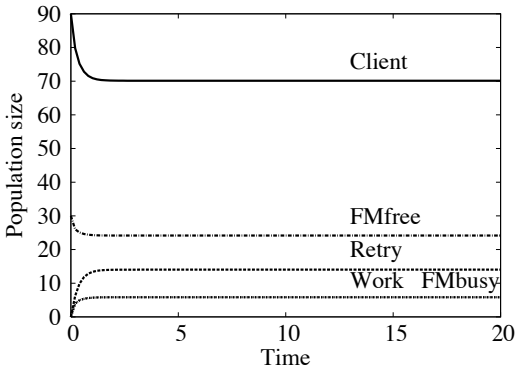
6.2 Analysis of the Retry Policy

The liberal Retry policy (Figure 7) shows at first sight a similar pattern to the Waiting-list policy. In the case of long editing sessions of about 4 hours on average we observe:

1. a rapid decrease in the number of users performing other activities than trying to get files and edit them;
2. the available files are quickly occupied;
3. approximately 45 Clients are at any time busy (re)trying to obtain files;
4. in editing sessions of 5 minutes there remains a considerable number of Clients (about 12) busy retrying to obtain files, compared with the Waiting-list policy under the same circumstances in that model almost no Clients are waiting in a queue.



(a) Number of processes in each state in liberal Retry model with Clients editing files for 4 hours on average.



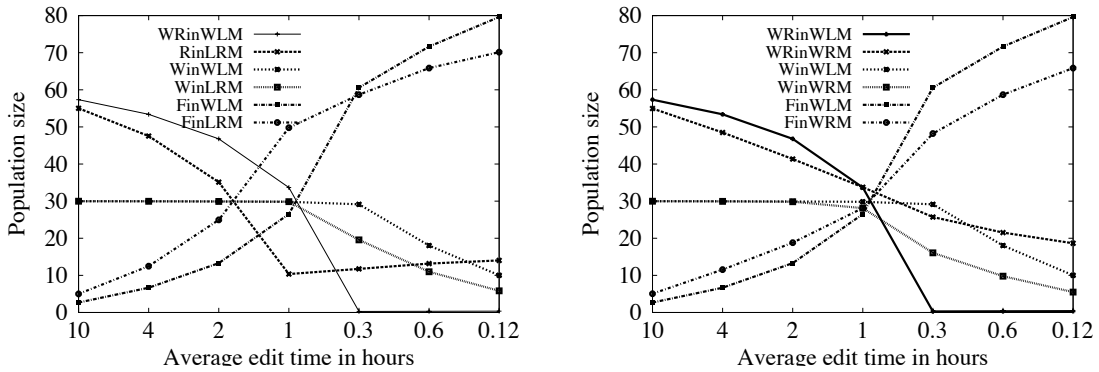
(b) Number of processes in each state in liberal Retry model with Clients editing files for 5 min. on average.

Fig. 7. (a)-(b) Results of the ODE analyses for the Retry policy for short and long editing periods.

6.3 Comparing the Usability of the Two File Access Policies

In summary the liberal Retry model and the Waiting-list model both tend toward a stable situation in relation to the number of processes that are in certain states at any moment. In Figure 8(a) we compare the usability of the liberal Retry model (LRM) and the Waiting-list model (WLM) by showing the number of free Clients (series labelled by FinLRM and FinWLM respectively), the number of working Clients (series labelled by WinLRM and WinWLM respectively) and waiting or retrying Clients (series labelled by RinLRM and WRinWLM respectively) after 20 hours of operation.

These numbers are shown under different assumptions on the average duration of the edit sessions for both the liberal Retry model and the Waiting-list model. Note that the average edit time ranges from 10 hours on average on the left, to 5 minutes on the right of the figure. The liberal Retry model appears to outperform the Waiting-list model when the duration of the edit time is more than approximately 20 minutes. This is because there are more Clients waiting for a file or involved in retry in the Waiting-list model than in the Retry model. The number of Clients working on a file is the same when the edit time is more than one hour, and the files are in that case all checked out. This result can be explained by the fact that in the liberal Retry model, when many files are checked out, the Client can in every retry attempt have a possibility to obtain a free file when available. In the Waiting-list model the Client is forced to stay in a queue and wait until an occupied file is again available. The Retry model represents a strategy in which a Client is more free to dynamically adapt their work to the situation.



(a) Liberal Retry vs. Waiting-list. (b) Waiting Retry vs. Waiting-list.

Fig. 8. (a)-(b) Comparison of Waiting-list and Retry policies.

