thinkteam® with Replicated Data Repositories: Stochastic Model Checking in Industry*

Maurice H. ter Beek Stefania Gnesi Diego Latella Mieke Massink
Gianluca Trentanni
ISTI–CNR, Via G. Moruzzi 1, 56124 Pisa, Italy
{terbeek,gnesi,massink,latella,trentanni}@isti.cnr.it

Maurizio Sebastianis
think3 Inc., Via Ronzani 7/29, 40033 Bologna, Italy
sebastianis@think3.com

Abstract

Product Data Management (PDM) systems support the document management of design processes like those used in the manufacturing industry. They allow enterprises to capture, organise, automate and share engineering information efficiently. Essential aspects of such systems are handling queries on product information and down- and uploading assemblies of related files for modification by designers. The efficiency of such a system as perceived by its users depends on its correct functioning, but also for a significant part on its performance aspects. We apply stochastic model checking in an industrial setting to evaluate a service-oriented extension of the PDM system thinkteam *with multiple replicated vaults. We investigate the effect of different quality of service assumptions on the system's usability from a user's perspective, and briefly report on our industrial experience with stochastic model checking.*

1 Introduction

We report on formal analyses of several qualitative and quantitative properties of a small, yet relevant industrial case study concerning think3's application thinkteam. Product Lifecycle Management (PLM) is the activity of managing a company's product across its lifecycle—from conception, through design and manufacturing, to service and disposal—in the most effective way [18]. think3's thinkPLM is a suite of integrated PLM applications built on thinkteam, think3's Product Data Management (PDM) application catering the document management needs of design processes in the manufacturing industry. It allows enterprises to capture, organise, automate and share engineering information in an efficient way. The current thinkteam setting consists of a number of users that interact with one centralised Relational Database Management System (RDBMS). This RDBMS controls the storage and retrieval of data (like CAD files) in a file-system-like repository, called the vault. Resource management is based on a 'retrial' principle: There is no queue (or reservation system) handling the user's requests for editing rights on a file.

In [4, 5], we used *qualitative* model checking to verify thinkteam extended with publish/subscribe notification. We analysed a number of qualitative correctness properties addressing concurrency, correctness, usability and awareness. Some usability issues, influenced more by a system's performance than by its functional behaviour, however cannot be analysed by qualitative model checking alone. One such issue was raised in [4, 5]: We showed that a user can be excluded from obtaining a file simply because competing users are more 'lucky' in their attempts to obtain the same file. Such behaviour is explained by the fact that users are only provided with a retry-based file-access mechanism.

Analyses with qualitative model checking can show a problem exists, but not quantify its effect on usability. In our case, the number of retries (i.e. of repeated requests) a user has to perform before obtaining a file is an important usability measure. If the number is high, a waiting-list policy should be considered rather than the simpler retry policy. The trade-off between these design options was analysed in [6] with stochastic model checking, an extension of qualitative model checking allowing the analysis of qualitative and performance- and dependability-related—i.e. quantitative—system properties [9, 11, 14, 19]. The challenge is to introduce this technique in industry, where model checking is not part of software engineering practice.

*Work partially funded by EU project Sensoria (IST-2005-016004), Italian MIUR/FIRB project tocai.it and Italian CNR/RSTL project XXL.

In [7] we used qualitative and quantitative model checking to evaluate thinkteam, and three proposed extensions, to assist the design phase of this industrial groupware system. In the sequel we revisit one analysis of [7], namely the one using *stochastic* model checking to verify an extension with multiple replicated vaults. We use the same stochastic model, but vary the parameters. In particular, we use realistic parameters obtained by analysing one think3 client's thinkteam log-file by using the statistical software package SPSS [15]. This provides analysis results that show the model's use to analyse a specific thinkteam usage context, improving slightly the part of [7] in which more generic assumptions on edit and download times were made.

The thinkteam model is assumed to contain persistent data on the status of the replicated vault locations. When a user queries thinkteam, the latter typically responds by assigning her/him the 'best possible' vault location (usually the one with a good connection in terms of bandwidth). If, however, the preferred location is down or has a too high workload, then a second-best location is assigned. The choice of which vault location to assign of course affects system performance and thus also the adequacy of the support the system can give its users. Among the properties we verify are the guarantee that data is modified by at most one user at a time, the expected waiting time for users querying a vault, the effect of workload conditions on thinkteam's usability and on the overhead in down- and uploading data.

This paper starts by describing thinkteam, the log-file analysis and the proposed extension, followed by an introduction to stochastic model checking. Then our stochastic thinkteam model is described and relevant correctness and performance properties are formalised and verified and the results are interpreted. A conclusion discusses future work.

2 thinkteam

This section contains an overview of think3's PDM application thinkteam. For details, see `www.think3.com`.

Design activities produce and consume information—both documental (CAD drawings, models, and manuals) and non-documental (bill of materials, reports, and workflow trails). The composition of this information eventually activates the process that produces a physical object. Information mismanagement can—and often does—have direct impact on the cost structure of the manufacturing phase. An important part of the design office's work is to maintain and update projects that have been previously released: A historical view of past information is an absolute must for this. This is where PDM applications come into play.

2.1 Technical Characteristics

thinkteam is a three-tier data management system running on Wintel platforms. A typical installation scenario is a network of desktop clients interacting with one centralised RDBMS server and one or more file servers. In this setting, components resident on each client node supply a graphical interface, metadata management and integration services. Persistence services are achieved by building on the characteristics of the RDBMS and file servers. We now give a general description of the operations of the (logical) thinkteam subsystems relevant to the purpose of this paper.

Metadata management. thinkteam allows its users to manage representations of concrete entities (e.g. documents and components). These representations (often called business items or business objects) are described using an object model or meta-object model which can be customised by the end users, e.g. by changing the attributes pertaining to various types of objects or by adding object types. Metadata management refers to operations on object instances and to the rules these operations obey to, as they are implemented in thinkteam. Typical operations are creation, attribute editing (e.g. adding/changing description, price, etc.), revisioning, changing state, connecting with other objects and deletion. thinkteam uses a RDBMS to persist and retrieve both its object model and the objects that are created during operation. RDBMS interactions are fairly low level in nature and are completely transparent to end users.

Vaulting. Controlled storage and retrieval of document data in PDM applications is traditionally called vaulting, the vault being a file-system-like repository. The two main functions of vaulting are (1) to provide a single, secure and controlled storage environment in which the documents controlled by the PDM application are managed and (2) to prevent inconsistent updates or changes to the document base, while still allowing the maximal access compatible with the business rules. While the first function is subject to the implementation of the vaulting system's lower layers, the second is implemented in thinkteam's underlying protocol by a set of operations available to the users, namely:

get: extract a read-only copy of a document from the vault;
import: insert an external document into the vault;
checkOut: extract a copy of a document from the vault to modify it (exclusive);
unCheckOut: cancel the effects of a previous checkout;
checkIn: replace an edited (previously checked out) document in the vault;
checkInOut: replace an edited document in the vault (keeping it checked out).

