Workflow Management with Service Quality Guarantees

Michael Gillmann
infor: business solutions AG
michael.gillmann@infor.de

Gerhard Weikum
University of the Saarland
weikum@cs.uni-sb.de

Wolfgang Wonner
University of the Saarland
wonner@cs.uni-sb.de

ABSTRACT

Workflow management systems (WFMS) that are geared for the orchestration of business processes across multiple organizations are complex distributed systems: they consist of multiple workflow engines, application servers, and communication middleware servers such as ORBs, where each of these server types can be replicated on multiple computers for scalability and availability.

Finding an appropriate system configuration with guaranteed application-specific quality of service in terms of throughput, response time, and tolerable downtime is a major challenge for human system administrators. This paper presents a tool that largely automates the task of configuring a distributed WFMS. Based on a suite of mathematical models, the tool derives the necessary degrees of replication for the various server types in order to meet specified goals for performance and availability as well as "performability" when service is degraded due to outages of individual servers. The paper describes the configuration tool, with emphasis on how to capture the load behavior of workflows in a realistic manner. We also present extensive experiments that evaluate the accuracy of the tool's underlying models and demonstrate the practical feasibility of automating the task of configuring a distributed WFMS. The experiments use a detailed simulation which in turn has been validated through measurements with the Mentor-lite prototype system.

1 INTRODUCTION

1.1 Motivation

Next-generation e-services such as advanced forms of electronic sales, auctions, and brokerage will achieve their mission only if they provide guaranteed "quality of service" (QoS). This goal encompasses both high availability and user-acceptable response time, and ideally even the combined notion of "performability" which considers the impact of transient outages on throughput and response time [11].

Today's Web-based e-services are still far from providing QoS guarantees: they frequently exhibit inconvenient outages, and often have absolutely unacceptable responsiveness during popular business hours (i.e., when load surges occur). The bottlenecks include both networking and server-side issues [17], but most often it is congested application and data servers at the e-business site that cause performance problems. The reason is that these servers are poorly configured, tuned, and administered. Obviously, there are exceptions to this largely poor situation, but these are exactly those sites that heavily invest into their human support staff for proper configuration and tuning. However, the scarceness and high cost of these human experts, on one hand, and the high dynamics of e-business processes and customer demands, on the other hand, mandate an automated approach to the problem.

Advanced e-services, especially those in the B2B (business-to-business) category such as supply chains, have rich application logic and need to support long-lived business processes [5]. The appropriate technology for this purpose, within a standard three-tier architecture, is workflow management, either in explicit form by setting up workflow engine as part of the site's middle-tier application server (see, e.g., the WISE project [15]) or implicitly by orchestrating application server scripts (e.g., ASP or PHP scripts) into business processes. Workflow engines have become relatively mature products (e.g., MQ Workflow, Staffware, Biztalk, E-Speak, etc.) that can interact with a Web server as the application frontend as well as backend database servers. Furthermore and most importantly, by replicating a workflow server across multiple computers it is possible to scale up throughput, improve responsiveness, and enhance availability in the presence of individual server failures. Unlike the replication of transactional data, the replication of workflow servers does not pose any severe scalability problems. In contrast to simple, stateless Web applications, the workflow engines need to maintain state information for ongoing workflow instances, but this can be easily delegated to (a cluster of) database servers.

So the replication of workflow servers in a distributed system is the mechanism toward better quality of service, and the same argument holds also for servers that handle invoked applications that are spawned as activities of a workflow. The critical issue, however, is how to devise a strategy for the degree of replication and how to configure the entire system such that it can *guarantee* the application's QoS specification. The problem is made more difficult by the fact that business processes may span different enterprises or autonomous units within an organization, so the complete system consists of different types of workflow engines dedicated to handle specific subworkflows. The key question to be answered is how many replicas we need for which kind of servers. This is the issue that we address in this paper.

1.2 Contribution

In this paper we present a fully implemented tool for the automatic configuration of a distributed workflow system in order to meet specified goals for throughput, response time, availability, and performability. The workflow system that we have primarily considered, is our own prototype system Mentor-lite [19, 24], which is accessible from Web application servers via XML messages. Our tool can be easily adapted to other workflow systems; Mentor-lite serves as an example platform over which we have full control for instrumentation and experimental studies. The configuration tool, coined Goliat (for Goal-driven auto-configuration tool), automatically derives suitable degrees of replication for workflow servers and application servers of different types as well as request brokering and communication servers (e.g., Corba ORBs) so that the entire system can guarantee the quality of service that is requested for the e-business application under consideration.

