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

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

[1] CNR - ISTI, Pisa, Italy
{massink,latella,terbeek}@isti.cnr.it
[2] School of Computing Science, Newcastle University, UK
Michael.Harrison@ncl.ac.uk
[3] Universita' di Firenze, Dip. di Sistemi e Informatica, Italy
loreti@dsi.unifi.it

Abstract. The analysis of usability aspects of multi-user systems, such as co-operative 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 a groupware application.

Keywords: Formal Methods, Model-based usability analysis, Performance Evaluation Process Algebra, Ordinary Differential Equations, Groupware Systems.

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 co-operative 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 individual's 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 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 [21] 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. [10,20]) the impact of modeling 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 [23] 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 [22].

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 the thinkteam example is introduced, followed in Section 4 by a specification of the example. Section 5 describes the analysis and section 6 outlines briefly future directions.

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. $Prob(X \le t)$, is $1-e^{-r \times t}$ if $t \ge 0$ 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 $t, t' \ge 0$. 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. [21]. 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[1]:

$$P ::= (a, r).P \mid P + P \mid P \|_{\{L\}} P \mid A$$

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 co-operation operator *P* $\|_{\{L\}}$ *Q* defines the set of action types *L* on which components *P* and *Q* must synchronize (or *co-operate*); both components proceed independently with any activity not occurring in *L*. The expected duration of a co-operation 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 [15] for definition of PEPA). An important special case is the situation where one component is passive (a rate T 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*. We introduce two shorthand notations. If the set *L* is empty *P* $\|_{\{L\}}$ *Q* is written as the parallel composition of *P* and *Q*: *P|Q*. If there are *n* copies of *P* in parallel co-operating with *m* parallel copies of *Q* this is written as: *P[n]* $\|_{\{L\}}$ *Q[m]*. We will present PEPA specifications as stochastic state transition diagrams throughout the paper. Full PEPA specifications for the same system can be found in the full version of the paper [19].

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 preserving its semantics. For example, PEPA specifications can be analyzed by means of a stochastic model checker, such as PRISM [18], and it can also be used for simulation. As already mentioned PEPA specifications can be translated into sets of Ordinary Differential Equations (ODEs) [15]. A very brief summary of the approach follows; more details can be found in [19,15]. Suppose a PEPA model $S_1[n_1] \|_{L1} S_2[n_2] \|_{L2} ... \|_{Lk-1} S_k[n_k]$ is given, which is composed of $n_1+n_2+ ...+n_k$ sequential components. Each component S_j is defined by means of a

---

[1] For technical reasons 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 [9] for details. The symbol for co-operation in PEPA is different from the one used in the present paper.

PEPA defining equation S_j = ... S_{jr} ... S_{jv} ... S_{jw}, where S_j, S_{jr}, S_{jv} ... S_{jw} are the relevant states of S_j; all such states are themselves defined by means of equations. The solution of the set of ODEs associated with the PEPA model is a set of *continuous functions*. In particular, there is one function $\underline{S}(t)$ for each state S occurring in the original speci-fication and, for each time instant t, $\underline{S}(t)$ yields a continuous approximation of the total number of components which are in state S at time t, given the initial conditions $\underline{S_1}(0)= n_1$, $\underline{S_2}(0)= n_2$, ... ,$\underline{S_k}(0)=n_k$. Notice that the fact that the values n_1, n_2, ..., n_k of the *number of components* in the system (at the initial configuration) can be very high, e.g. in the order of millions, makes the approach intrinsically scalable. In the experi-ments described in Section 5, results are compared with those obtained via discrete event simulation and are found to be comparable.

3   The Thinkteam Groupware

Thinkteam (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 docu-ments in PDM applications is called vaulting, the vault being a file-system-like re-pository. The system is designed to be a secure and controlled storage environment, in which vaulting prevents inconsistent changes to the document base while still allow-ing maximal access compatible with business rules. A standard set of operations is supported (see Table 1).

Access to files (via a *checkOut*) is based on *the retrial principle*: no queue or res-ervation 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*.

Table 1. Thinkteam user operations

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 the 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

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.

