Performance and Availability Assessment for the Configuration of Distributed Workflow Management Systems♣

Michael Gillmann[1], Jeanine Weissenfels[1], Gerhard Weikum[1], Achim Kraiss[2]

[1]University of the Saarland, Germany
{gillmann,weissenfels,weikum}@cs.uni-sb.de
http://www-dbs.cs.uni-sb.de/

[2]Dresdner Bank AG, Germany
achim.kraiss@dresdner-bank.com
http://www.dresdner-bank.com/

Abstract

Workflow management systems (WFMSs) that are geared for the orchestration of enterprise-wide or even "virtual-enterprise"-style business processes across multiple organizations are complex distributed systems. They consist of multiple workflow engines, application servers, and ORB-style communication servers. Thus, deriving a suitable configuration of an entire distributed WFMS for a given application workload is a difficult task.

This paper presents a mathematically based method for configuring a distributed WFMS such that the application's demands regarding performance and availability can be met while aiming to minimize the total system costs. The major degree of freedom that the configuration method considers is the replication of the underlying software components, workflow engines and application servers of different types as well as the communication server, on multiple computers for load partitioning and enhanced availability. The mathematical core of the method consists of Markov-chain models, derived from the application's workflow specifications, that allow assessing the overall system's performance, availability, and also its performability in the degraded mode when some server replicas are offline, for given degrees of replication. By iterating over the space of feasible system configurations and assessing the quality of candidate configurations, the developed method determines a configuration with near-minimum costs.

1 Introduction

1.1 Problem Statement

The main goal of workflow management systems (WFMSs) is to support the efficient, largely automated execution of business processes. Large enterprises demand the reliable execution of a wide variety of workflow types. For some of these workflow types, the availability of the components of the underlying, often distributed WFMS is crucial; for other workflow types, high throughput and short response times are mandatory. However, finding a configuration of the WFMS (e.g., with replicated components) that meets all requirements is a difficult problem. Moreover, it may be necessary to adapt the configuration over time due to changes of the workflow load, e.g., upon adding new workflow types. Therefore, it is not sufficient to find an appropriate initial configuration; it should rather be possible to reconfigure the WFMS dynamically. The first step towards a (dynamic) configuration tool is the analysis of the WFMS to predict the performance and the availability that would be achievable under a new configuration.

The goal of our research is to build a configuration tool based on a system model that is able to predict the best configuration for a given workflow load. The configuration tool should optimize the ratio between performance and cost, or availability and cost, or even the combination of both, the so-called "performability".

1.2 Contribution

In this paper, we consider distributed WFMSs that consist of components like workflow engines, application servers, and communication servers such as ORBs. The WFMS can be

♣ This work was performed within the research project "Architecture, Configuration, and Administration of Large Workflow Management Systems" funded by the German Science Foundation (DFG).

configured such that each of these components may be replicated on different computers for availability and/or load partitioning. We present an analytic approach that considers both the performance and the availability of the entire WFMS in its assessment of a given configuration. The approach is based on stochastic methods [19, 20], specifically continuous-time Markov chains (CTMC), and shows the suitability of these models for a new application field. The developed analytic model allows us to rank the performance and availability of different configurations that use replicated components. Moreover, we can predict the performance degradation caused by transient failures and repair or downtime periods of servers (e.g., for upgrading software etc.). These considerations lead to the notion of "performability" [19], a combination of performance and availability metrics. From the analytic model we can also derive the necessary number of WFMS component replications to meet specified goals for performance and availability. So a crucial part of a configuration tool for distributed WFMS becomes analytically tractable, and no longer depends on expensive trial-and-error practice or the subjective intuition of the system administration staff.

1.3 Related Work

Although the literature includes much work on scalable WFMS architectures [1, 4, 5, 6, 12, 15], there are only few research projects that have looked into the quantitative assessment of WFMS configurations with regard to performance and availability. The work reported in [2, 3] presents several types of distributed WFMS architectures and discusses the influence of different load distribution methods on the network and workflow-server load, mostly using simulations. [18] presents heuristics for the allocation of workflow-type and workflow-instance data onto servers. Mechanisms for enhanced WFMS availability by replicating state data on a standby backup server have been studied in [9, 14]. None of this prior work has addressed the issue of how to configure a WFMS for given performance and availability goals.

The use of CTMC models in the context of workflow management has been pursued by [13]. This work uses the steady-state analysis of such models to analyze the efficiency of different outsourcing strategies in a virtual-enterprise setting. Our approach is more far-reaching in that we use methods for the transient analysis of Markov chains to estimate the dynamic behavior of workflow instances and the resulting performance. In addition, we address also the availability and performability dimensions, which are beyond the scope of [13].

1.4 Outline

The rest of the paper is organized as follows. In Section 2, we introduce our model of a distributed WFMS. In Section 3, we describe how we can stochastically model the dynamic behavior of a workflow instance; we use a simplified e-commerce application as an illustrating example. In Sections 4 and 5, we develop the performance model and the availability model, respectively. In Section 6, we combine both models into the performability model that allows us to predict the influence of transient failures and downtime periods on the overall performance. Section 7 discusses how the presented models are integrated into the core of an automated configuration tool.

2 Architectural Model

In this section, we introduce an architectural model for distributed WFMSs. We basically follow the framework of [23]. Although the model is simple, it is powerful enough to capture the architecture models of most WFMS products and research prototypes in a reasonable way. Based on this model, we will introduce the central notions of the *system configuration* and the *system state* of a distributed WFMS.

A *workflow (instance)* is a set of *activities* that are spawned according to the control-flow specification of a given *workflow type*. An activity can either directly invoke an application, which is typical of automated activities, or it can first require the assignment to an appropriate human actor or organizational unit according to a specified worklist management policy.