The Goliat auto-configuration tool is driven by the specifications of the business processes, which are statecharts in our specific Mentor-lite environment, and statistics about the execution paths of these workflows. Its core asset is a suite of analytic models, using stochastic methods like continuous-time Markov chains (CTMC) and Markov reward models, to predict the performance, availability, and performability of a given system configuration under a given load. The performance model estimates the maximum sustainable throughput in terms of workflow instances per time unit and the mean waiting time for service requests such as interactions upon starting an activity. The availability model estimates the mean downtime of the entire system for given failure and restart rates of the various components. Finally, the performability model takes into account the performance degradation during transient failures, and estimates the effective mean waiting time for service requests with explicit consideration of periods during which only a subset of a server type's replicas are running. These models, which form the mathematical underpinnings of the Goliat tool, have mostly been derived in our earlier work published in [8].

The current paper extends our earlier work in two major ways:

1) We have refined and improved the load and performance model of Goliat so that it can capture the behavior of workflow executions in a more realistic manner. In particular, we have extended the model so that it can cope with arbitrary subworkflows and loops. Also, we can now support constructs for which exponential distributions, the standard assumption in Markov models, are inadequate, namely, component downtimes and activity turnaround times. We eliminate this restriction by the technique of approximating generalized Erlang distributions with appropriately designed Markov sub-models [22].
2) The Goliat tool has now been fully implemented, validated, and evaluated. To this end, we have carried out extensive experiments with the Mentor-lite environment. For better reproducibility and efficiency of systematic studies we have developed a simulator for Mentor-lite, which actually executes the Mentor-lite code but has parameterized functions for the usage of (virtual, i.e., simulated) resources. The simulator has been validated by measurements of Mentor-lite, and its parameters have been calibrated by these measurements. The evaluation of the Goliat tool against the simulations and measurements shows that its analytic predictions are sufficiently accurate to be practically viable.

1.3 Related work

Although the literature includes much work on scalable WFMS architectures (e.g., [1, 6, 16]), 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. [20] 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 capture the dynamic behavior of workflow instances and the resulting performance. In addition, we address also the availability and performability dimensions.

1.4 Outline

The rest of the paper is organized as follows. Section 2 presents the architecture of the Goliat configuration tool. Section 3 reviews the suite of stochastic models that we developed in [8]. Section 4 presents new extensions of the stochastic performance model to capture the behavior of workflow executions in a more realistic manner. Section 5 describes the experimental testbed for the validation and evaluation of the developed auto-configuration method. Section 6 presents the results of the performance and performability experiments.

2 ARCHITECTURE OF THE GOLIAT CONFIGURATION TOOL

A distributed configuration of Mentor-lite consists of different workflow servers (i.e., instances of the workflow engine) and application servers, and one communication server (i.e., ORB). Each server of the first two categories can be dedicated to a specified set of workflow activities or invoked applications on a per type basis. Each of these dedicated servers and also the communication server can be replicated across multiple computers for enhanced performance and availability. Given this flexibility (which is supported in similar ways also by some commercial WFMSs), it is a difficult problem to choose an appropriate configuration for the entire WFMS that meets all requirements with regard to throughput, interaction response time, and availability. Moreover, it may be necessary to adapt an initial configuration over time due to changes of the workflow load, e.g., upon adding new workflow types.

To solve this problem, we have developed the Goliat auto-configuration tool based on a suite of analytic models, using stochastic methods like continuous-time Markov chains and Markov reward models, to predict the performance, availability, and performability for given configuration and workload. These models have been developed in [8]; we will outline the general approach in Section 3 and will present new extensions in Section 4. The Goliat tool is driven by the workflow specifications that it obtains from the repository and by statistics on the workload from the monitoring tool of Mentor-lite. Goliat feeds this information into its analytic models for a what-if analysis of a hypothetical configuration. By systematic variation of the parameters for different candidate configurations the tool is also able to derive the (analytically) best configuration, i.e., the minimum degree of replication of each of the involved server types to meet given performance, availability, and performability goals. The Goliat tool is largely independent of a specific WFMS, by using specific, easily replacable, stubs for its interactions with the WFMS.