Note that document access (through a *checkOut*) is based on a 'retrial' principle: There is no queue or reservation system handling the requests for editing rights on a document.

2.2 thinkteam at Work

thinkteam supports CAD designers in various design

phases of the overall industrialisation part of a given project. Vaulting capabilities are most frequently used by CAD designers during the modelling phases, briefly described next.

Geometry information retrieval. The most usual design work in the manufacturing industry (thinkteam's prime target) involves the production of components that are part of more complex goods. The CAD models describing these products are called *assemblies* and are structured as composite documents referring to several (sometimes thousands) individual model files. Most of the geometry data a designer deals with thus consists of reference material, i.e. parts surrounding the component (s)he is actually creating or modifying. The designer needs to interact with this material in order to position, adapt and mate to the assembly the part (s)he is working on. Most reference parts are production items subjected to PDM management, whose physical counterparts (model files) reside in the vault. The logical operation by which the designer gains access to them is the *get* operation, which is performed automatically. This is the type of activity that happens most often and which is involved in all other activities listed below, as well as in many others not explicitly mentioned (e.g. visualisation, printing).

Geometry modification. Modifying an existing part is the second-most-used operation a designer performs. As it is an already existing and managed part (i.e. present in the vault) the designer must express the intent to modify it with an explicit exclusive *checkOut* operation that prevents modification attempts by other users. When ready to publish her/his work, the designer releases it to the system via the explicit *checkIn* operation, which frees the model for modification. Were the designer to change her/his mind, then (s)he may use the *unCheckOut* operation, which frees the model but discards any changes that occurred since the *checkOut*. Finally, (s)he may use the *checkInOut* operation to release to the system an intermediate version of the model, without releasing it for further modification.

Geometry creation. Lastly, a designer may create a completely new component and insert it into the system. As the part will initially be created outside the system vault, an *import* operation is required to register it in thinkteam.

3 Analysis of thinkteam Log-File

thinkteam handles a few hundred thousand files for some 20-100 users. To obtain realistic data on the particular use that clients make of thinkteam, think3 provided us with a cleaned-up log-file to analyse, comprising all activity (in terms of the operations listed in Sect. 2.1) of one of the manufacturing industries using thinkteam from 2002 to 2006. This log-file contains, for each operation, its time stamp (in the format day-month-year and hour-minute-second), the name of the user that performed it and the file the operation refers to. In this way, each line in the log-file represents an atomic access to the Vault. The format of the log-file is easy to handle, but it contains a really huge amount of data ($792,618$ Vault accesses by 104 users w.r.t. $183,492$ files). Moreover, think3 has improved its logging mechanism during the years. For these reasons, we restricted our analysis to the year 2006. The aim of our analysis was to obtain some insight on the timing issues concerning the duration of editing sessions and the occupancy of files.

The data of 2006 concerns 83 users collaborating on a total of $181,535$ files, $23,134$ of which were checked out at least once during the year. The remaining files were used exclusively as reference material, e.g. downloaded in read-only fashion by means of *get* operations. A total of 65 users was found to be involved in editing sessions. We present the analysis of a subset of the data that has direct relevance for the model we will use in the sequel. These concern the duration of editing sessions (i.e. the time that passes between a *checkOut* and a *checkIn* of a file by the same user) and the duration of periods in which files were not locked (i.e. the time that passes between a *checkIn* and *checkOut* of the same file by possibly different users). Instead, the number of times a user unsuccessfully tries to *checkOut* a locked file (i.e. checked out by another user) is not explicitly logged, so little can be said about that. It is however possible to obtain an indirect approximation of the number of users that compete for access to the same file by analysing the number of users that modify the same file during the investigated period. To give an idea, Fig. 1 shows one week of editing activity on a particular file. We see that four users are responsible for 20 editing sessions that week to the same file.

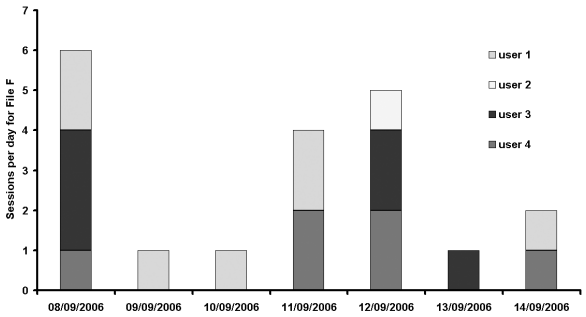

Figure 1. A week of *checkOut* sessions on a file.

Figs. 2 and 3 present data obtained after pruning the log-file to filter out the logging of irrelevant operations, like numerous *get* operations (recall this is the most-frequently-used operation), operations originating from system administrator interventions and other anomalous log operations. The graphs were drawn by SPSS v15.0 [15] after computing the *mean trimmed* 40% (i.e. discard the lower and higher 20% scores and take the mean of the remaining scores).

Fig. 2 shows a histogram of the distribution of the duration of editing sessions. On the x-axis, the time is presented in seconds. The histogram thus shows all ses-

sions except for 20% of the shortest and 20% of the longest sessions, meaning that sessions of less than 111 seconds (i.e. ca. 2 minutes) and more than 22,256 seconds (i.e. ca. 370 minutes) have been removed. This was done because the log-file contained many very short sessions not corresponding to real user editing sessions, but rather to automatic system operations that were also logged and that feature in the log-file as very short *checkOut-checkIn* sessions. We see that the mean duration trimmed 40% of an edit session is 2,657 seconds, so ca. 44 minutes. It is easy to see that most sessions tend to be rather short.

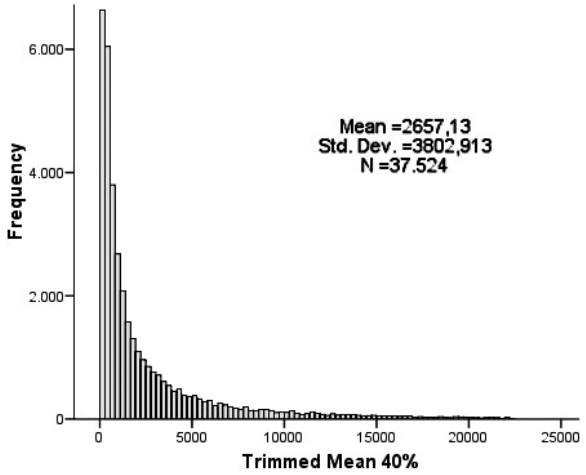

Figure 2. Duration of edit sessions.

In Fig. 3 a histogram is shown of the duration of intervals during which files, that were involved in at least two editing sessions, were *not* locked (i.e. taken in *checkOut*). These data give an impression of the time that passes between Vault accesses to the same file by possibly different users. Each bar in this histogram corresponds to a duration of about 17 hrs. Many of the intervals fall into the first few periods, indicating that there are many cases in which files were used rather intensively.