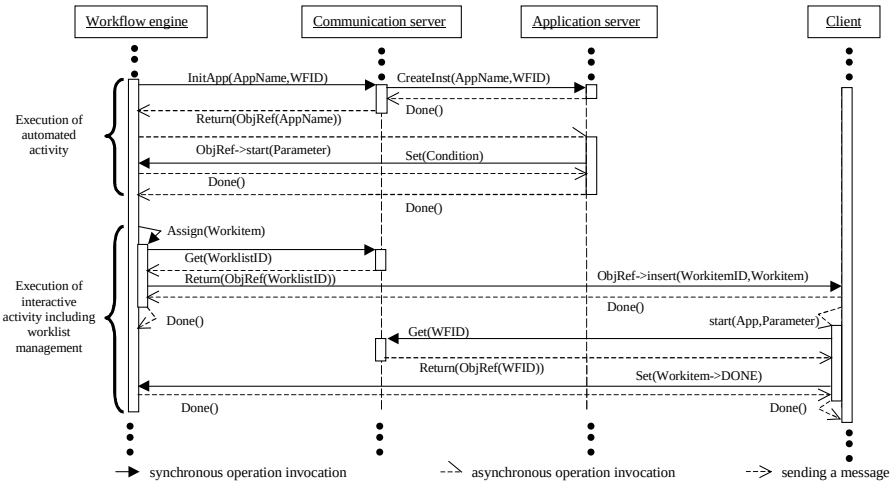

Figure 1: Sequence diagram of the execution of two activities

A distributed WFMS executes workflow instances in a decentralized manner: each workflow instance is partitioned into several *subworkflows* which may run on different *workflow engines*, for example, with one workflow engine per subworkflow type according to the organizational structure of the involved enterprises. Invoked applications of specific types run on dedicated *application servers*, for example, under the control of a Web application server and often with a database system as a backend. Finally, the communication within the underlying, often widely distributed and heterogeneous system environment is assumed to be handled by a special kind of *communication server*, for example, an object request broker (ORB) or a similar piece of middleware. These three types of WFMS components – workflow engines, application servers, and communication servers – will henceforth be viewed as abstract servers of specific types within our architectural model. For simplicity, we assume that each such server resides on a dedicated computer, and that all involved computers are connected by an intranet or the Internet. The situation where multiple servers run on the same computer can be addressed within our model, too, but would entail some technical extensions.

The interaction of the various components on behalf of a workflow instance is illustrated in the UML-style sequence diagram of Figure 1. Note that each activity involves exactly one workflow engine of a specific type, one application server of a given type, and the communication server. Each activity incurs a certain, activity-specific processing load on these servers. The first part of Figure 1 shows the sequence of the requests for the asynchronous execution of an automated activity. The second part of Figure 1 shows the sequence of the requests for an interactive activity. As that activity is executed on a client machine, the application server is not involved. The specific details of how many requests are sent at which timepoints between the various servers is not relevant, however, as far as the performance assessment and configuration planning is concerned. Rather we consider only the total load induced by an activity instance on each of the involved server types. So, in our example of Figure 1, the execution of the automated activity induces 3 requests at the workflow engine, 2 requests at the communication server, and 3 requests at the application server.

For scalability and availability reasons, many industrial-strength WFMSs support *the replication of server types* within the system. For simplicity, we will refer to the replicas of a server type as *servers*. For example, a workflow engine that is capable to handle instances of specific subworkflow types can be installed on multiple computers, with the total load being partitioned across all such servers, e.g., by assigning them subworkflow instances in a round-robin or hashed manner. In addition, each server provides capabilities for backup and online failover in the case that another server of the same type fails or is taken down for maintenance.

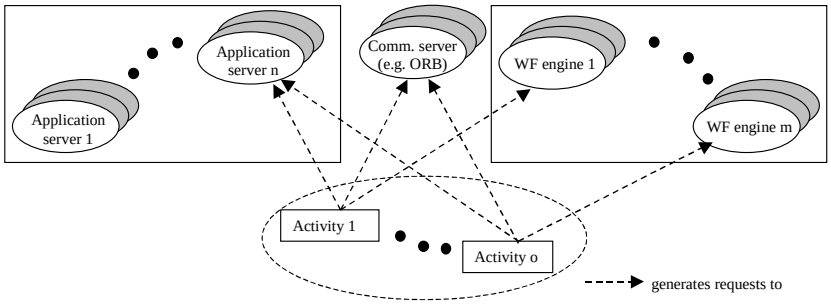

Figure 2: Architectural model of a distributed WFMS

In that case, the total load would be distributed across one less server, leading to (temporarily) degraded performance.

Figure 2 illustrates the presented architectural model: the WFMS consists of one type of communication server, *m* different types of workflow engines, and *n* different types of application server. The arcs denote service requests of workflow activities to the several server types. For example, the execution of an activity of type 1 requires work on the communication server, a workflow engine of type 1 and an application server of type *n*.

Each server of type *x* is assumed to have a failure rate λ_x and a repair rate μ_x. These rates correspond to the reciprocals of the mean time to failure and the mean time to repair, respectively. Here the notion of a failure includes downtimes for maintenance, and the repair time is the duration of a restart (including all necessary recovery steps) after a real failure or downtime in general.

Note that our architectural model could be easily extended to include more server types, for example, to incorporate directory services or worklist management facilities as separate servers if this were desired. The three server types made explicit in our model appear to be the most relevant ones for performance and availability assessment. Also note that we do not include clients as explicit components in the model, for the simple reason that client machines are usually not performance-critical. Rather the shared, and heavily utilized resources of servers usually form the bottlenecks in multi-user applications. Finally, we disregard all effects of human user behavior, e.g., their speed of reaction, intellectual decision making etc., for the assessment of workflow turnaround times, as these aspects are beyond the control of the computer system configuration.

We are now ready to define the central notion of the *system configuration* of a distributed WFMS. With *k* different server types, the system configuration of the WFMS is the vector of replication degrees $(Y_1,\ldots,Y_k)$ for each server type; so Y_x is the number of servers of server type *x* that we have configured the system with. Because of server failures and repairs, the number of available servers of a given type varies over time. For a given point of time, we call the vector $(X_1,\ldots,X_k)$ (with $X_x \le Y_x$ for all $1 \le x \le k$) of the numbers of currently available servers of each server type the current *system state* of the WFMS.