The components of the Goliat tool and its embedding into the overall system environment are illustrated in Figure 1. Goliat consists of four main 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 regarding specified goals.

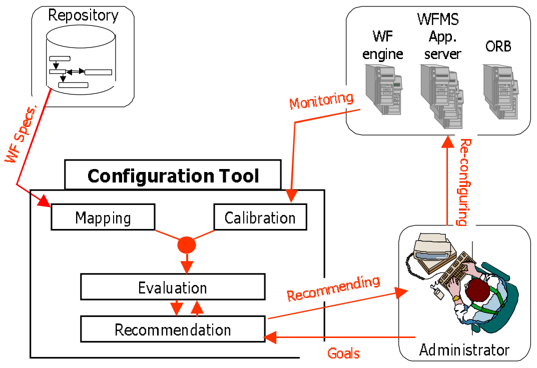
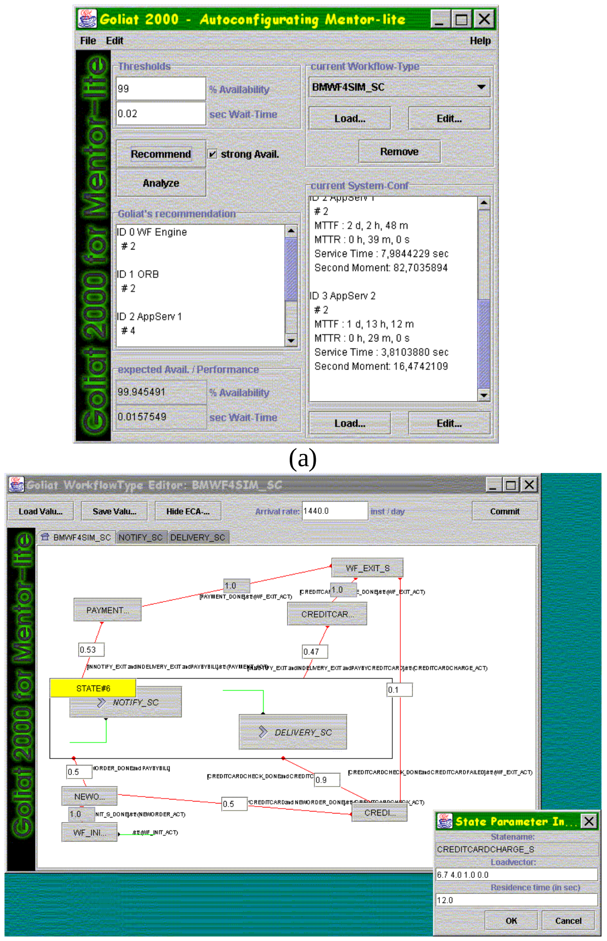


Figure 1 : Integration of the auto-configuration tool

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 are used as a second source (e.g., to estimate typical control flow behavior etc.). The configuration tool translates the workflow specifications into corresponding continuous-time Markov chain models. For the evaluation of the models, additional parameters may have to be calibrated; for example, the first two moments of server-type-specific service times for various elementary service requests (e.g., starting an activity) have to be fed into the models. This calibration is again based on appropriate online monitoring. So both the mapping and calibration components exploit 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.
Goliat uses the results of the model evaluations to generate recommendations to system administrators or application 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). The more far-reaching use of the Goliat tool is to ask it for the minimum-cost configuration that meets specified performability (i.e., throughput and response time) and availability (i.e., downtime) goals. Computing the best configuration requires searching the space of possible configurations and evaluating the tool's internal models for each candidate configuration. In our current implementation we use a simple greedy heuristics for this purpose, thus possibly arriving at a suboptimal, but sufficiently good, result. The 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 [8].



Thresholds
99 % Availability
0.02 sec Wait-Time
✓ strong Avail.