The situation changes considerably, however, for average editing periods shorter than approximately 20 minutes. We can observe then that there are fewer Clients editing a file in the Retry model than in the Waiting-list model. In fact, in the Waiting-list model for edit sessions of less than 20 minutes very few Clients need to wait for a file, whereas a relatively large number of Clients are retrying in the Retry model. This is due to the fact that Clients do not get notified about the fact that a file became available and are wasting time in between consecutive retries. In the Waiting-list model, the waiting Clients are immediately informed about the availability of the file of interest. Figure 8(b) shows the results comparing the Waiting-Retry model (WRM) with the Waiting-list model (WLM).

The series show the number of free Clients (series labelled by FinWRM and FinWLM respectively), the number of working Clients (series labelled by WinWRM and WinWLM respectively) and waiting or retrying Clients (series labelled by WRinWRM and WRinWLM respectively) after 20 hours of operation. We can observe that for edit sessions that last more than one hour the two policies have now a more similar performance. The Waiting-Retry model still gives slightly better performance than the Waiting-list model when looking at the Clients who are free or busy retrying/waiting. This may be explained by the fact that we required that Clients in the Waiting-Retry model wait only for the duration of one session whereas when all files are occupied it is much more likely that Clients should wait for two editing sessions. This is the case for the Waiting-list model. For editing sessions of less than one hour, when not all files are continuously occupied, it is clear that the Waiting-Retry model has worse usability performance than the Waiting-list policy in the sense that Clients waste more time in retry activity than they would waiting in a queue in the Waiting-list model. Again, this is due to the fact that Clients do not know how long they should wait before attempting another *checkOut*. So, even if the file of interest is already available, Clients keep waiting before attempting a next *checkOut* request. In the Waiting list policy instead, Clients are immediately notified about the availability of the desired file, and therefore, on average, they are wasting less time.

7 Conclusions and Further Research

We have used the Performance Evaluation Process Algebra (PEPA) to develop combined user and system models to investigate usability aspects of multi-user systems with a large number of users. This has been achieved by solving sets of Ordinary Differential Equations that are automatically derived from PEPA specifications. This analysis allows for the evaluation of systems with a very high number of replicated, independent components at the cost of abstracting from the identities of these components. We have illustrated how the analysis technique can be used to inform design choices for user interaction in multiuser systems where user behavior may directly affect usability. Different usage patterns may influence performance aspects of groupware systems that are directly

relevant to its usability. We have shown how a file access policy based on a retrial principle and one based on waiting lists can be modeled and their effects on usability of the overall system can be compared for different assumptions on usage patterns. The ODE analysis results show that for usage patterns in which in the long run not all files are checked out, the Waiting-list policy makes users waste less time in waiting/retry activities than the Retry policy would under the same circumstances. Such a comparison was made by analyzing the number of Clients that are involved in certain activities at any time. These activities correspond to particular states in the respective models. In this paper we explored some initial ideas for the application of the ODE technique to the analysis of usability aspects of multi-user systems.

We think that the results are encouraging and we plan to investigate their use also in more extended case studies. In particular we are interested in using this technique to explore smart spaces, and in particular how a ubiquitous system might affect the collective behavior of users within the smart spaces. First considerations in the context of a dynamic context sensitive guidance system can be found in [13].