3 Stochastic Modeling of Workflow Behavior

In this section, we present a model that stochastically describes the behavior of a single workflow instance. In Subsection 3.1, we will first introduce a simplified electronic-commerce scenario as an illustrating example. In Subsection 3.2, we develop the stochastic model that allows us to estimate the (expected) number of activity executions for each activity type within a workflow instance of a given type. For the sake of concreteness, we will use state and activity charts as a workflow specification language, but other, comparable languages could be incorporated in our approach as well.

3.1 Example Scenario

As an example of a workflow type, we present a simplified e-commerce scenario. To underline the completeness of our approach, we include the full spectrum of control flow structures, i.e., branching splits, parallelism, joins, and loops. The workflow is similar to the TPC-C benchmark for transaction systems [21], with the key difference that we combine multiple transaction types into a workflow and further enhance the functionality (see [7] for a full description of this workflow).

The workflow specification is given in the form of a state chart [10, 11]. This specification formalism has been adopted for the behavioral dimension of the UML industry standard [22], and it has been used for our own prototype system Mentor-lite [16, 24]. State charts specify the control flow between activities. A state chart is essentially a finite state machine with a distinguished initial state and transitions driven by event-condition-action rules (ECA rules). Throughout this paper, we assume that each workflow state chart has a single final state (i.e., one without outgoing edges). (If there were multiple final states, they could be easily connected to an additional termination state.) A transition from state s_i to state s_j, annotated with an ECA rule of the form *E[C]/A*, fires if event *E* occurs and condition *C* holds. The effect is that state s_i is left, state s_j is entered, and action *A* is executed. Conditions and actions are expressed in terms of variables that are relevant for the control and data flow among activities. In addition, an action *A* can explicitly start an activity, expressed by *st!(activity)*, and can generate an event *E* or modify a condition variable *C* (e.g., *fs!(C)* sets the condition *C* to false). Each of the three components of an *E[C]/A* triple may be empty.

Important additional features of state charts are nested states and orthogonal components. Nesting of states means that a state can itself contains an entire state chart. The semantics is that upon entering the higher-level state, the initial state of the embedded lower-level state chart is automatically entered, and upon leaving the higher-level state all embedded lower-level state charts are left. The capability for nesting states is especially useful for the refinement of specifications during the design process and for incorporating subworkflows. Orthogonal components denote the parallel execution of two state charts that are embedded in the same higher-level state (where the entire state chart can be viewed as a single top-level state). Both components enter their initial states simultaneously, and the transitions in the two components proceed in parallel, subject to the preconditions for a transition to fire.

Figure 3 shows the top-level state chart for our example workflow. Each state corresponds to an activity or one (or multiple, parallel) subworkflow(s), except for initial and final states. We assume that for every activity *act* the condition *act*_DONE is set to true when *act* is finished. So, we are able to synchronize the control flow so that a state of the state chart is left when the corresponding activity terminates. For parallel subworkflows, the final states of the corresponding orthogonal components serve to synchronize the termination (i.e., join in the control flow).

The workflow behaves as follows. Initially, the *NewOrder* activity is started. After the termination of *NewOrder*, the control flow is split. If the customer wants to pay by credit card, the condition *PayByCreditCard* is set and the *CreditCardCheck* activity checks the validity of the credit card. If there are any problems with the credit card, the workflow is terminated. Otherwise the shipment, represented by the nested top-level state *Shipment_S*, is initiated spawning two orthogonal/parallel subworkflows, specified in the state charts *Notify_SC* and *Delivery_SC* , respectively. After the termination of both subworkflows, the control flow is synchronized, and split again depending on the mode of payment. The workflow terminates in the finishing state *EP_EXIT_S*.

3.2 Stochastic modeling

For predicting the expected load induced by the execution of a workflow instance, we have to be able to predict the control flow behavior of workflow instances. As workflows include conditional branches and loops, the best we can do here is to describe the execution

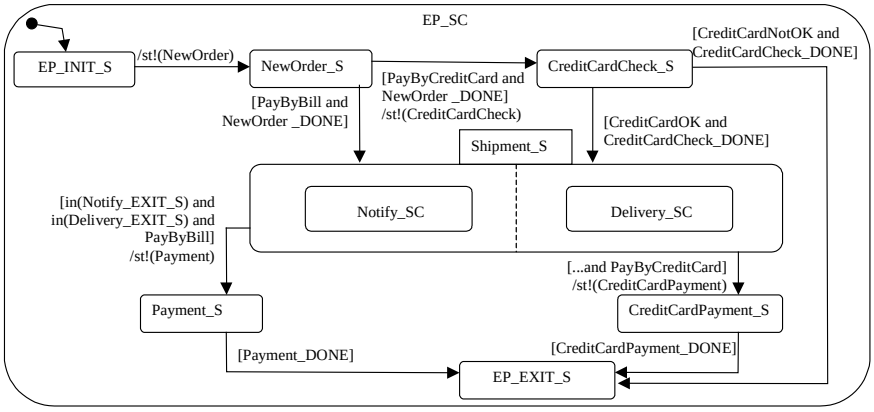

Figure 3: State chart of the electronic purchase (EP) workflow example

stochastically. Our goal thus is to estimate the number of activity invocations per workflow instance, for each activity type; from this estimate we can then derive the load induced by a workflow instance on the various server types. In the following, we will concentrate on workflows without nesting, and will come back to the general case later in Section 4 when we show how to incorporate subworkflows in the overall model.

A suitable stochastic model for describing the control flow within a simple workflow instance without nested subworkflows is the model of continuous-time, first-order Markov chains (CTMC) [19, 20]. A CTMC is a process that proceeds through a set of states in certain time periods. Its basic property is that the probability of entering the next state within a certain time depends only on the currently entered state, and not on the previous history of entered states. The mathematical implication is that the residence time in a state - that is, the time the process resides in the state before it makes its next transition - follows a (state-specific) exponential distribution. Consequently, the behavior of a CTMC is uniquely described by a matrix $P = (p_{ij})$ of transition probabilities between states and a vector $H = (H_i)$ of the mean residence times of the states.