Finally, Table 1 shows the number of files that in 2006 were edited by more than one user. A further analysis of these files shows that it is quite common that multiple users are editing the same file on the same day (cf. Fig. 1) and that there are even days in which upto 8 users are accessing the same file. We are aware of the fact that the log-file analysis is only covering one year of data collected at one particular client of think3, and may therefore not be completely representative of a typical thinkteam use. However, the logged data are real observations and as such do provide information on an example use of thinkteam, which can help to put our modelling results in a proper perspective.

Nr. files	5,077	1,407	301	79	24	22	8	8	6	10	3	1	1	1
Nr. users	2	3	4	5	6	7	8	9	10	11	12	13	14	17

Table 1. Nr. of files edited by at least 2 users.

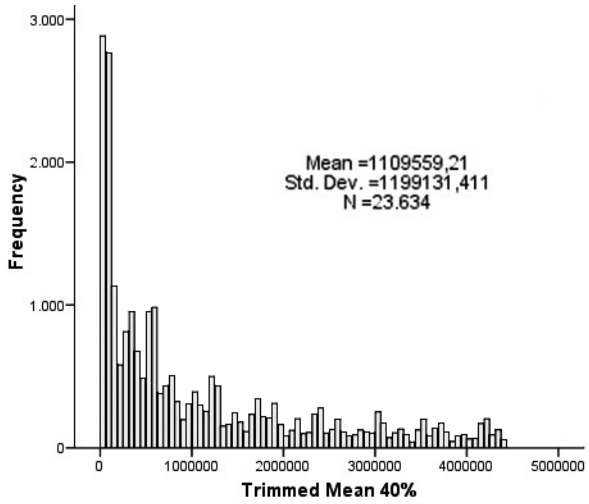

Figure 3. Duration of file inter-access time.

4 thinkteam with Replicated Vaults

think3 is considering to add a service-oriented functionality to thinkteam, namely multiple replicated vaults. These vaults reside in a number of locations, assumed to be geographically distributed (cf. Fig. 4). thinkteam is assumed to have persistent data on the status of the replicated vaults and of all files, i.e. whether a file is currently checked out by a designer or available for modification.

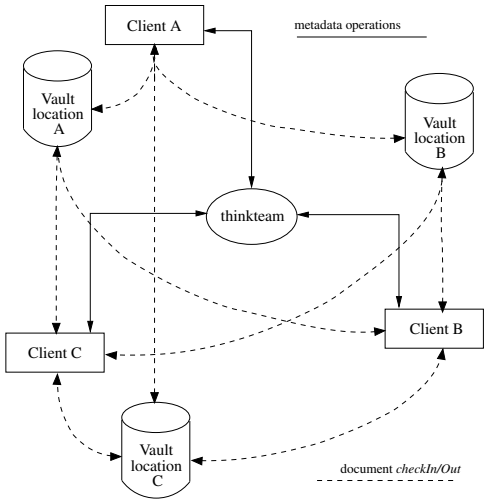

Figure 4. thinkteam with replicated vaults.

When a designer queries thinkteam in this new setting, e.g. for a copy of a file, thinkteam typically responds by assigning the 'best possible' vault location. Ideally, this is the designer's preferred vault location (with a good connection in terms of bandwidth), while a second-best location is assigned if the preferred location is down or has a too high workload. If, on the other hand, thinkteam notices that the

most recent *checkIn* of the requested file was performed by the same designer, then (s)he is told to use the local version of the file on her/his desktop (thus saving a *checkOut*).

When a designer has obtained a vault address, (s)he may *checkOut* the file, edit it and eventually *checkIn* the file, again with a strong preference for her/his preferred vault location. After each *checkIn*, the respective location informs thinkteam that the file has been uploaded. Afterwards, thinkteam updates the file's status, i.e. removes the lock, and makes it available for other designers. This communication also transfers the status information of the vault locations to thinkteam. Neither the communications between the vault locations needed to keep them consistent nor those between the vault locations and thinkteam are represented in Fig. 4. In the model of thinkteam we will consider in this paper, we do not explictly address the communications between vaults but assume that they are kept consistent using suitable algorithms. The communication between the vaults and thinkteam will be modelled explicitly.

5 Stochastic Model Checking

In the past, functional and performance analysis of systems were separate areas of research and practice. The latest developments in model checking, extending system verification to performance and dependability aspects, have led to integrated qualitative and quantitative analysis techniques.

In this paper, we use the probabilistic symbolic model checker PRISM [14, 17] which supports, among others, the verification of Continuous Stochastic Logic (CSL) properties over Continuous Time Markov Chains (CTMCs) [13]. CTMCs can be generated by high-level description languages, among which the Performance Evaluation Process Algebra PEPA [12] for which PRISM provides a front end. CSL [1, 2] is a stochastic variant of the well-known Computational Tree Logic CTL [10]. The PEPA expressions used in this paper have this syntax:

$$P ::= (\alpha, r).P \;\Big|\; P \;+\; P \;\Big|\; P \bowtie_L P \;\Big|\; X$$

The basic mechanism to construct behavioural expressions is through prefixing. Component $(\alpha, r).P$ carries out activity (α, r), with action type α and duration Δt determined by rate r. The average duration is given by $1/r$ since, by definition, Δt is an exponentially distributed random variable, with rate r. After performing the activity, the component behaves as P. Component $P + Q$ represents 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 synchronise; both components proceed independently with any activities not occurring in L. The expected duration of a cooperation where activities are shared (i.e. L is not empty) is a function of the expected durations of the corresponding activities in the components, in essence corresponding to the longest one. A special case is the situation in which one component is passive (i.e. has the special rate -) w.r.t. the other component. In this case the total rate is determined by that of the active component only. The behaviour of process variable X is that of P, provided a definition $X = P$ is available for X.

Let I be an interval on the real line, p a probability value and $\bowtie$ a comparison operator, i.e. $\bowtie \in \{<, \leq, \geq, >\}$. Then the syntax of CSL is:

State formulae

$$\Phi ::= a \;\Big|\; \neg\,\Phi \;\Big|\; \Phi \;\vee\; \Phi \;\Big|\; \mathcal{S}_{\bowtie p}\,(\Phi) \;\Big|\; \mathcal{P}_{\bowtie p}\,(\varphi)$$

$\mathcal{S}_{\bowtie p}\,(\Phi)$: probability Φ holds in steady state is $\bowtie p$
$\mathcal{P}_{\bowtie p}\,(\varphi)$: probability a path fulfills φ is $\bowtie p$

Path formulae

$$\varphi ::= X^I\,\Phi \;\Big|\; \Phi\,\mathcal{U}^I\,\Phi$$

$X^I\,\Phi$: next state is reached at $t \in I$ and fulfills Φ
$\Phi\,\mathcal{U}^I\,\Psi$: Φ holds along path until Ψ holds at $t \in I$

The meaning of atomic propositions (a), negation ($\neg$) and disjunction ($\vee$) is standard. Using these operators, other boolean operators can be defined in the usual way. In the variant of CSL used in PRISM, the probability bound $\bowtie\ p$ can be replaced by $=?$, denoting that one is looking for the value of the probability rather than verifying whether it respects a certain bound. The intervals $I = [0, t]$ and $I = [t, \infty)$ are usually written as $\leq t$ and $\geq t$, resp.