4   Modeling 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 rates as explained in Section 2.

4.1   The Retry Policy

Figure 1 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 [19]. 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 *check-Out* 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).
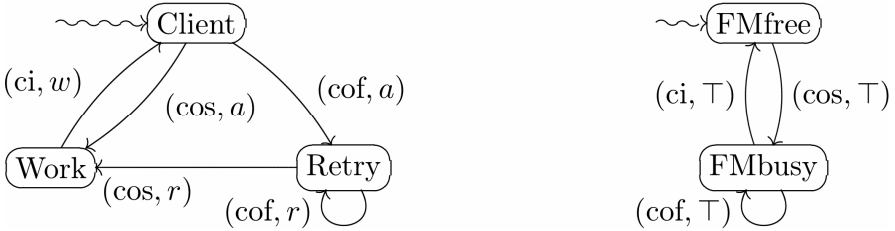

Fig. 1. From left to right: Stochastic Automata of Client and FileManager components

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 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 co-operation operator: Client[90] $\|_{\{cos,ci,cof\}}$ FMfree[30].

A modified specification of the Retry model (the waiting retry model) is given in Figure 2. 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 together and 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'
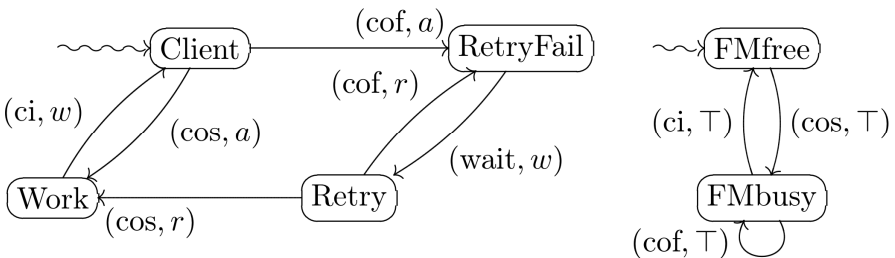

Fig. 2. From left to right: Stochastic Automata of Client and FileManager components

and wait less time when the Client that is currently editing has almost finished. 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 Client[90] $\|_{\{cos,ci,cof\}}$ FMfree[30].

4.2   The Waiting-List Policy

Figure 3 models the Waiting-list policy. The model of the FileManager supporting this policy is given in Figure 4. 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 write access. Any further requests are answered with a 'queue full' message (qf).
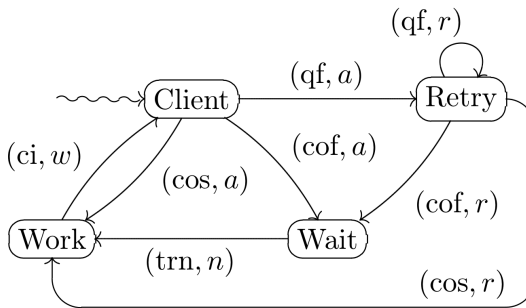

Fig. 3. Client component

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 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 Client[90] $\|_{\{cos,ci,cof,qf,trn\}}$ FMfree[30] now takes the new definitions for Client and FMfree.

This model is not concerned with exactly 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
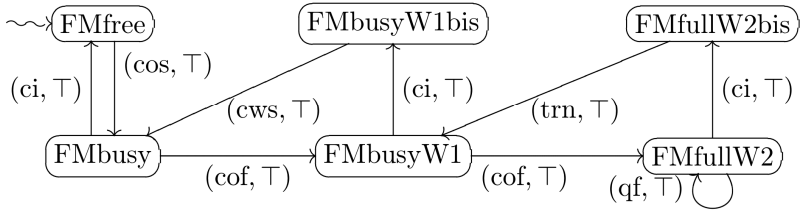
Fig. 4. The FileManager

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 therefore we do not need to model in which queue the Client is. In this model it is necessary to synchronize also on the actions denoting queue full (qf) and next turn (trn).

5   Analysis of File Access Policies in Thinkteam

The models in Section 4 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 $5 \times a$
- Editing sessions of different average duration $1/w$
- Each Client has at any moment at most one file checked out.

In addition in the 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.