Let $\{ s_i \mid i = 0 \,..\, n-1 \}$ be the set of *n* execution states of a workflow type *t*. The control flow of an instance of *t* will be modeled by a CTMC where the states correspond to the workflow execution states s_i. The state transition probability p_{ij} corresponds to the probability that a workflow instance of workflow type *t* enters state s_j when leaving state s_i. The transition probabilities have to be provided by the workflow designer based on the semantics of the conditions between the workflow activities and the anticipated frequencies of business cases. If the entire workflow application is already operational and our goal is to reconfigure the WFMS (or investigate if a reconfiguration is worthwhile), then the transition probabilities can be derived from audit trails of previous workflow executions. The mean residence time H_i of a state *i* corresponds to the mean time that instances of workflow type *t* stay in the execution state s_i, i.e., the turnaround time of the corresponding activity (or the mean runtime of the corresponding nested subworkflow), and needs to be estimated or observed analogously. In accordance with the workflow specification, we assume that the CTMC has a single initial state s_0. In the initial state-probability vector of the CTMC, the probability is set to *1* for the initial state s_0 and to *0* for all other states. Moreover, we add a transition from the final execution state into an artificial absorbing state s_A. The transition probability of this transition is set to *1*, and the residence time of the absorbing state is set to infinity.

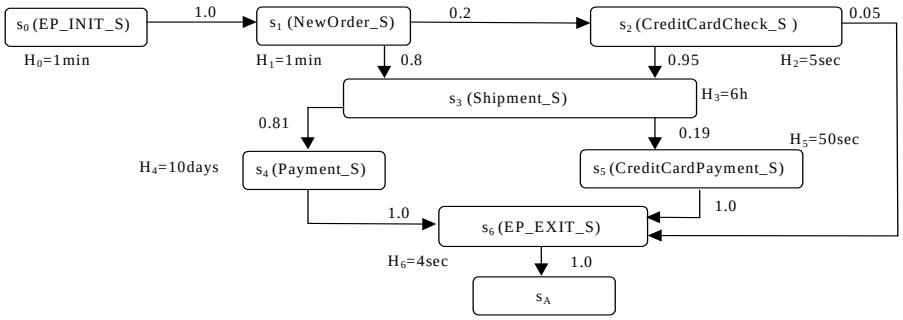

Figure 4: CTMC representing the EP workflow type

For workflow types with subworkflows, the subworkflows are represented by single states within the CTMC of the parent workflow. In the case of parallelism, the corresponding state represents all parallel subworkflows together. For the mean residence time of that state, we will use the maximum of the mean turnaround times of the parallel subworkflows.

Figure 4 gives an example for the CTMC representing the e-commerce workflow type of Figure 3. Besides the absorbing state s_A, the CTMC consists of seven further states, each representing the seven states of the workflow's top-level state chart. The values for the transition probabilities and the mean residence times are fictitious for mere illustration. With this CTMC model, we are now able to predict the expected number of invocations for each activity type, namely, the number of visits to the corresponding CTMC state before eventually reaching the absorbing state, using standard analysis techniques for Markov chains. We will provide more details in the following section, where we will also show how to derive the expected total load of a workflow instance.

4 Performance Model

In this section, we discuss the server-performance model. We proceed in four stages:

1. We analyze the *mean turnaround time* R_t of a workflow of a given type *t*, based on the analysis of state visit frequencies and the state residence times of the CTMC model. This analysis makes use of standard techniques for the transient behavior of Markov chains. (Note that the Markov chains in our approach are non-ergodic; so stationary state probabilities do not exist, and a steady-state analysis is not feasible.)
2. We determine *load* induced on each server type by a workflow instance of type *t*. We will model this load as the *expected number of service requests* to a server type. Technically, this is the most difficult step, and we will use a Markov reward model for this derivation.
3. We then aggregate, for each server type, the load over all workflow instances of all types (using the relative fractions of the various workflow types as implicit weights). The *total load* per server of a given server type is then obtained by dividing the overall load by the number of such servers (i.e., the server-type-specific degree of replication) in the configuration. In this stage we also derive the *maximum sustainable throughput* in terms of workflow instances per time unit.
4. Finally we derive, from the turnaround times of workflows and the total load per server, the *mean waiting times* of service requests caused by queueing at a heavily loaded server. This is a direct measure for the system's responsiveness as perceived by human users in interactions for the activities within a workflow. Overly high waiting times are an indication of a poorly configured system, and we can then identify the server type(s) that forms the bottleneck.

4.1 Workflow Turnaround Time

We derive the mean turnaround time of a workflow instance of type t by the transient analysis of the corresponding CTMC [20]. The mean turnaround time, R_t, is the mean time that the CTMC needs to enter the absorbing state for the first (and only) time, the so-called *first-passage time* of the absorbing state s_A.

The first-passage time of a CTMC state is generally computed by solving a set of linear equations [20] as follows. Assume that for all states s_i of the CTMC the probability that the first epoch at which the CTMC makes a transition into the absorption state s_A starting in state s_i is finite, is equal to one (which is the case for the specific CTMC models in our context of workflow management). Then, the mean first-passage time m_{iA} until the first transition of the CTMC into s_A starting in state s_i can be computed from the system of linear equations

$$-\nu_i m_{iA} + \sum_{j \neq A,\, j \neq i} q_{ij} m_{jA} = -1 \qquad , \; i \neq A$$

where $\nu_i = 1/H_i$ is the rate of leaving state s_i, and $q_{ij} = \nu_i p_{ij}$ is the transition rate from state s_i to state s_j [20]. This linear equation system can be easily solved using standard methods such as the Gauss-Seidel algorithm.