6 Stochastic Model of thinkteam

We consider a model composed of three Vault locations (Va, Vb and Vc) containing identical file repositories, three explicitly modelled clients (CA, CB and CC) competing for the same file and the thinkteam application TT. Each Vault location is connected to TT and they communicate their status regularly to TT. Interesting aspects of a Vault location's status for performance analysis purposes are, e.g., its workload, availability (i.e. being up or down) and the bandwith offered to the various Clients. TT keeps a record of the status of all files, like whether a file is locked (i.e. checked out by a Client) or available for download and modification.

Assumptions. The current model is the same as that of [7] and it is based on the following assumptions.

(1) The bandwith between a Client and a Vault is constant and each Client prefers down- and uploading files from the Vault to which it has the best connection. At times this connection may be down, however, in which case a Client will use the next preferred Vault.

(2) Each Client has a static preference list indicating the preferred Vault order.

(3) The three explicitly modelled Clients do not influence significantly the overall performance of the full system (including many active Clients modelled implicitly by the reponsiveness characteristics of the various Vaults). Our aim is to analyse a number of correctness and usability aspects of the system from these three Clients' viewpoint.

(4) We only consider a subset of the operations available to thinkteam Clients, namely the most important ones: *checkOut* and *checkIn*. This keeps the model relatively simple. Further operations could easily be added.

(5) We currently do not enable TT to inform a Client that (s)he can use the local version of the file on her/his desktop if TT notices that the most recent *checkIn* of the requested file was performed by that Client.

Model. The complete PEPA specification is given in Appendix A. For a compact presentation, we depict the Clients, Vaults and TT as kind of stochastic automata in Figs. 5–7, which have also been used in discussions with our colleagues from think3. The labels of the states and transitions play an important role in the next section, when the analyses of the model are discussed. The transition labels are of the general form ⟨from⟩_⟨to⟩_⟨action⟩, in which the ⟨from⟩_⟨to⟩ part indicates the direction of the information flow between processes (e.g. CA_TT denotes a communication from CA to TT) while the ⟨action⟩ part indicates a specific action (e.g. cO_s for successful *checkOut*, cO_f for unsuccessful *checkOut*, cI for *checkIn*).

Client process. The behaviour of a Client CX, with X=A,B,C, is modelled as follows (cf. Fig. 5). Initially, in state CX, with rate λ CX performs a request to TT to download a file for modification. This request is successful when the file is available (CX_TT_cO_s, λ) and fails when the file is currently being edited by another Client (CX_TT_cO_f, λ). If the request is successful, TT provides the address of the 'best possible' Vault location to the CX (e.g. TT_CX_Va means that CX receives the address of Vault A).

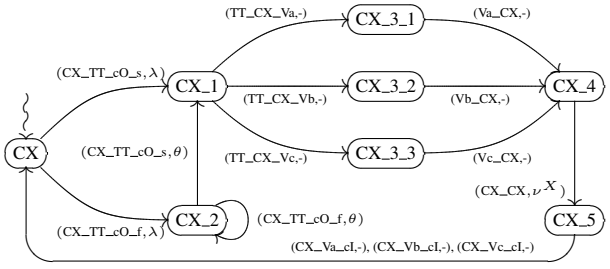

Figure 5. Client CX, with X=A,B,C.

The policy to assign a Vault location is kept very simple in the current model: Each Client receives the address of her/his preferred Vault location (the first on her/his preference list) with highest static probability and the addresses of different Vault locations with lower probabilities. This models the fact that the preferred Vault location is not always available, be it due to a high workload or due to temporary unavailability. These probabilities can be tuned so as to better match the performance characteristics of the system. Indications for such probabilities can be obtained, e.g., from the analysis of the log-files of the performance characteristics of the single vault currently used in thinkteam.

When Client CX has obtained the address of a Vault location, (s)he can *download* (e.g. by (Va_CX,-) from Vault A) the requested file (which is usually part of a document composed of the file to be edited and of files of the parts surrounding the component that are not to be edited), then edit the file while in state CX_4, leaving that state (by (CX_CX,ν^X)) with rate ν^X corresponding to an average time $1/\nu^X$ for editing for this specific Client and, finally, *upload* the file to a Vault by means of a *checkIn* (e.g. by (CX_Va_cI,-) to Vault A), following a preference list as for downloading, and return to initial state CX. Actions with a rate indicated by '-' are passive, i.e. the rate value is established during synchronisation—in this case between the Client and the Vault process, the latter determining its value.

If the Client's request for a file fails, it starts a series of retry actions (in state CX_2) to obtain the file at a later moment. This essentially means that the Client continues to make requests (by (CX_TT_cO_f,θ)) but at a higher rate θ. After a number of successive failed requests, the Client eventually makes a successful request and moves to state CX_1 (PRISM assumes a fair process semantics).

Vault process. The behaviour of a Vault Vy, for y=a,b,c, is modelled as follows (cf. Fig. 6). A Vault (location) can receive *download* operations from a Client (by (Vy_CX,γ_y^X)) with rate γ_y^X corresponding to the average download time $1/\gamma_y^X$ for that specific Client and Vault, alternated with *checkIn* operations ((CX_Vy_cI,μ_y^X)) with rate μ_y^X. After each *checkIn*, the Vault informs TT (by (Vy_TT,δ)) that the file has been *uploaded*. In this way, TT can update the status of the file, i.e. remove the lock and make it available for other Clients. The same communication also models the transfer of status information of the Vaults to TT.

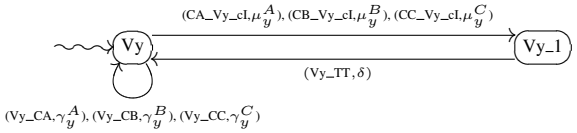

Figure 6. Vault Vy, with y=a,b,c.

TT process. The behaviour of TT is modelled as follows (cf. Fig. 7). Initially, TT waits for file requests from Clients to honour (e.g. from Client A by action (CA_TT_cO_s,-)). In case of a successful file request, TT assigns a Vault to the Client, using the assignment policy described above.