Goliat's recommendation
ID 0 WF Engine
 # 2
ID 1 ORB
 # 2
ID 2 AppServ 1
 # 4

expected Avail. / Performance
99.945491 % Availability
0.0157549 sec Wait-Time

current Workflow-Type
BMWF4SIM_SC

current System-Conf
ID 2 AppServ 1
 # 2
 MTTF : 2 d, 2 h, 48 m
 MTTR : 0 h, 39 m, 0 s
 Service Time : 7,9844229 sec
 Second Moment: 82,7035894
ID 3 AppServ 2
 # 2
 MTTF : 1 d, 13 h, 12 m
 MTTR : 0 h, 29 m, 0 s
 Service Time : 3,8103880 sec
 Second Moment: 16,4742109

Arrival rate: 1440.0 inst / day

Statename:
CREDITCARDCHARGE_S
Loadvector:
6.7 4.0 1.0 0.0
Residence time (in sec)
12.0

Figure 2 : User interfaces of the configuration tool

Figure 2 gives an impression of the user interface of Goliat for the interactive case that either no run-time statistics are available yet or the system administrator wants to carry out certain sensitivity studies in an explorative manner. The main control panel shown in Figure 2(a) allows the user to explore a workflow type (i.e., a statechart file from which the specification is loaded) and a system configuration as the basic input for the calculations. The parameters for the quantitative behavior of the workflow can be edited in the workflow editor shown in Figure 2(b). States are represented by buttons and transitions as arrows with a small circle denoting the target state of the transiton. The user can switch between the top level workflow and its subworkflows by clicking on the corresponding state button. Nested states are marked with an arrow. The small textfields at the transitions serve to specify transition probabilities. Clicking on a state that does not represent a subworkflow opens a dialog in which the state parameters can be edited. These parameters are 1) the state-specific load vector (i.e., the number of service requests that this state generates on the different server types), and 2) the state's mean residence time, which is the time that is spent in this state (i.e., the turnaround time of the corresponding activity) before entering the next state. In addition to these state-specific load parameters, the user must provide the arrival rate of workflows for the given workflow type (i.e., the number of workflows initiated per time unit).

The system configuration can also be loaded from a file or edited. The parameters that can be set for each server type are: 1) the name of the server type, 2) the number of replications of the server type, 3) the server type's mean time to failure (MTTF), 4) the server type's mean time to repair (MTTR), 5) the mean service time for a single request sent to the server, and 6) the second moment of the service time distribution. The user can also add or remove new server types.

On the left hand side of the control panel, the user can specify acceptable thresholds for the system availability and the mean waiting time of the service requests in the WFMS. In the analyis mode, where one configuration is assessed, Goliat produces the following output:

- the total load in average that is induced on the different server types in terms of the number of requests for the execution of a single workflow instance,
- the maximum sustainable throughput in terms of service requests per time that each server can process,
- the expected mean waiting time at each server, which is the main indicator for deciding whether the system’s performance is user-acceptable or not,
- the mean turnaround time of the specified workflow type based on the given residence times and transition probabilities of the workflow type,
- the expected availability of the system (in percent) and its expected downtime in hours, minutes, and seconds per year,
- the performability (i.e., throughput and mean waiting time metrics) of the system with temporary degradation and unavailability of servers taken into account.

3 STOCHASTIC MODELING OF WORKFLOW BEHAVIOR

3.1 Example scenario

As an example we consider a simplified e-commerce scenario, which we suggested as a benchmark in [7]. This electronic purchase workflow is similar to the TPC-C benchmark for transaction systems [23], with the key difference that we combine multiple transaction types into a workflow and further enhance the functionality. Figure 3 shows the workflow specification in the form of a statechart [11]. A statechart is a finite state machine with a distinguished initial state and transitions driven by event-condition-action rules (ECA rules).

Important additional features of state charts are nested states and orthogonal components. Nesting of states means that a state can itself contains an entire statechart. The semantics is that upon entering the higher-level state, the initial state of the embedded lower-level statechart is automatically entered, and upon leaving the higher-level state all embedded lower-level state charts are left. The support for state nesting is especially useful for incorporating subworkflows, for stepwise refinement during the design of business processes, and as an abstraction for service composition. Orthogonal components denote the parallel execution of two or more statecharts that are embedded in the same higher-level state. Such 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(a) shows the top-level statechart for the electronic purchase workflow. Each state corresponds to an activity or a (set of 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. For parallel subworkflows, the final states of the corresponding orthogonal components serve to synchronize the termination (i.e., a join in the control flow).