4.2 Load (Service Requests) per Workflow Instance

The execution of a workflow instance spawns a set of activities, which in turn generate *service requests* to different server types. For example, the invocation of an activity incurs a certain initialization and termination load, and a processing load is induced during the entire activity, on involved workflow engine and application server type, and also on the communication server type. Let the matrix $L^t = \left(L_{xa}{}^t\right)$ denote the number of service requests generated on server type x by executing a single instance of the activity type a within an instance of workflow type t.

Consider the *EP* workflow type of our running e-commerce example. The corresponding CTMC has eight states in total, but the absorbing state s_A does not invoke an activity and thus does not incur any load. With three server types, the state-specific load vectors have three components each, and the entire load matrix L^{EP} is a 3×7 matrix.

In practice, the entries of the load matrix have to be determined by collecting appropriate runtime statistics.

4.2.1 Computing the load of a top-level workflow instance without subworkflows

To calculate the load that one workflow instance of a given type generates on the various server types, we use methods for the transient analysis of CTMC models. We will first disregard the possible existence of subworkflows, and will later augment our method to incorporate subworkflows.

The first, preparatory step is to eliminate the potential difficulty that the state residence times are non-uniform; there are standard techniques for transforming the CTMC into a uniform CTMC where the mean residence time is identical for all states and whose stochastic behavior is equivalent to that of the original model [20]. This is more of a technicality and serves to simplify the formulas. The actual analysis is based on a Markov reward model (MRM) which can be interpreted as follows: each time a state is entered and we spend some time there, we obtain a reward, and we are interested in the totally accumulated reward until we enter the absorbing state. This metric is known as the expected reward earned until absorption [20]. Here the reward that we obtain in each state is the number of service requests that are

generated upon each visit of a state. (The term "reward" is somewhat misleading in our problem context, but it is the standard term for such models.) The expected number of service requests that an instance of the workflow type *t* generates on server type *x* can be computed by the formula

$$r_{x,t} = L^t_{x0} + \frac{1}{\nu^t}\left(\sum_{a \neq A} \sum_{z=0}^{\infty} \overline{p}^t_{0a}(z) \sum_{b \neq A, b \neq a} q^t_{ab} L^t_{xb} \right),$$

where $\nu^t = \max_{a \neq A} \left\{ \nu^t_a = 1/H^t_a \right\}$ is the maximum of the departure rates of the CTMC states with H^t_a denoting the state residence time of state s_a, $q^t_{ab} = \nu^t_a p^t_{ab}$ is the transition rate from state s_a to state s_b, and $\overline{p}^t_{0a}(z)$ is the taboo probability that the process will be in state s_a after *z* steps without having visited the absorbing state s_A starting in the initial state s_0. The taboo probabilities can be recursively computed from the Chapman-Kolmogorov equations

$$\overline{p}^t_{ab}(z) = \sum_{c \neq A} \overline{p}^t_{cb} \overline{p}^t_{ac}(z-1), \qquad a, b \neq A$$

starting with $\overline{p}^t_{00}(0) = 1$, and $\overline{p}^t_{0a}(0) = 0$ for $a \neq 0$, and with

$$\overline{p}^t_{ab} = \begin{cases} \dfrac{\nu^t_a}{\nu^t} p^t_{ab}, & b \neq a, \ a \neq A \\ 1 - \dfrac{\nu^t_a}{\nu^t}, & b = a, \ a \neq A \end{cases}$$

denoting the one-step transition probabilities of the CTMC after uniformization [20].

For an efficient approximation of $r_{x,t}$, the summation over the number of steps, *z*, has to be terminated when *z* exceeds a predefined upper bound $z_{\max}$. The value of $z_{\max}$ is set to the number of state transitions that will not be exceeded by the workflow within its expected runtime with very high probability, say 99 percent. This value of $z_{\max}$ can be easily determined during the analysis of the CTMC.

4.2.2 Incorporation of subworkflows

Once we add subworkflows, the expected turnaround time and the expected number of service requests generated by an instance of a workflow type *t* can be calculated in a hierarchical manner. For a CTMC state that represents a subworkflow or a set of parallel subworkflows, H^t_s corresponds to the mean turnaround time and the entries $L_{xs}{}^t$ of the load matrix L^t correspond to the expected number of service requests for the entire set of nested subworkflows. Thus, the mean residence time H^t_s is approximately the maximum of the mean turnaround times of the parallel subworkflows $H^t_s = \max_{b \in S}\{R_b\}$, and the number of service requests $L_{xs}{}^t$ for server type *x* equals the sum of the expected number of service requests generated by the parallel subworkflows $L^t_{xs} = \sum_{b \in S} r_{x,b}$.

Note that the maximum of the mean turnaround times of the parallel subworkflows is actually a lower bound of the mean residence time of the corresponding higher-level state. So the approximation is conservative with regard to the induced load per time unit.

4.3 Total Server Load and Maximum Sustainable Throughput

We associate with each workflow type *t* an arrival rate ξ_t which denotes the mean number of user-initiated workflows (of the given type) per time unit; the actual arrival process would typically be described as a Poisson process for systems with a relatively large number of independent clients (e.g., within an insurance company). By Little's law, the mean number of concurrently executing instances N^t_{active} of workflow type *t* is given by the product of the arrival rate ξ_t of new instances and the mean turnaround time R_t of a single instance of workflow type *t*: $N^t_{active} = \xi_t R_t$.

The server-type-specific request arrival rate $l_{x,t}$ of a single instance of workflow type *t* is given by dividing the expected number of service requests to server type *x*, $r_{x,t}$, by the mean runtime R_t of an instance of *t*. Then the server-type-specific total load, i.e., its request arrival rate over all concurrently active instances of workflow type *t* is the product of $l_{x,t}$ with the mean number, N^t_{active} , of active workflow instances.