This policy can be modelled stochastically by forcing a race condition between the different assignments as follows. If Client A should be assigned Vault A in ca. 50% of the

cases, Vault B in ca. 33% and Vault C in ca. 16% of the cases, one can choose suitable rates to reflect this. State TT_1_1, e.g., has three outgoing transitions, labelled by (TT_CA_Va,300), (TT_CA_Vb,200) and (TT_CA_Vc,100). The total exit rate from state TT_1_1 is thus $300 + 200 + 100 = 600$ and the probability of Client A being assigned Vault A is then $300/600 = 0.5$. Such relatively high exit rates model the fact that Vault assignment is very fast compared with other activities. As a result, modelling preferences will not significantly influence the model's performance analysis.[1] Actions (TT_CX_Vy,ϵ) model assigning a Vault, locking the file and sending a Client a Vault address.

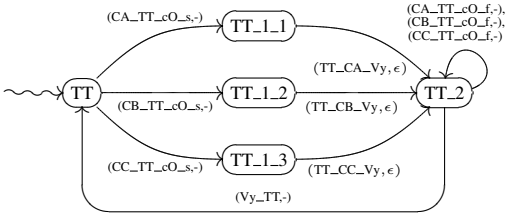

Figure 7. TT, with y=a,b,c.

Any further request for the same file is explicitly denied (e.g. to Client A by (CA_TT_cO_f),-) until TT has received a message from a Vault indicating that the file has been uploaded (e.g. for Vault Y by (Vy_TT,-)). TT is then back in its initial state, ready to accept further requests for the file.

Full specification. The specification of thinkteam is completed by the parallel composition, by means of the PEPA cooperation operator, of the three Client processes, the three Vault processes and the TT process, as follows:

$$\text{USERS} \bowtie_{\text{CX\_TT\_cO\_z,TT\_CX\_Vy,CX\_Vy\_cI,Vy\_CX}} \text{SYSTEM}$$

with USERS=CA⋈CB⋈CC, X=A,B,C, y=a,b,c, z=f,s and SYSTEM=TT$\bowtie_{\text{Vy_TT}}$(Va⋈Vb⋈Vc). The complete PEPA specification is given in Appendix A. It leads to a CTMC with 104 states and 330 transitions.

Note that the model restricts us to the investigation of the performance characteristics for three Clients competing for the same file during approximately the same period. The model can easily be extended with a limited number of explicitly modelled Clients, as in our earlier work addressing a thinkteam model with one centralised Vault [6].

7 Analysis of the thinkteam model

All analyses reported in this section were performed with PRISM v3.1.1 and took a negligible amount of CPU time. The iterative numerical method used for the computation of probabilities was Gauss-Seidel and the accuracy 10^{-6}. See `www.prismmodelchecker.org` and [17] for details.

[1] For reasons of space and readability, these details are abstracted from in Fig. 7: Only a nominal indication ϵ of the relevant rates is given.

While we use the same stochastic model as in [7], we verify a variation of properties for new assumptions on important parameters, namely the edit and download rates. We use the outcome of the analysis of a log-file of actual thinkteam usage by a particular industry and detailed discussions thereof with think3. Note that this analysis refers to only one client's thinkteam use during one specific year. The rate values used for our analyses correspond to the results of the log-file analysis reported in Sect. 3 and are listed in Table 2. These rates should be read considering the letters in the subscripts (superscripts) to refer to the names of the Vaults (Clients). Hence γ_b^A, e.g., is the download rate between Vault B and Client A.

λ	γ_a^A μ_a^A	γ_a^B μ_a^B	γ_a^C μ_a^C	γ_b^A μ_b^A	γ_b^B μ_b^B	γ_b^C μ_b^C	γ_c^A μ_c^A	γ_c^B μ_c^B	γ_c^C μ_c^C	ν^A ν^B ν^C	θ	δ	ϵ	ϵ_2	ϵ_3
0.1	8	4	6	6	8	4	4	6	8	1.35	6	100	100	200	300

Table 2. Rate values for functional analyses.

We set hours as our model's time unit. Rate $\gamma_b^A = 6$, e.g., models that on average the download time between Vault B and Client A is $60/6 = 10$ minutes. This may seem much at first but, as explained in Sect. 2.2, CAD designers use thinkteam to share assemblies, i.e. composite documents referring to several (sometimes thousands) individual model files. Hence, when a Client has to modify one file, many other files (forming the context of this file) need to be downloaded. Rate $\nu^A = 1.35$, e.g., models that on average Client A spends $60/1.35 \approx 44$ minutes editing the file in her/his possession, which indeed corresponds to the mean duration of editing sessions reported in Sect. 3 (cf. Fig. 2).

7.1 Analyses of Functional Properties

Prior to analysing a model's performance it is important to gain confidence in its functional correctness. We verified qualitative properties like deadlock absence, progress properties and mutual exclusion of editing rights on a file.

The probability should be (at most) zero that eventually a deadlock is encountered. This property can be formalised in CSL by using the predefined PRISM label "deadlock", which labels each absorbing state (i.e. a state from which there is a zero probability of exiting), in the following way:

$$\mathcal{P}_{\leq 0}\,([\,\text{true}\ \mathcal{U}\ \text{"deadlock"}\,]), \qquad \text{(P0)}$$

which PRISM verification confirmed to hold for our model.

Whenever Client X succeeds to *checkOut* a file from Vault A, (s)he eventually performs a *checkIn* of that file. This property is captured by the following CSL formula:

$$\mathcal{P}_{\geq 1}\,([\,\text{true}\ \mathcal{U}\ \text{"CXcheckIn"}\,\{\text{"CXcheckOut"}\}\,]), \qquad \text{(P1)}$$

where label "CXcheckIn" is defined as CX_STATE = CX_5 and "CXcheckOut" as CX_STATE = CX_3_1. Verification with PRISM confirmed Formula P1 to hold for our model.

The probability should be (at most) zero that two Clients will eventually obtain permission to modify the same file at the same time. This property can be formalised in CSL as:

$$\mathcal{P}_{\leq 0}\,(\,[\,\text{true}\;\mathcal{U}\;(\text{"OkAB"}\vee\text{"OkAC"}\vee\text{"OkBC"})\,]), \qquad \text{(P2)}$$

in which, for XY=AB,AC,BC, label

$$\text{"OkXY"} \stackrel{\text{def}}{=} (\text{CX\_STATE}=\text{CX\_1}) \wedge (\text{CY\_STATE}=\text{CY\_1}),$$

meaning that two Clients (X and Y) have obtained permission to edit the file (i.e. are in state CX_1 and CY_1, resp.). PRISM confirmed that Formula P2 holds for our model.

7.2 Analyses of Performance Properties

In this section we show performance issues of our model; in particular usability issues seen from a Client perspective.

Swiftness of releasing file. Formula P1 only shows that a Client eventually uploads the file (after downloading it). The below CSL formula can be used to quantify this issue:

$$\mathcal{P}_{=?}\,([\,\text{true}\;\mathcal{U}^{\leq 1}\;\text{"CXcheckIn"}\,\{\text{"CXcheckOut"}\}\,]), \quad \text{(P3)}$$

i.e. what is the probability that within 1 hour after downloading (by a *checkOut*) the file (state CX_3_1), Client X is in state CX_5 ready to upload (by a *checkIn*) the file? The results are presented in Fig. 8 for edit rate ν^X varying from 0.75 to 5, i.e. from an average of 12 minutes to 80 minutes editing the file (corresponding to values within half the standard deviation of the mean in the distribution in Fig. 2). As expected, the less time a Client spends editing, the higher the probability that (s)he returns it within 1 hour.