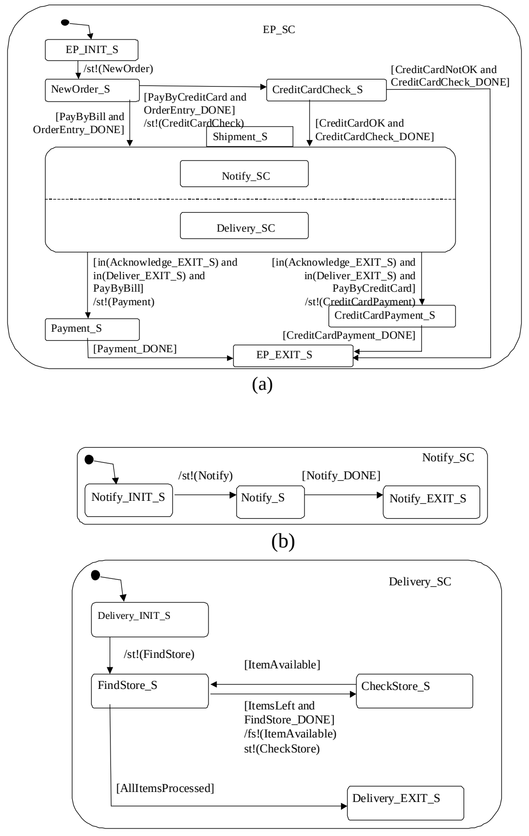

EP_SC
EP_INIT_S
/st!(NewOrder)
NewOrder_S
[PayByCreditCard and OrderEntry_DONE] /st!(CreditCardCheck)
CreditCardCheck_S
[CreditCardNotOK and CreditCardCheck_DONE]
[PayByBill and OrderEntry_DONE]
Shipment_S
[CreditCardOK and CreditCardCheck_DONE]
Notify_SC
Delivery_SC
[in(Acknowledge_EXIT_S) and in(Deliver_EXIT_S) and PayByBill] /st!(Payment)
[in(Acknowledge_EXIT_S) and in(Deliver_EXIT_S) and PayByCreditCard] /st!(CreditCardPayment)
CreditCardPayment_S
Payment_S
[CreditCardPayment_DONE]
[Payment_DONE]
EP_EXIT_S
(a)
Notify_SC
/st!(Notify)
[Notify_DONE]
Notify_INIT_S
Notify_S
Notify_EXIT_S
(b)
Delivery_SC
Delivery_INIT_S
/st!(FindStore)
[ItemAvailable]
FindStore_S
CheckStore_S
[ItemsLeft and FindStore_DONE] /fs!(ItemAvailable) st!(CheckStore)
[AllItemsProcessed]
Delivery_EXIT_S
(c)

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

The workflow behaves as follows. First, the *NewOrder* activity is executed. Upon its completion the control flow is split. If the customer uses a credit card for payment, the condition *PayByCreditCard* is set and the *CreditCardCheck* activity checks the validity of the credit card. If there are problems with the credit card, the workflow is terminated. In the standard case the shipment, represented by the state *Shipment_S*, is initiated spawning two parallel subworkflows which are specified in the statecharts *Notify_SC* (Figure 3(b)) and *Delivery_SC* (Figure 3(c)). The first subworkflow, *Notify*, has only one activity that sends an acknowledgment mail. The second subworkflow, *Delivery*, (sequentially) invokes for each ordered item an activity that identifies a store or warehouse from which the item could be shipped. Then, a second activity instructs the store to deliver the item and waits for an acknowledgement. The two activities *FindStore* and *CheckStore* are repeated within a loop over all ordered items. After the termination of both subworkflows, the parallel branches in the control flow are synchronized, and then split again depending on the mode of payment. The workflow terminates in the final state *EP_EXIT_S*.