Finally, the request arrival rate l_x to server type *x* over all workflow types is obtained by

$$l_x = \sum_t N^t_{active} \frac{r_{x,t}}{R_t} = \sum_t \xi_t r_{x,t} \ .$$

With Y_x servers of server type x and uniform load distribution across these replicas, the total load per server (of type *x*) is $\tilde{l}_x = l_x / Y_x$. Note that this is actually the service request arrival rate. If we assume that each request keeps the server busy for a service time of length b_x on average, then the server's actual throughput is the maximum value $\hat{l}_x \le \tilde{l}_x$ such that $\hat{l}_x b_x \le 1$; for arrival rates such that $\tilde{l}_x b_x > 1$ the server could not sustain the load. So the maximum sustainable throughput (in terms of processed workflow instances per time unit) is given by the minimum of the $\hat{l}_x$ values over all server types *x* (i.e., the server type that saturates first).

4.4 Waiting Time of Service Requests

For analyzing the mean waiting time of service requests, we model each server type *x* as a set of Y_x M/G/1 queueing systems where Y_x is the number of server replicas of the server type. So we assume that service requests are, on average, uniformly distributed across all servers of the same type. This can be achieved by assigning work to servers in a round-robin or random (typically hashing-based) manner. In practice these assignments would typically be performed when a workflow instance starts, so that all subworkflows, activities, or invoked applications of the same type within that workflow instance are assigned to the same server instance for locality. While this realistic load partitioning policy may create temporary load bursts, the long-term (steady-state) load would be spread uniformly.

Each server is modeled only very coarsely by considering only its mean service time per service request and the second moment of this metric. Both of these server-type-specific values can be easily estimated by collecting and evaluating online statistics. We do not model the details of a server's hardware configuration such as CPU speed, memory size, or number of disks. Rather we assume that each server is a well-configured building block, and that we scale up the system by adding such building blocks. In particular, the CPU and disk I/O power of a server are assumed to be in balance, so that neither of these resources becomes a bottleneck while the other is way underutilized. Commercially available commodity servers for informa-

tion systems in general are configured this way, and workflow management would fall into this category. Nevertheless, even if it turns out that one resource type always tends to be the bottleneck, our coarse-grained model is applicable under the assumption that the abstract notion of service time refers to the bottleneck resource within a server.

Let l_x be the arrival rate of service requests at server type *x* as derived in the previous subsection, b_x the mean service time of service requests at server type *x*, and $b_x^{(2)}$ the second moment of the service time distribution of service requests at server type *x*. The mean arrival rate of service requests at a single server of server type *x* is given by $\tilde{l}_x = l_x / Y_x$ where Y_x is the number of servers of server type *x*. Then the mean waiting time w_x of service requests at an individual M/G/1 server of type *x* is given by the standard formula [17]:

$$w_x = \frac{\tilde{l}_x b_x^{(2)}}{2(1-\rho_x)}$$

where $\rho_x = \tilde{l}_x b_x$ is the utilization of the server. This mean waiting time is our main indicator of the responsiveness of the WFMS whenever user interactions take place.

The generalized case for configurations where multiple server types, say *x* and *z*, are assigned to the same computer is handled as follows: the server-type-specific arrival rates $\tilde{l}_x$ and $\tilde{l}_z$ are summed up, the server types' common service time distribution is computed, and these aggregate measures are fed into the M/G/1 model to derive the mean waiting time common to all server types on the same computer.

Note that our approach is so far limited to a homogenous setting where all underlying computers have the same performance capacity, but could be extended to the heterogeneous case by adjusting the service times on a per computer basis.

5 Availability Model

In this section we present the availability model for a distributed WFMS according to our architectural model of Section 2. We analyze the influence of transient component failures on the availability of the entire system.

5.1 CTMC for System States

Following the standard approach, our availability model is again based on continuous-time Markov chains (CTMC). The steady-state analysis of the CTMC delivers information about the probability of the current system state of the WFMS. Each state of the CTMC represents a possible system state of the WFMS. So a state of the CTMC is a *k*-tuple with *k* being the number of different server types within the WFMS, and each entry of the tuple represents the number X_x of *currently available* servers of server type *x* at a given point of time. For example, the system state *(2,1,1)* means that the WFMS consists of three different server types and there are *2* servers of type *1*, *1* server of type *2*, and *1* server of type *3* currently running while the others have failed and are being restarted or have been taken down for maintenance. When a server of type *x* fails, the CTMC performs a transition to the system state with the corresponding value for server type *x* decreased by one. For example, the system state $(X_1,...,X_x,...,X_k)$ is left when a server of type *x* fails, and the system state $(X_1,...,(X_x-1),...,X_k)$ is entered. Analogously, when a server of type *x* completes its restart, the CTMC performs a transition into the state where the value for server type *x* is increased by one. The failure rates λ_x and the repair rates μ_x of the server types are the corresponding transition rates of the CTMC. This basic model implicitly assumes that the time spent in a state is exponentially distributed, but note that non-exponential failure or repair rates (e.g., anticipated periodic downtimes for soft-

ware maintenance) can be accommodated as well, by refining the corresponding state into a (reasonably small) set of exponential states [20]. This kind of expansion can be done automatically once the distributions of the non-exponential states are specified.

With such a CTMC at hand, which can be shown to be ergodic, we are able to compute the steady-state probability for each state of the CTMC. Then the probability for the entire system being unavailable is simply the sum of the state probabilities over all those states where at least one server type is completely unavailable (i.e., has a zero entry in the *X* vector). We next present the details of the steady-state analysis of the CTMC in the following subsection.

5.2 Steady-state Analysis of the Availability-Model CTMC

Let *k* be the number of different server types, $Y = (Y_1,\ldots,Y_k)$ the WFMS configuration, and let $X = \{\ (X_1,\ldots,X_k) \mid\ 0 \le X_x \le Y_x\ ,\ 1 \le x \le k\ \}$ be the finite set of the system states of the WFMS.

We encode *X* into a set $\tilde{X}$ of integer values that denote the states of the CTMC of the previous subsection as follows:

$$(X_1,\ldots,X_k) \mapsto \sum_{j=1}^{k} X_j \prod_{l=1}^{j-1}(Y_l+1).$$

For example, for a CTMC with three server types, two servers each we encode the states (0,0,0), (1,0,0), (2,0,0), (0,1,0) etc. as integers 0, 1, 2, 3, and so on.