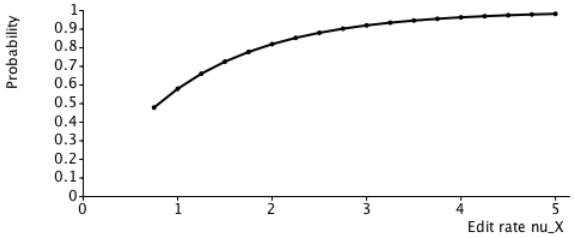

Figure 8. Probability for a Client to *checkIn* a file within one hour after the *checkOut*.

Behaviour on the long run. A parameter influencing the Clients' use of thinkteam, and thus also the time spent on different activities, is the average time that passes between accesses to the same file. The change in average time spent on different activities by a typical Client when varying this inter-access time is visualised in Fig. 9: For each activity we see the percentage of time Client X spends on it, for various values of λ, which has a strong impact on the request rate.

We see that when λ is very low, most time is spent on activities other than editing, retrying and down/uploading. This pattern changes considerably as λ increases. The time spent waiting for the file (retrying) rapidly increases and a Client thus spends much less time on other activities. (S)he also spends more time editing, but this increase is less rapid. It is interesting to note that from a certain point onwards (when $\lambda \approx 0.45$) a Client spends more time retrying to obtain the file than (s)he does actually editing it.

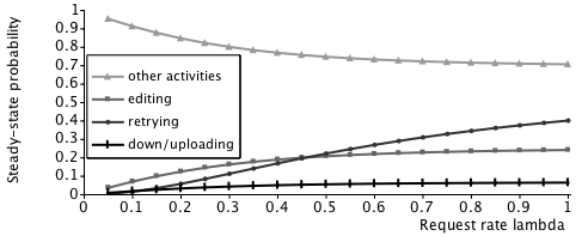

Figure 9. Time a Client spends on different activities, for various file inter-access times.

The properties analysed to obain the results of Fig. 9 are simple steady-state properties formalised in CSL as:

$$\mathcal{S}_{=?}\,([\,\text{"ClientXinStateZ"}\,]), \qquad \text{(P4)}$$

where label "ClientXinStateZ" is replaced by the state indicating the specific activity of interest, e.g. CX_STATE = CX_2 to indicate that Client X is retrying to obtain a file.

Time spent down- and uploading files. The time Clients spend down- and uploading files depends largely on the bandwith of their connection to the Vaults, the file sizes and the workload of the Vaults. Fig. 10 shows the effect that a change in workload of Vaults A and B has on the percentage of time Client A spends down- and uploading files.

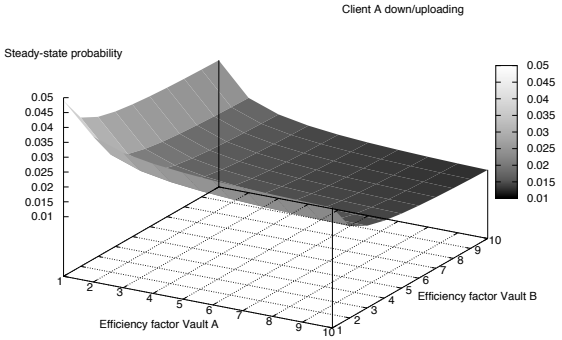

Figure 10. Steady-state probability for a Client to be down- or uploading files.

The x-axis shows an efficiency factor, ranging from 1 to 10, that multiplies the download rates γ_a^X, for X=A,B,C, of Vault A for Clients A, B and C, initially set to 2, 1 and 1.5, resp. Likewise on the y-axis, multiplying the

download rates of Vault B for Clients A, B and C, initialised at 1.5, 2 and 1, resp. All other rates are as in Table 2.

Hence, the higher the efficiency factor, the faster the Vaults perform. Indeed, as expected, we see that the probability that Client A spends time down- and uploading on the long run is smallest when both Vaults (A and B) are working optimally. We also observe that if only Vault B has a high workload, and thus performs slower, then this influences the time Client A spends down- and uploading. This is because part of the time Client A downloads from (and uploads to) Vault B. Another observation is that this percentage does not decrease much after a certain performance of the Vaults has been reached: This occurs more or less at efficiency factor 5, for the parameter settings chosen for the analysis. The results in Fig. 10 have been obtained by verifying Formula P4 for Client A being in either of the states CA_3_1, CA_3_2, CA_3_3 or CA_5.

Number of retries per success. The perceived usability of thinkteam also depends on how often a Client is unable to obtain a file (s)he intends to modify. Failing to obtain a file means the Client needs to spend time on either keep trying to obtain it at a later stage or change her/his workplan. If this situation occurs frequently, it might be perceived as annoying by the Client. Moreover, it may lead to the introduction of errors (the Client may forget to edit the file later, or forget what modifications to make) or to problems in the overall workflow plan, and result in a delay in the delivery of the final product. It would therefore be useful to be able to quantify this problem under various conditions and for different user profiles of thinkteam Clients. For instance, the different design phases may induce a different use of thinkteam: Initially, Clients may need more time to modify a file because of a completely new design, but closer to the deadline there might be a phase in which many Clients need to frequently make small modifications in order to obtain a fine-tuning of the various components (cf. Fig. 1).

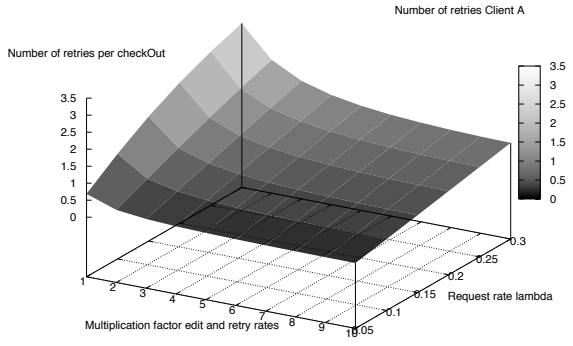

Figure 11. Average number of retries a Client performs to obtain a file.

Fig. 11 shows the results of one such an analysis. It shows how the average number of retries needed by Client A to obtain the permission to modify a file changes with the simultaneous increase of all Clients' edit and retry rates (leading to shorter average editing times and more frequent retrying attempts) and an increase in the frequency with which Clients need the file, i.e. modelling the aforementioned Client behaviour close to a deadline.