3.2 Basic workload model

For predicting the load that is induced by workflow execution on the underlying servers, we have to analyze the control flow behavior of workflow instances. As workflows include conditional branches and loops, the best we can do in this regard is to describe the execution stochastically. Our goal thus is to estimate, for each type of activity, the mean number of activity invocations per workflow instance. We first concentrate on workflows without nesting, and will come back to the general case later by showing how to incorporate subworkflows in the overall model. To describe the control flow behavior we use stochastic processes, more specifically first-order continuous-time Markov chains (CTMC) (see, e.g., the textbook [22]).

3.2.1 The flow process

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 $P = (p_{i,j})$ of one-step 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 possible execution states of a workflow type *t*. The control flow of an instance of *t* is modeled by a CTMC where the states correspond to the workflow execution states s_i. The state transition probability $p_{i,j}$ 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 s_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.

The formal definition of the flow process of a workflow type is as follows:

Definition 1 : (flow process, mean residence time, departure rate, one-step probability)

The *flow process* $F^t(\tau)$ of the workflow type *t* is a stochastic process with the finite set of states $Z^t \cup \{s_A\}$, where Z^t is the set of possible execution states of the workflow (i.e., the states of its statechart) and s_A is an additional absorbing state with incoming transitions from the workflow's final states and no outgoing transition. The transitions between the states of $F^t(\tau)$ achieve the following conditions:

1. When the flow process enters a state $s_i \in Z^t$ the time until the flow process performs the next transition is randomly distributed with mean value H_i. H_i is called the *mean residence time* of the process in state s_i, and $\nu_i = 1/H_i$ is called the *departure rate* of the process from state s_i.

2. The flow process enters the state $s_j \in Z^t \cup \{s_A\}$ when leaving the state s_i with a probability of $p_{i,j}$ with $p_{i,i} = 0$ and $\sum_{s_j \in Z^t \cup \{s_A\}} p_{i,j} = 1$. This probability is independent of the time period that the flow process spent in state s_i and also independent of the process's earlier history before reaching s_i. $p_{i,j}$ is called the *one-step probability* from state s_i to state s_j.

3. The process starts in the initial state s_0 with a probability of 1 and in every other state including s_A with a probability of 0.

With the assumption that the residence times of the execution states $s_i \in Z^t$ are exponentially distributed over all instances of the workflow type *t*, the flow process is a CTMC.

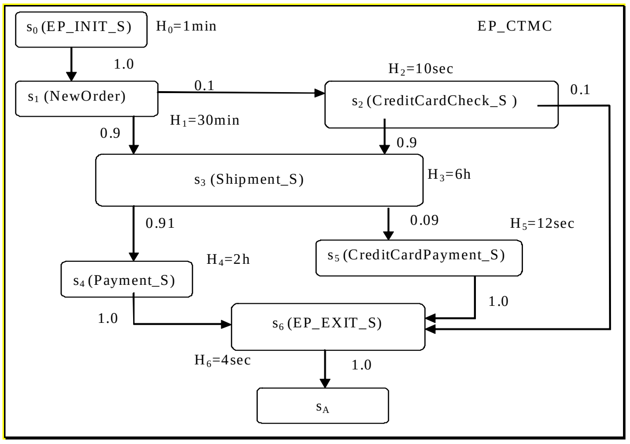


Figure 4 : CTMC representing the electronic purchase workflow type

Figure 4 gives an example for the CTMC representing the electronic purchase workflow type of Figure 3(a). The CTMC consists of seven states (plus the absorbing state s_A), each representing one of the seven possible execution states of the workflow's top-level statechart.

For better readability of Figure 4, the names of the CTMC's states are extended by the names of the corresponding states of the statechart. The values for the one-step probabilities and the mean residence times are fictitious for mere illustration. The shown values would, for example, mean that 10% of all orders are paid by credit card, and 10% of these cases are rejected because the credit card is not valid.

3.2.2 *Mean turnaround time of workflow instances*

We derive the mean turnaround time of a workflow instance of type *t* by the transient analysis of the corresponding CTMC. 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 tim*e of the absorbing state s_A [22]. The first-passage time of a CTMC state is generally computed by solving a set of linear equations that can be easily solved using standard methods such as the Gauss-Seidel algorithm.

3.2.3 *Induced load 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 the underlying workflow engine, application server, and also the communication server. Let the matrix $L = (L_{xa})$ denote the number of service requests generated on server type *x* by executing a single instance of the activity type *a*. Each column in this matrix is an activity-specific load vector.