To derive the steady-state probabilities of the CTMC, we have to solve a system of linear equations

$$\pi Q = 0$$
$$\sum_i \pi_i = 1$$

where π_i denotes the steady-state probability of state $i \in \tilde{X}$, π is the vector (π_i), and *Q* is the infinitesimal generator matrix of the CTMC [19]. The generator matrix $Q = (q_{ij})$ with $i, j \in \tilde{X}$ is obtained by setting q_{ij} to the transition rate from the state $(X_1,\ldots,X_k)$ corresponding to $i \in \tilde{X}$ into the state $(X'_1,\ldots,X'_k)$ corresponding to $j \in \tilde{X}$. The diagonal elements of *Q* are set to $q_{ii} = -\sum_{j \ne i} q_{ij}$. Note that $-q_{ii}$ is the rate at which the system departs from state $i \in \tilde{X}$ [19]. The resulting linear equation system can again be solved easily by using standard methods such as the Gauss-Seidel algorithm.

As an illustrating example consider a scenario with three server types. Let server type *1* be the communication server, server type *2* be one type of workflow engine, and server type *3* be one type of application server. The failure rates are assumed as follows (where failures would typically be software-induced "Heisenbugs" [8] or downtimes for maintenance): one failure per month (so $\lambda_1 = (43200\ \text{min})^{-1}$) for a communication server, one failure per week (so $\lambda_2 = (10080\ \text{min})^{-1}$) for a workflow engine, and one failure per day (so $\lambda_3 = (1440\ \text{min})^{-1}$) for an application server.

We further assume that the mean time to repair of a failed server is 10 minutes regardless of the server type, so the repair rates are $\mu_1 = \mu_2 = \mu_3 = (10\ \text{min})^{-1}$. Note that these absolute figures are arbitrary, but the ranking of server types with respect to failure rates may reflect the maturity of the underlying software technologies. The entire WFMS is available when at least one server of each server type is running, and the WFMS is down when all server replications of at least one server type are down.

The CTMC analysis computes an expected downtime of 71 hours per year if there is only one server of each server type, i.e., no server is replicated. By 3-way replication of each server type, the system downtime can be brought down to 10 seconds per year. However, replicating the most unreliable server type, i.e., the application server type in our example, three times and having two replicas of each of the other two server types is already sufficient to bound the unavailability by less then a minute.

6 Performability Model

With the performance model alone we are able to predict the performance of the WFMS for a single, given system state. So changes of the system state over time caused by failures and repairs are not captured. In this section, we present a performability model that allows us to predict the performance of the WFMS with the effects of temporarily non-available servers (i.e., the resulting performance degradation) taken into account.

Our performability model is a hierarchical model constituted by a Markov reward model (MRM) for the availability CTMC of Section 5, where the state-specific rewards are derived from the performance model of Section 4. The probability of being in a specific system state of the WFMS is inferred from the availability model. As the reward for a given state of the availability CTMC, we use the mean waiting time of service requests of the WFMS in that system state. So we need to evaluate the performance model for each considered system state, rather than only for the overall configuration which now is merely the "upper bound" for the system states of interest. Then the steady-state analysis of the MRM [19] yields the expected value for the waiting time of service requests for a given WFMS configuration with temporary performance degradation induced by failures.

Let Y be a given system configuration and π_i be the steady-state probability for the system state $i \in \tilde{\tilde{X}}$ as calculated in Section 5. Let $w^i = \left(w_x^i\right)$ be the vector of the expected waiting times of service requests of all server types x for a given system state $i \in \tilde{X}$ as calculated in Section 4. Then the performability vector of the expected values of the waiting times of service requests $W^Y = \left(W_x^Y\right)$ for server types x under configuration Y and with failures taken into account, is obtained by conditioning the system-state-specific waiting time vectors $w^{\acute{I}}$ with the system state probabilities π_i, thus yielding $W^Y = \sum_{i \in \tilde{X}} w^i \pi_i$

The value of W^Y derived this way is the ultimate metric for assessing the performance of a WFMS, including the temporary degradation caused by failures and downtimes of server replicas. The system's responsiveness is acceptable if no entry of the waiting-time vector W^Y is above a critical tolerance threshold.

7 Configuration Tool

In this section, we sketch a configuration tool that we are currently developing based on the presented analytic models. We also discuss the integration of the tool into a given workflow environment.

7.1 Functionality and Architecture of the Configuration Tool

The configuration tool consists of four components: the *mapping* of workflow specifications onto the tool's internal models, the *calibration* of the internal models by means of statistics from monitoring the system, the *evaluation* of the models for given input parameters, and the computation of *recommendations* to system administrators and architects, with regard to specified performability goals.

For the mapping the tool interacts with a workflow repository where the specifications of the various workflow types are stored. In addition, statistics from online monitoring may be used as a second source (e.g., to estimate transition probabilities etc.). The configuration tool translates the workflow specifications into the corresponding CTMC models. For the evaluation of the models, additional parameters may have to be calibrated; for example, the first two moments of the server-type-specific service times have to be fed into the models. This calibration is again based on appropriate online monitoring. So both the mapping and calibration components require online statistics about the running system. Consequently, when the tool is to be used for configuring a completely new workflow environment, many parameters have to be intellectually estimated by a human expert. Later, after the system has been operational for a while, these parameters can be automatically adjusted, and the tool can then make appropriate recommendations for reconfiguring the system.

The evaluation of the tool's internal CTMC models is driven by specified performability goals. System administrators or architects can specify goals of the following two kinds: 1) a tolerance threshold for the mean waiting time of service requests that would still be acceptable to the end-users, and 2) a tolerance threshold for the unavailability of the entire WFMS, or in other words, a minimum availability level.

The first goal requires evaluating the performability model, whereas the second one merely needs the availability model. The tool can invoke these evaluations either for a given system configuration (or even a given system state if failures are not a major concern), or it can search for the minimum-cost configuration that satisfies both goals, which will be discussed in more detail in the next subsection. The cost of a configuration is assumed to be proportional to the total number of servers that constitute the entire WFMS, but this could be further refined with respect to different server types. Also, both kinds of goals can be refined into workflow-type-specific goals, by requiring, for example, different maximum waiting times or availability levels for specific server types.