References

1. ter Beek, M. H., Massink, M. Latella, D.: Towards Model Checking Stochastic Aspects of the thinkteam User Interface. In S.W. Gilroy and M.D. Harrison, editors, Interactive Systems: Design, Specification, and Verification-Revised papers of DSVIS'05, volume 3941 of Lecture Notes in Computer Science, pages 39-50. Springer, 2006.
2. ter Beek, M. H., Massink, M., Latella, D., Gnesi, S.: Model Checking Groupware Protocols. In F. Darses, R. Dieng, C. Simone, and M. Zacklad, editors, Cooperative Systems Design-Scenario-Based Design of Collaborative Systems, volume 107 of Frontiers in Artificial Intelligence and Applications, pages 179-194. IOS, 2004.
3. ter Beek, M. H., Massink, M., Latella, D., Gnesi, S., Forghieri, A., Sebastianis, M.: Model Checking Publish/Subscribe Notification for thinkteam. In A. Arenas, J. Bicarregui, and A. Butterfield, editors, Proceedings of FMICS'04, volume 133 of Electronic Notes in Theoretical Computer Science, pages 275-294, 2005.
4. ter Beek, M. H., Massink, M., Latella, D., Gnesi, S., Forghieri, A., Sebastianis, M.: A Case Study on the Automated Verification of Groupware Protocols. In C. Heitmeyer and K. Pohl, editors, Proceedings of ICSE'05, pages 596-603. ACM, 2005.
5. Benoit, A., Cole, M., Gilmore, S., Hillston, J.: Scheduling Skeleton-Based Grid Applications Using PEPA and NWS Source. The Computer Journal, 48(3):369-378, 2005.
6. Benoit, A., Cole, M., Gilmore, S., Hillston, J.: Enhancing the effective utilisation of grid clusters by exploiting on-line performability analysis. In Proceedings of CCGRID'05, pages 317-324. IEEE Computer Society, 2005.
7. Calder, M., Gilmore, S., Hillston, J.: Automatically deriving ODEs from process algebra models of signalling pathways. In Proceedings of CMSB'05, pages 204-215, 2005.
8. Campos, J., Harrison, M.: Model checking interactor specifications. Automated Software Engineering, 8:275-310, 2001.
9. Campos, J., Harrison, M.: Considering context and users in interactive systems analysis. In G. van de Veer, P. Palanque, and J. Wesson, editors, Proceedings of EIS'07, Lecture Notes in Computer Science. Springer, 2007. To appear.
10. Fields, R.: Analysis of erroneous actions in the design of critical systems. PhD thesis, Department of Computer Science, University of York, 2001.
11. Gilmore, S., Tribastone, M.: Evaluating the Scalability of a Web Service-Based Distributed e-Learning and Course Management System. In M. Bravetti, M. Nunes, G. Zavattaro, editors, Proceedings of WS-FM'06, volume 4184 of Lecture Notes in Computer Science, pages 214-226. Springer, 2006.
12. Harper, R.H.R.: The organization in ethnography - a discussion of ethnographic fieldwork programs in CSCW. Computer Supported Cooperative Work 9(2): 239-264. 2000.
13. Harrison, M.D., Kray, C., Campos, J.C.: Exploring an option space to engineer a ubiquitous computing system. Electronic Notes in Theoretical Computer Science, 208C:4155, 2008.

14. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press, 1996.
15. Hillston, J.: Fluid flow approximation of PEPA models. In Proceedings of QEST'05, pages 33-43. IEEE Computer Society, 2005.
16. Kindberg, T., Spasojevic, R., Fleck, R. and Sellen, A.: The ubiquitous camera: an in-depth study of camera phone use. IEEE Pervasive Computing 4(2)42-50. 2005.
17. Kulkarni, V.: Modeling and Analysis of Stochastic Systems. Chapman & Hall, 1995.
18. Kwiatkowska, M., Norman, G., Parker, D.: Probabilistic symbolic model checking with PRISM: A hybrid approach. In J.-P. Katoen and P. Stevens, editors, Proceedings of TACAS'02, volume 2280 of Lecture Notes in Computer Science, pages 52-66. Springer, 2002.
19. Paterno, F., Mancini, C., Meniconi, S.: ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In S. Howard, J. Hammond, and G. Lindgaard, editors, Proceedings of INTERACT'97, volume 96 of IFIP Conference Proceedings, pages 362-369. Chapman & Hall, 1997.
20. Strigini, L., Povyakalo, A and Alberdi, E.: Human machine diversity in the use of computerized advisory systems: a case study. International Conference on Dependable Systems and Networks (DSN03). pages 249-258. 2003.
21. Tribastone, M.: The PEPA Plug-in Project. In M. Harchol-Balter, M. Kwiatkowska, and M. Telek, editors, Proceedings of QEST'07, pages 53-54. IEEE Computer Society, 2007.
22. Wirsing, M., Clark, A. , Gilmore, S., Holzl, M., Knapp, A., Koch, N., Schroeder, A.: Semantic-Based Development of Service-Oriented Systems. In Proceedings of FORTE'06, volume 4229 of Lecture Notes in Computer Science, pages 24-45. Springer, 2006.

APPENDICES

A PEPA Formal Semantics

In this appendix we recall the formal semantics of the subset of PEPA used in the present paper. For more information the interested reader is referred to [14].

Let $\mathcal{A}$ be a set of *action types*, ranged over by $\alpha, \alpha', \alpha_1, \ldots$ and let $P, P', P_1, \ldots$ be process terms defined according to the following grammar[5]

$$P ::= (\alpha, r).P \mid P + P \mid P \bowtie_L P \mid A$$

where r is a positive real number, denoting the rate of the duration of activity α, or the symbol $\perp$, denoting that α is *passive*, $L \subset \mathcal{A}$, and A is a *constant* which is assumed defined by a proper defining equation $A \stackrel{def}{=} P$ for some process term P. The structured operational semantics are given in Figure 9. In the rule for cooperation, with $\alpha \in L$, R stands for the following value:

$$R = \frac{r_1}{r_\alpha(P_1)} \cdot \frac{r_2}{r_\alpha(P_2)} \cdot min(r_\alpha(P_1), r_\alpha(P_2))$$

where, for process P, $r_\alpha(P)$ denotes the *apparent rate* of α in P, i.e. the total capacity of P to carry out activities of type α. For example $r_\alpha((\alpha, \lambda).P) = \lambda$, while $r_\alpha((\alpha, \lambda).P_1 + (\alpha, \mu).P_2) = \lambda + \mu$ and $r_\alpha((\alpha, \lambda).P_1 + (\alpha, \lambda).P_2 = 2\lambda$. Function r_α, for $\alpha \in \mathcal{A}$, can be defined formally, by induction on the syntax of PEPA terms [14].

[5] For technical reasons, actually, there are some restrictions on the nesting of parallel processes in the dialect of PEPA suitable for the translation to ODEs. For the sake of simplicity, we refrain from discussing the issue here and refer to [15] for details.

$$(\alpha, r).P \xrightarrow{(\alpha,r)} P \qquad \frac{P_1 \xrightarrow{(\alpha,r)} P'}{P_1 + P_2 \xrightarrow{(\alpha,r)} P'} \qquad \frac{P_2 \xrightarrow{(\alpha,r)} P'}{P_1 + P_2 \xrightarrow{(\alpha,r)} P'}$$

$$\frac{P_1 \xrightarrow{(\alpha,r)} P',\ \alpha \notin L}{P_1 \bowtie_L P_2 \xrightarrow{(\alpha,r)} P' \bowtie_L P_2} \qquad \frac{P_1 \xrightarrow{(\alpha,r_1)} P_1',\ P_2 \xrightarrow{(\alpha,r_2)} P_2',\ \alpha \in L}{P_1 \bowtie_L P_2 \xrightarrow{(\alpha,R)} P_1' \bowtie_L P_2'} \qquad \frac{P_2 \xrightarrow{(\alpha,r)} P',\ \alpha \notin L}{P_1 \bowtie_L P_2 \xrightarrow{(\alpha,r)} P_1 \bowtie_L P'}$$

$$\frac{P \xrightarrow{(\alpha,r)} P',\ (A \stackrel{def}{=} P)}{A \xrightarrow{(\alpha,r)} P'}$$

Fig. 9. Operational semantics rules

On the basis of the operational semantics rules of Figure 9 the subset of PEPA of our interest is defined as the labelled multi-transition system

$$(\mathcal{P}, Act, \{ \xrightarrow{(\alpha,r)} \mid (\alpha, r) \in Act \})$$

where $\mathcal{P}$ is the set of processes, Act is the set of activities and, for each activity (α, r) the multi-relation $\xrightarrow{(\alpha,r)}$ is given by the rules of Figure 9.

B PEPA Specifications

In this appendix we provide the full PEPA specifications of the liberal Retry model depicted in Figure 2 and of the Waiting-list model depicted in Figures 4 and 5.

First, the full PEPA specification of the liberal Retry model with parameters $a = 0.5$, $w = 0.25$ and $r = 5.0 * a$ is as follows:

$$\begin{aligned} \text{Client} &= (\text{cos}, a).\text{Work} + (\text{cof}, a).\text{Retry} \\ \text{Work} &= (\text{ci}, w).\text{Client} \\ \text{Retry} &= (\text{cof}, r).\text{Retry} + (\text{cos}, r).\text{Work} \end{aligned}$$

$$\begin{aligned} \text{FMfree} &= (\text{cos}, \top).\text{FMbusy} \\ \text{FMbusy} &= (\text{cof}, \top).\text{FMbusy} + (\text{ci}, \top).\text{FMfree} \end{aligned}$$

$$\text{Client}[90] \bowtie_{\text{cos,ci,cof}} \text{FMfree}[30]$$

Second, the full PEPA specification of the Waiting-list model with parameters $a = 0.5$, $w = 0.25$, $r = 5.0 * a$ and $n = 1000$ is as follows:

$$\begin{aligned} \text{Client} &= (\text{cos}, a).\text{Work} + (\text{cof}, a).\text{Wait} + (\text{qf}, a).\text{Retry} \\ \text{Work} &= (\text{ci}, w).\text{Client} \\ \text{Wait} &= (\text{trn}, n).\text{Work} \\ \text{Retry} &= (\text{cof}, r).\text{Wait} + (\text{cos}, r).\text{Work} + (\text{qf}, r).\text{Retry} \end{aligned}$$