Consider the *Delivery* workflow type of our running example. Recall that each activity corresponds to exactly one state, and vice versa. Figure 5 shows the corresponding CTMC with the state-specific load vectors for the service requests L_0 of the initial state s_0 and $L_2^{Delivery}$ of state s_2. Here the x^{th} vector component denotes the server-type *x* load L_{x0} and L_{x2}, respectively, assuming three different server types.

The corresponding CTMC has five states in total, but the absorbing state s_A does not invoke an activity and thus does not incur any load anymore. With three server types, the state-specific load vectors have three components each, and the entire load matrix L is a 3×4 matrix that could, for example, look as follows:

$$L^{Delivery} = \begin{pmatrix} 2 & 2 & 5 & 1 \\ 2 & 3 & 7 & 1 \\ 4 & 2 & 0 & 2 \end{pmatrix}.$$

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

Note that our server 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 categories made explicit in our model are 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.

3.2.4 *Nested subworkflows*

For workflow types with subworkflows, each subworkflow is represented as a single state within the CTMC of the parent workflow. In the case of parallelism (i.e., orthogonal components in statechart terminology), the corresponding state represents all parallel subworkflows together. Subworkflows are analyzed first in terms of their load matrix and their turnaround time. To incorporate these results into the corresponding parent workflow, the expected turnaround time of the parent is calculated in the following hierarchical manner. For a CTMC state *s* that represents a subworkflow or a set of parallel subworkflows S, H_S corresponds to the mean turnaround time for the entire set of nested subworkflows. Thus, the mean residence time H_S is set to the maximum of the mean turnaround times of the parallel subworkflows $H_S = \max_{t \in S} \{R_t\}$. 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.

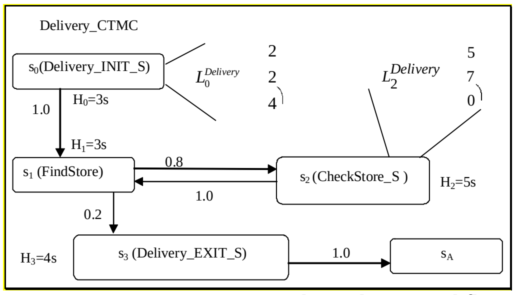


Figure 5 : CTMC representing the *Delivery* workflow type

3.3 Performance model

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 [22, 21] where Y_x is the number of server replicas of the server type in the system configuration. 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 information 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.

3.4 Performability model regarding transient component outages

The performability model 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) [21]. A MRM consists of a continuous time Markov chain and a reward function. The reward function assignes state-specific rewards to the states of the CTMC. In our case, the CTMC models the availability of the WFMS components (see [8]). At each point in time when a failure occures or the repair of a failed component is finished, the CTMC makes a step into another state. The states of the CTMC represent the system states of the WFMS, i.e., 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* when the CTMC is in that state. Note that the overall configuration is merely the "upper bound" for the system states of interest. The transition rates between the states of the CTMC are derived by the mean time to failure (MTTF) and mean time to repair (MTTR) of the involved server types. We assume that the time periods of availability of a component and the component downtimes are exponentially distributed. This assumption is common in the performance evaluation community when Markov models are used for availability and reliability analysis [21]. However, we will show in Section 4.2 how to extend our model to capture also non-exponential, generally distributed, component downtimes and uptimes. As the reward for a given state of that availability CTMC, we use the mean waiting time of service requests of the WFMS in that system state. The steady-state analysis of the MRM yields the expected value for the waiting time of service requests for a given WFMS configuration with temporary performance degradation incurred by failures.

4 EXTENDED PERFORMABILITY MOD.

The models presented in Section 3 have essentially been developed in [8]. In this section, we extend the workload model to capture workload behavior in a more realistic manner. Section 4.1 discusses the realistic modeling of loops in the control flow, and Section 4.2 shows how to model non-exponentially distributed component downtimes and activity turnaround times.

4.1 Modeling of control flow loops

For loops in the control flow of a workflow, the number of loop iterations is usually a random variable with a discrete distribution. Without explicit considerations, the CTMC model would typically use a geometric distribution for the number of iterations, which is not exactly realistic. Rather we introduce a better approaches that captures uniformly distributed loop repetitions.