The tool uses the results of the model evaluations to generate recommendations to the system administrators or architects. Such recommendations may be asked for regarding specific aspects only (e.g., focusing on performance and disregarding availability), and they can take into account specific constraints such as limiting or fixing the degree of replication of particular server types (e.g., for cost reasons).

So, to summarize, the functionality of the configuration tool comprises an entire spectrum ranging from the mere analysis and assessment of an operational system all the way to providing assistance in designing a reasonable initial system configuration, and, as the ultimate step, automatically recommending a reconfiguration of a running WFMS.

7.2 Greedy Heuristics Towards a Minimum-cost Configuration

The most far-reaching use of the configuration tool is to ask it for the minimum-cost configuration that meets specified performability and availability goals. Computing this configuration requires searching the space of possible configurations, and evaluating the tool's internal models for each candidate configuration. While this may eventually entail full-fledged algorithms for mathematical optimization such as branch-and-bound or simulated annealing, our first version of the tool uses a simple greedy heuristics.

The greedy algorithm iterates over candidate configurations by increasing the number of replicas of the most critical server type until both the performability and the availability goals are satisfied. Since either of the two criteria may be the critical one and because an additional server replica improves both metrics at the same time, the two criteria are considered in an interleaved manner. Thus, each iteration of the loop over candidate configurations evaluates the performability and the availability, but adds servers to two different server types only after re-evaluating whether the goals are still not met. This way the algorithm avoids "oversizing" the system configuration.

8 Conclusion

In this paper, we have developed models to derive quantitative information about the performance, availability, and performability of configurations for a distributed WFMS. These models form the core towards an assessment and configuration tool. As an initial step towards evaluating the viability of our approach, we have defined a WFMS benchmark [7], and we are conducting measurements of various products and prototypes, including our own prototype coined Mentor-lite, under different configurations. These measurements are a first touchstone for the accuracy of our models. In addition, we have started implementing the configuration tool sketched in Section 7. This tool will be largely independent of a specific WFMS, using product-specific stubs for the tool's monitoring, calibration, and recommendation components. We expect to have the tool ready for demonstration by the middle of this year.

References

[1] G. Alonso, D. Agrawal, A. El Abbadi, C. Mohan, Functionality and Limitations of Current Workflow Management Systems, IEEE Expert Vol.12 No. 5, 1997

[2] T. Bauer, P. Dadam, A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration, IFCIS Conf. on Cooperative Information Systems (CoopIS), Charleston, South Carolina, 1997

[3] T Bauer, P. Dadam, Distribution Models for Workflow Management Systems - Classification and Simulation (in German), Technical Report, University of Ulm, Germany, 1999

[4] A. Cichocki, A. Helal, M Rusinkiewicz, D. Woelk, Workflow and Process Automation, Kluwer Academic Publishers, 1998

[5] A. Dogac, L. Kalinichenko, M. Tamer Ozsu, A. Sheth (Eds.), Workflow Management Systems and Interoperability, NATO Advanced Study Institute, Springer-Verlag, 1998

[6] D. Georgakopoulos, M. Hornick, A. Sheth, An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure, Distributed and Parallel Databases Vol. 3 No. 2, 1995

[7] M. Gillmann, P. Muth, G. Weikum, J. Weissenfels, Benchmarking of Workflow Management Systems (in German), German Conf. on Database Systems in Office, Engineering, and Scientific Applications, Freiburg, Germany, 1999

[8] J. Gray, A. Reuter, Transaction Processing – Concepts and Techniques, Morgan Kaufmann, 1993

[9] C. Hagen, G. Alonso, Backup and Process Migration Mechanisms in Process Support Systems, Technical Report, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland, 1998

[10] D. Harel, State Charts: A Visual Formalism for Complex Systems, Science of Computer Programming Vol. 8, 1987

[11] D. Harel, E. Gery, Executable Object Modeling with Statecharts, IEEE Computer Vol.30 No.7, 1997

[12] S. Jablonski, C. Bussler, Workflow Management, Modeling Concepts, Architecture, and Implementation, International Thomson Computer Press, 1996

[13] J. Klingemann, J. Waesch, K. Aberer, Deriving Service Models in Cross-Organizational Workflows, Int'l Workshop on Reasearch Issues in Data Engineering (RIDE), Sydney, Australia, 1999

[14] M. Kamath, G. Alonso, R. Günthör, C. Mohan, Providing High Availability in Very Large Workflow Management Systems, Int'l Conf. on Extending Database Technology (EDBT), Avignon, France, 1996

[15] C. Mohan, Workflow Management in the Internet Age, Tutorial, http://www-rodin.inria.fr/~mohan

[16] P. Muth, D. Wodtke, J. Weissenfels, G. Weikum, A. Kotz Dittrich, Enterprise-wide Workflow Management based on State and Activity Charts, in [5]

[17] R. Nelson, Probability, Stochastic Processes, and Queueing Theory, Springer-Verlag, 1995

[18] H. Schuster, J. Neeb, R. Schamburger, A Configuration Management Approach for Large Workflow Management Systems, Int'l Joint Conf. on Work Activities Coordination and Collaboration (WACC), San Francisco, California, 1999

[19] R. A. Sahner, K. S. Trivedi, A. Puliafito, Performance and Reliability Analysis of Computer Systems, Kluwer Academic Publishers, 1996

[20] H.C. Tijms, Stochastic Models, John Wiley and Sons, 1994

[21] Transaction Processing Performance Council, http://www.tpc.org/

[22] Unified Modeling Language (UML) Version 1.1, http://www.rational.com/uml/

[23] Workflow Management Coalition, http://www.wfmc.org/

[24] D. Wodtke, G. Weikum, A Formal Foundation For Distributed Workflow Execution Based on State Charts, Int'l Conf. on Database Theory (ICDT), Delphi, Greece, 1997