5.1   Analysis of the Waiting-List Policy

Results show average durations of editing sessions of *4* hours (Figure 5(a)) and *5* minutes (Figure 5(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 4. 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 (Figure 5(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.
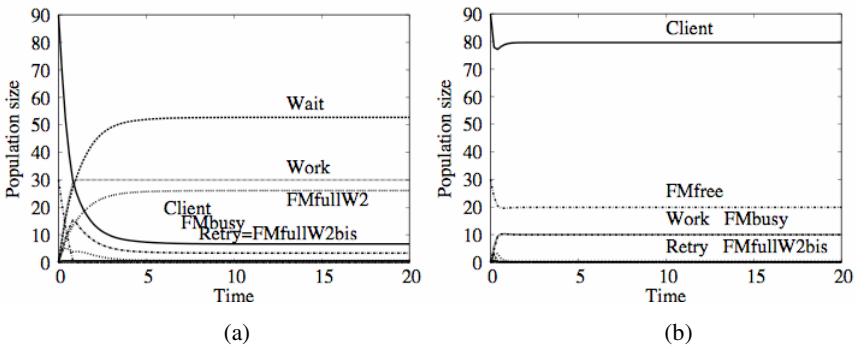

Fig. 5. ODE analysis of Waiting-list model, number of processes in each state with Clients editing files for: (a) *4* hours on average; (b) *5* minutes on average

In the shorter sessions (Figure 5(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.

5.2   Analysis of the Retry Policy

The liberal Retry policy (Figure 6) 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 try-
    ing 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 Cli-
    ents (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.

5.3 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 7(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 (se-
ries labelled by WinLRM and WinWLM respectively) and waiting or retrying Clients
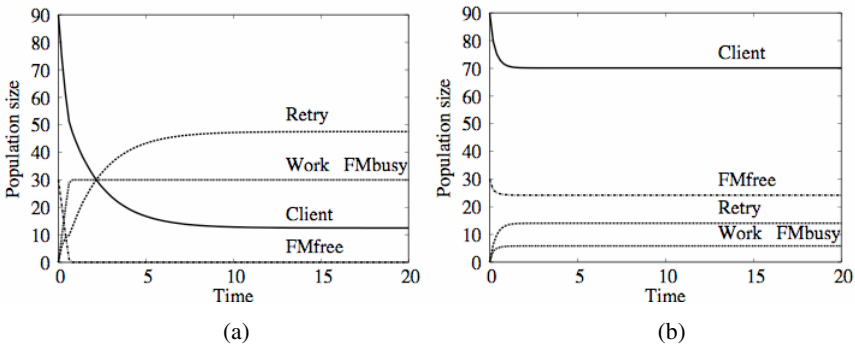(series labelled by RinLRM and WRinWLM respectively) after *20* hours of operation.

Fig. 6. ODE analyses for the liberal Retry policy for number of processes in each state editing
files for (a) *4* hours on average (b) *5* minutes on average

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 possi-
bility 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.

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 7(b) shows the results comparing the Waiting-Retry model (WRM) with the Waiting-list model (WLM).
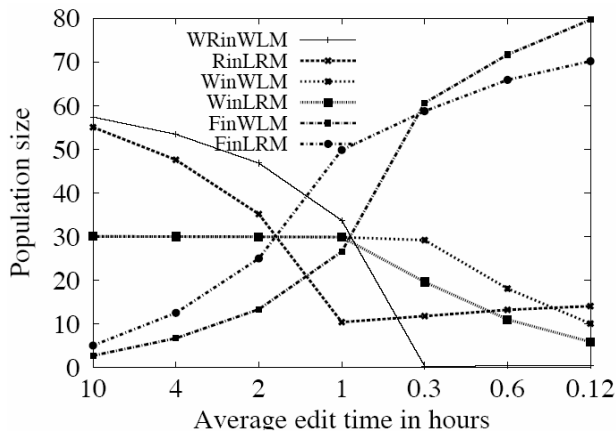

Fig. 7(a). Comparison of policies: liberal Retry vs. Waiting-list

The series shows 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
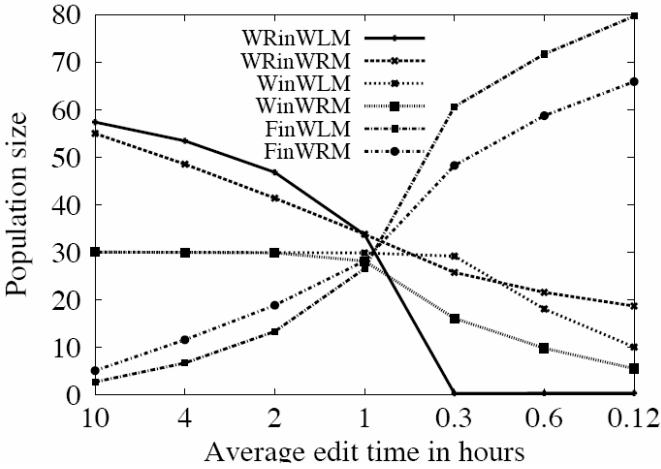
Fig. 7(b). Comparison of policies: Waiting Retry vs. Waiting-list

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.