The chosen edit and retry rates were initialised at 0.5 and 1, resp., which have been multiplied with a factor ranging from 1 to 10, while the request rate (whose inverse gives the average time that passes between a *checkIn* of a file and the following request to modify the same file) ranges from 0.05 to 0.3. We thus consider, on the one hand, average Client behaviour ranging from editing the file for 2 hrs. and retrying to checkOut the file once every hour (x-axis' left end) to editing the file for only 12 minutes and retrying to checkOut the file as much as every 6 minutes (x-axis' right end) and, on the other hand, the time that passes between a file's *checkIn* and a subsequent request to *checkOut* that same file to range from only roughly once a day (y-axis' left end) to as often as once every 3 hrs (y-axis' right end).

Fig. 11 was obtained by extending the PRISM specification of our model with a reward structure to count the number of successful requests and another to count the number of retries. Reward structures defined over a CTMC define which actions need to be counted. The model can then be analysed w.r.t. a number of reward-based properties. We use the steady-state reward formula that calculates the average reward on the long run. This is formalised in PRISM as:

$$\mathcal{R}\{\text{"label"}\}=?\,[\,S\,],$$

in which "label" is a reward structure and S denotes that the steady-state reward is calculated. The number of repeated failed requests per successful request that is shown in Fig. 11 was thus obtained by dividing the outcome of this formula for label=NrFailedRequestsClientA by that for label=NrSuccessfulRequestsClientA.

We see in Fig. 11 that the number of retries a Client needs to perform to obtain the file increases when, on average, Clients spend a longer time editing the file while their attempts to retry to *checkOut* the file are not so frequent (towards the x-axis' left end), in particular if the time between a file's *checkIn* and subsequent *checkOut* request simultaneously decreases (towards the y-axis' right end).

Having more detailed data on a particular usage context allows a more accurate stochastic analysis using the same model. The specific usage characteristics of thinkteam in the studied environment show that the number of retries per successful *checkOut* on average is quite acceptable, also when edit and retry rates go up close to a deadline period during which many Clients need to frequently make small modifications to the file (towards the x-axis' right end).

Analyses of the same model in [7], for less favourable situations assuming much longer editing sessions, show the number of retries per success may become unacceptably high. In such cases, a waiting-list policy can be shown more suitable than a retry policy, reducing the time Clients waste retrying to obtain the files. A similar observation has been confirmed by the analysis performed in [16] with a different analysis technique, allowing one to consider the dynamic effects of the presence of a much larger number of Clients.

8 Conclusions and Future Work

We performed formal analyses of qualitative and quantitative properties of a small, yet relevant industrial case study concerning think3's PDM application thinkteam. We modelled the added functionality of replicated vaults with the stochastic process algebra PEPA and verified properties expressed in the stochastic temporal logic CSL over the model with the stochastic model checker PRISM. The verified properties include qualitative properties like vault assignment, mutual exclusion of editing rights and absence of deadlocks and, in particular, performance properties providing information on the usability aspects of thinkteam under different assumptions on the quality of service of connections, vault availabilities and work patterns of users.

Our graphical representation of the PEPA specification helped to develop and discuss the model with our colleagues from industry. The approach taken is to develop and study a rather limited, abstract base model of a case study proposed by industry, and to use this model to verify properties that provide insight in the dependencies of various aspects of the model and their effect on system performance from an easy to understand user perspective. This approach thus implicitly forms a 'hands-on' practical experience to illustrate the potential of stochastic model checking for the analysis of software systems in the manufacturing industry and also of its current limitations. One concern is scalability. A model addressing the dynamic behaviour of many users accessing many files would provide a more accurate view on performance. We recently experimented with a preliminary complementary PEPA model of thinkteam that can be used to generate ordinary differential equations [16]. This allows the evaluation of systems with a very high number of replicated, independent components, at the cost of abstracting from the components' identities. Its further development and a comparison with this paper's model are ongoing.

References

[1] A. Aziz, K. Sanwal, V. Singhal and R. Brayton, Model checking continuous time Markov chains. *ACM Trans. on Computational Logic* 1, 1 (2000), 162–170.

[2] C. Baier, J.-P. Katoen and H. Hermanns, Approximate symbolic model checking of continuous-time Markov chains. In *CONCUR'99*, *LNCS* 1664, 1999, 146–162.

[3] C. Baier, B. Haverkort, H. Hermanns and J.-P. Katoen, Automated performance and dependability evaluation using model checking. In *Performance Evaluation of Complex Systems*, *LNCS* 2459, 2002, 261–289.

[4] M.H. ter Beek, M. Massink, D. Latella, S. Gnesi, A. Forghieri and M. Sebastianis, Model checking publish/subscribe notification for thinkteam®. In *FMICS'04*, *ENTCS* 133, 2005, 275–294.

[5] M.H. ter Beek, M. Massink, D. Latella, S. Gnesi, A. Forghieri and M. Sebastianis, A Case Study on the Automated Verification of Groupware Protocols. In *ICSE'05*, ACM, 2005, 596–603.

[6] M.H. ter Beek, M. Massink and D. Latella, Towards model checking stochastic aspects of the thinkteam user interface. In *DSVIS'05*, *LNCS* 3941, 2006, 39-50.

[7] M.H. ter Beek, S. Gnesi, D. Latella, M. Massink, M. Sebastianis and G. Trentanni, Assisting the Design of a Groupware System. ISTI Technical Report. Submitted, 2008. Available at URL: http://www1.isti.cnr.it/~Massink/TerBeekEtAL_TR2008.pdf

[8] E. Brinksma, H. Hermanns and J.-P. Katoen (Eds.), *Lectures on Formal Methods and Performance Analysis*, *LNCS* 2090, 2001.

[9] P. Buchholz, J.-P. Katoen, P. Kemper and C. Tepper, Model-checking large structured Markov chains. *JLAP* 56 (2003), 69–96.

[10] E. Clarke, E. Emerson and A. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. on Programming Languages and Systems* 8 (1986), 244–263.

[11] H. Hermanns, J-P. Katoen, J. Meyer-Kayser and M. Siegle, A tool for model-checking Markov chains. *STTT* 4, 2 (2003), 153–172.

[12] J. Hillston, *A Compositional Approach to Performance Modelling*. Cambridge University, 1996.

[13] V. Kulkarni. *Modeling and Analysis of Stochastic Systems*. Chapman & Hall, 1995.

[14] M. Kwiatkowska, G. Norman and D. Parker, Probabilistic symbolic model checking with PRISM: A hybrid approach. In *TACAS'02*, *LNCS* 2280, 2002, 52–66.

[15] R. Levesque, SPSS Programming and Data Management, SPSS Inc., 2007. URL: http://www.spss.com

[16] M. Massink, D. Latella, M.H. ter Beek, M.D. Harrison and M. Loreti, A Fluid Flow Approach to Usability Analysis of Multi-user Systems. To appear in *HCSE'08*, *LNCS*, 2008. Available at URL: http://www1.isti.cnr.it/~Massink/t3ode.pdf

[17] D. Parker, G. Norman and M. Kwiatkowska, *PRISM 2.0 — Users' Guide*, February 2004.