$$\begin{aligned} \text{FMfree} &= (\text{cos}, \top).\text{FMbusy} \\ \text{FMbusy} &= (\text{cof}, \top).\text{FMbusyW1} + (\text{ci}, \top).\text{FMfree} \\ \text{FMbusyW1} &= (\text{cof}, \top).\text{FMfullW2} + (\text{ci}, \top).\text{FMbusyW1bis} \\ \text{FMbusyW1bis} &= (\text{trn}, \top).\text{FMbusy} \\ \text{FMfullW2} &= (\text{qf}, \top).\text{FMfullW2} + (\text{ci}, \top).\text{FMfullW2bis} \\ \text{FMfullW2bis} &= (\text{trn}, \top).\text{FMbusyW1} \end{aligned}$$

$$\text{Client}[90] \bowtie_{\text{cos,ci,cof,qf,trn}} \text{FMfree}[30]$$

C Reagent-centric model for the liberal Retry model

In this section we present the reagent-centric model of the PEPA liberal Retry model. This model is automatically generated by the PEPA workbench in a format (cdml) that is accepted by solvers such as ISBJava. Reactions are provided in the format *reactionname*,*reactants* $\rightarrow$ *products*, *reactionrate*. Note that in the version below we used an explicit high rate *top* instead of a passive rate $\top$.

```
//Rates
a = 0.5;
r = 5.0*a;
top = 90.0*r;
w = 0.1;

//Population sizes
Client = 90; // Client
FMbusy = 0; // FMbusy
FMfree = 30; // FMfree
Retry = 0; // Retry
Work = 0; // Work

//Reactions
ci, Work + FMbusy -> Client + FMfree, [min(Work*w,FMbusy*top)];
cof1, Client + FMbusy -> Retry + FMbusy, [(a/(r+a))*min((Client*(r+a)),FMbusy*top)];
cof2, Retry + FMbusy -> Retry + FMbusy, [(r/(r+a))*min((Retry*(r+a)),FMbusy*top)];
cos1, Retry + FMfree -> Work + FMbusy, [(r/(r+a))*min((Retry*(r+a)),FMfree*top)];
cos2, Client + FMfree -> Work + FMbusy, [(a/(r+a))*min((Client*(r+a)),FMfree*top)];
```

D Ordinary Differential Equations for the liberal Retry model

Below we present the ordinary differential equations corresponding to the PEPA liberal Retry model. Note that in the equation for Retry and FMbusy the terms related to the self-loops have been omitted since their sum results in zero. The equations have been obtained interpreting the reagent-centric model.

$$
\begin{aligned}
d\ \mathrm{Work}(t)/dt = &-\min(\mathrm{Work} * w, \mathrm{FMbusy} * top) \\
&+\min(\mathrm{Retry} * r, \mathrm{FMfree} * top * \tfrac{r}{(r+a)}) \\
&+\min(\mathrm{Client} * a, \mathrm{FMfree} * top * \tfrac{a}{(r+a)})
\end{aligned}
$$

$$
\begin{aligned}
d\ \mathrm{Client}(t)/dt = &-\min(\mathrm{Client} * a, \mathrm{FMfree} * top * \tfrac{a}{(r+a)}) \\
&-\min(\mathrm{Client} * a, \mathrm{FMbusy} * top * \tfrac{a}{(r+a)}) \\
&+\min(\mathrm{Work} * w, \mathrm{FMbusy} * top)
\end{aligned}
$$

$$
\begin{aligned}
d\ \mathrm{Retry}(t)/dt = &-\min(\mathrm{Retry} * r, \mathrm{FMfree} * top * \tfrac{r}{(r+a)}) \\
&+\min(\mathrm{Client} * a, \mathrm{FMbusy} * top * \tfrac{a}{(r+a)})
\end{aligned}
$$

$$
\begin{aligned}
d\ \mathrm{FMfree}(t)/dt = &-\min(\mathrm{Retry} * r, \mathrm{FMfree} * top * \tfrac{r}{(r+a)}) \\
&-\min(\mathrm{Client} * a, \mathrm{FMfree} * top * \tfrac{a}{(r+a)}) \\
&+\min(\mathrm{Work} * w, \mathrm{FMbusy} * top)
\end{aligned}
$$

$$
\begin{aligned}
d\ \mathrm{FMbusy}(t)/dt = &-\min(\mathrm{Work} * w, \mathrm{FMbusy} * top) \\
&+\min(\mathrm{Retry} * r, \mathrm{FMfree} * top * \tfrac{r}{(r+a)}) \\
&+\min(\mathrm{Client} * a, \mathrm{FMfree} * top * \tfrac{a}{(r+a)})
\end{aligned}
$$