6   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 multi-user 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 where 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].

Acknowledgments

References

1. ter Beek, M., Massink, M., Latella, D.: Towards Model Checking Stochastic Aspects of the thinkteam User Interface. In: Gilroy, S.W., Harrison, M.D. (eds.) DSV-IS 2005. LNCS, vol. 3941, pp. 39–50. Springer, Heidelberg (2006)
2. ter Beek, M., Massink, M., Latella, D., Gnesi, S.: Model Checking GroupwareProtocols. In: Darses, F., Dieng, R., Simone, C., Zacklad, M. (eds.) Co-operative Systems Design-Scenario-Based Design of Collaborative Systems. Frontiers in Artificial Intelligence and Applications, vol. 107, pp. 79–194. IOS (2004)
3. ter Beek, M., Massink, M., Latella, D., Gnesi, S., Forghieri, A., Sebastianis, M.: Model Checking Publish/Subscribe Notification for thinkteam. In: Arenas, A., Bicarregui, J., Butterfield, A. (eds.) Proceedings of FMICS 2004. Electronic Notes in Theoretical Computer Science, vol. 133, pp. 275–294 (2005)
4. ter Beek, M., Massink, M., Latella, D., Gnesi, S., Forghieri, A., Sebastianis, M.: A Case Study on the Automated Verification of Groupware Protocols. In: Heitmeyer, C., Pohl, K. (eds.) ICSE 2005, pp. 596–603. ACM, New York (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 2005, pp. 317–324. IEEE Computer Society Press, Los Alamitos (2005)
7. Calder, M., Gilmore, S., Hillston, J.: Automatically deriving ODEs from process algebra models of signalling pathways. In: Proceedings of CMSB 2005, pp. 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: van de Veer, G., Palanque, P., Wesson, J. (eds.) Proceedings of EIS 2007, Lecture Notes in Computer Science. LNCS. Springer, Heidelberg (to appear, 2007)
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: Bravetti, M., Núñez, M., Zavattaro, G. (eds.) WS-FM 2006. LNCS, vol. 4184, pp. 214–226. Springer, Heidelberg (2006)
12. Harper, R.H.R.: The organization in ethnography – a discussion of ethnographic fieldwork programs in CSCW. Computer Supported Co-operative 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, vol. 208C, pp. 41–55 (2008)
14. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press, Cambridge (1996)
15. Hillston, J.: Fluid flow approximation of PEPA models. In: Proceedings of QEST 2005, pp. 33–43. IEEE Computer Society Press, Los Alamitos (2005)
16. Kindberg, T., Spasojevic, R., Fleck, R., 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, Boca Raton (1995)
18. Kwiatkowska, M., Norman, G., Parker, D.: Probabilistic symbolic model checking with PRISM: A hybrid approach. In: Katoen, J.-P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, pp. 52–66. Springer, Heidelberg (2002)
19. Massink, M., Latella, D., ter Beek, M., Harrison, M., Loreti, M.: A Fluid Flow Approach to Usability Analysis of Multi-user Systems-Full version. ISTI Technical Report (to appear), http://www.isti.cnr.it/People/M.Massink/t3ode.pdf
20. Paterno, F., Mancini, C., Meniconi, S.: ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In: Howard, S., Hammond, J., Lindgaard, G. (eds.) Proceedings of INTERACT 1997, pp. 362–369. Chapman & Hall, Boca Raton (1997)
21. Strigini, L., Povyakalo, A., Alberdi, E.: Human machine diversity in the use of computerized advisory systems: a case study. In: International Conference on Dependable Systems and Networks (DSN 2003), pp. 249–258 (2003)
22. Tribastone, M.: The PEPA Plug-in Project. In: Harchol-Balter, M., Kwiatkowska, M., Telek, M. (eds.) Proceedings of QEST 2007, pp. 53–54. IEEE Computer Society Press, Los Alamitos (2007)
23. Wirsing, M., Clark, A., Gilmore, S., Holzl, M., Knapp, A., Koch, N., Schroeder, A.: Semantic-Based Development of Service-Oriented Systems. In: Najm, E., Pradat-Peyre, J.-F., Donzeau-Gouge, V.V. (eds.) FORTE 2006. LNCS, vol. 4229, pp. 24–45. Springer, Heidelberg (2006)