[18] J. Stark, *Product Lifecycle Management: 21st Century Paradigm for Product Realisation*. Springer, 2005.

[19] H. Younes and R. Simmons, Probabilistic verification of discrete event systems using acceptance sampling. In *CAV'02*, *LNCS* 2404, 2002, 223–235.

A PEPA Specification (TT, 3 Users, 3 Vaults)

```
lambda = 0.1;     %request rate (client/hr)
gamma_a_A = 8.0;  %download rate (file/hr) between Vault A-Client A
gamma_a_B = 4.0;  %download rate (file/hr) between Vault A-Client B
gamma_a_C = 6.0;  %download rate (file/hr) between Vault A-Client C
gamma_b_A = 6.0;  %download rate (file/hr) between Vault B-Client A
gamma_b_B = 8.0;  %download rate (file/hr) between Vault B-Client B
gamma_b_C = 4.0;  %download rate (file/hr) between Vault B-Client C
gamma_c_A = 4.0;  %download rate (file/hr) between Vault C-Client A
gamma_c_B = 6.0;  %download rate (file/hr) between Vault C-Client B
gamma_c_C = 8.0;  %download rate (file/hr) between Vault C-Client C
mu_a_A = 8.0;     %upload rate (file/hr) between Vault A-Client A
mu_a_B = 4.0;     %upload rate (file/hr) between Vault A-Client B
mu_a_C = 6.0;     %upload rate (file/hr) between Vault A-Client C
mu_b_A = 6.0;     %upload rate (file/hr) between Vault B-Client A
mu_b_B = 8.0;     %upload rate (file/hr) between Vault B-Client B
mu_b_C = 4.0;     %upload rate (file/hr) between Vault B-Client C
mu_c_A = 4.0;     %upload rate (file/hr) between Vault C-Client A
mu_c_B = 6.0;     %upload rate (file/hr) between Vault C-Client B
mu_c_C = 8.0;     %upload rate (file/hr) between Vault C-Client C
nu_A = 1.35;      %edit rate (file/hr) of Client A
nu_B = 1.35;      %edit rate (file/hr) of Client B
nu_C = 1.35;      %edit rate (file/hr) of Client C
theta = 6.0;      %retry rate (client/hr)
delta = 100;      %signal rate (message/hr)
epsilon = 100;    %signal rate (message/hr)
epsilon2 = 200;   %(a trick to raise the probability of
epsilon3 = 300;   %specific "nondeterministic" choices)

% Client A (Client B and Client C are analogous):
#CA = (CA_TT_cO_s,lambda).CA_1      %successful request for checkOut
    + (CA_TT_cO_f,lambda).CA_2;     %failed request for checkOut
#CA_1 = (TT_CA_Va,infty).CA_3_1     %TT assigns Vault A for checkOut
      + (TT_CA_Vb,infty).CA_3_2     %TT assigns Vault B for checkOut
      + (TT_CA_Vc,infty).CA_3_3;    %TT assigns Vault C for checkOut
#CA_3_1 = (Va_CA,infty).CA_4;       %checkOut file from Vault A
#CA_3_2 = (Vb_CA,infty).CA_4;       %checkOut file from Vault B
#CA_3_3 = (Vc_CA,infty).CA_4;       %checkOut file from Vault C
#CA_4 = (CA_CA,nu_A).CA_5;          %edit file
#CA_5 = (CA_Va_cI,infty).CA         %checkIn file in Vault A
      + (CA_Vb_cI,infty).CA         %checkIn file in Vault B
      + (CA_Vc_cI,infty).CA;        %checkIn file in Vault C
#CA_2 = (CA_TT_cO_s,theta).CA_1     %successful retry of checkOut
      + (CA_TT_cO_f,theta).CA_2;    %failed retry of checkOut

% Vault A (Vault B and Vault C are analogous):
#Va = (Va_CA,gamma_a_A).Va          %file checkOut by Client A
    + (Va_CB,gamma_a_B).Va          %file checkOut by Client B
    + (Va_CC,gamma_a_C).Va          %file checkOut by Client C
    + (CA_Va_cI,mu_a_A).Va_1        %file checkIn by Client A
    + (CB_Va_cI,mu_a_B).Va_1        %file checkIn by Client B
    + (CC_Va_cI,mu_a_C).Va_1;       %file checkIn by Client C
#Va_1 = (Va_TT,delta).Va;           %inform TT of file checkIn

% thinkteam:
#TT = (CA_TT_cO_s,infty).TT_1_1     %grant checkOut to Client A
    + (CB_TT_cO_s,infty).TT_1_2     %grant checkOut to Client B
    + (CC_TT_cO_s,infty).TT_1_3;    %grant checkOut to Client C
#TT_1_1 = (TT_CA_Va,epsilon3).TT_2  %assign Vault A to Client A
        + (TT_CA_Vb,epsilon2).TT_2  %assign Vault B to Client A
        + (TT_CA_Vc,epsilon).TT_2;  %assign Vault C to Client A
#TT_1_2 = (TT_CB_Va,epsilon).TT_2   %assign Vault A to Client B
        + (TT_CB_Vb,epsilon3).TT_2  %assign Vault B to Client B
        + (TT_CB_Vc,epsilon2).TT_2; %assign Vault C to Client B
#TT_1_3 = (TT_CC_Va,epsilon2).TT_2  %assign Vault A to Client C
        + (TT_CC_Vb,epsilon).TT_2   %assign Vault B to Client C
        + (TT_CC_Vc,epsilon3).TT_2; %assign Vault C to Client C
#TT_2 = (Va_TT,infty).TT            %Vault A signals file checkIn
      + (Vb_TT,infty).TT            %Vault B signals file checkIn
      + (Vc_TT,infty).TT            %Vault C signals file checkIn
      + (CA_TT_cO_f,infty).TT_2     %deny checkOut to Client A
      + (CB_TT_cO_f,infty).TT_2     %deny checkOut to Client B
      + (CC_TT_cO_f,infty).TT_2;    %deny checkOut to Client C

% full specification:
(CA <> CB <> CC)
      <CA_TT_cO_s,CB_TT_cO_s,CC_TT_cO_s,
       CA_TT_cO_f,CB_TT_cO_f,CC_TT_cO_f,
       TT_CA_Va,TT_CA_Vb,TT_CA_Vc,
       TT_CB_Va,TT_CB_Vb,TT_CB_Vc,
       TT_CC_Va,TT_CC_Vb,TT_CC_Vc,
       CA_Va_cI,CA_Vb_cI,CA_Vc_cI,
       CB_Va_cI,CB_Vb_cI,CB_Vc_cI,
       CC_Va_cI,CC_Vb_cI,CC_Vc_cI,
       Va_CA,Va_CB,Va_CC,
       Vb_CA,Vb_CB,Vb_CC,
       Vc_CA,Vc_CB,Vc_CC>
(TT <Va_TT,Vb_TT,Vc_TT> (Va <> Vb <> Vc))
```