Observations on operational workflow applications show that most loops in workflows have the following properties:

- There is exactly one execution state that the loop starts with (i.e., we rule out jumps into the middle of the loop's body).
- There is exactly one execution state from which the loop exits. This state may be the source of several transitions that leave the loop, and there is exactly one transition that goes back to a state inside the loop's body.
- There is a positive lower bound $m < 0$ for the number of loop iterations (e.g., every order consists of at least one item).
- There is a finite upper bound $n < \infty$ for the number of loop iterations (e.g., one cannot order more items than the shop has in its catalog).
- The number of loop iterations can be described by a discrete distribution with values between *m* and *n*.

A commonly found distribution is the uniform distribution *UD(m,n)* with values from *m* to *n* (i.e., every value between *m* and *n* has the same probability). To capture this distribution by expanding the statechart in the following manner. Let $\{s_i,...,s_{i+j}\}$ be the set of execution states that the loop consists of. Without loss of generality, let s_{i+j} be the distinct state from which the loop can be left, and let s_i be the successor of s_{i+j} inside the loop (i.e., there is a transition from s_{i+j} to s_i). So we assume that the loop's body starts at s_i and ends at s_{i+j}, and s_{i+j} is the exit state of the loop.

1. State expansion:

   We substitute each state $s_x \in \{s_i,...,s_{i+j}\}$ of the CTMC by a set of *n* new states $s_{x,1},...,s_{x,n}$. So, we "clone" the states of the CTMC that are involved in the loop iterations so that the maximum number of loop iterations corresponds to visiting each clone of each state exactly once.

2. Entering the loop:

   For all states $s_x \notin \{s_i,...,s_{i+j}\}$ and $s_y \in \{s_i,...,s_{i+j}\}$ we add a transition from s_x to $s_{y,1}$ with the one-step probability $p_{x,y}$ of the transition form s_x to s_y in the original CTMC.

3. Exiting the loop or next iteration:

   For each potential number of loop iteration $o \in \{m,...,n\}$ we do the following:

- We add a transition from the exit state of the *o*-th iteration $s_{i+j,o}$ into the states $s_x \notin \{s_i,...,s_{i+j}\}$ outside the loop with the one-step probability $p_{\{i+j,o\},x} = \frac{1}{n-o+1} p_{i+j,x}$ with $p_{i+j,x}$ being the one-step probability from state s_{i+j} to state s_x in the original CTMC.
- If $o < n$ we add a transition from state $s_{i+j,o}$ into state $s_{i,o+1}$ (i.e., a new iteration starts) with the one-step probability
  $$p_{\{i+j,o\},\{i,o+1\}} = \left(1 - \frac{1}{n-o+1}\right)\left(p_{i+j,i} + \sum_{z \notin \{i,...,i+j\}} p_{i+j,z}\right)$$
  with $\sum_{z \notin \{i,...,i+j\}} p_{i+j,z}$ being the sum of the one-step

probabilities of the transitions from s_{i+j} into states outside the loop.

Note, that $\frac{1}{n-o+1}$ is the probability that the loop is left after exactly o steps if $o \in \{m,...,n\}$.

For each $o < m$ we add a transition from state $s_{i+j,o}$ to state $s_{i,o+1}$ (i.e., a new iteration starts) with a

one-step probability of 1.

4. Transitions inside the loop:

For each $o \in \{1,...,n\}$, each $x \in \{i,...,i+j-1\}$, and each $y \in \{i,...,i+j\}$ we add a transition from state $s_{x,o}$ to state $s_{y,o}$ with the one-step probability $p_{x,y}$ of the transition from s_x to s_y in the original CTMC.

Figure 6 shows the resulting CTMC expansions of the example workflow type Delivery assuming that the number of loop iterations is uniformly distributed with values from 3 to 5. The first visit to execution state s_1 does not belong to the loop in our interpretation; i.e., $s_{i+j} = s_1$ is the exit state and $s_i = s_2$ is the first state of the loop in this example. So we have to model the first visit to state s_1 in a separate way by adding the state $s_{1,0}$ to the CTMC.

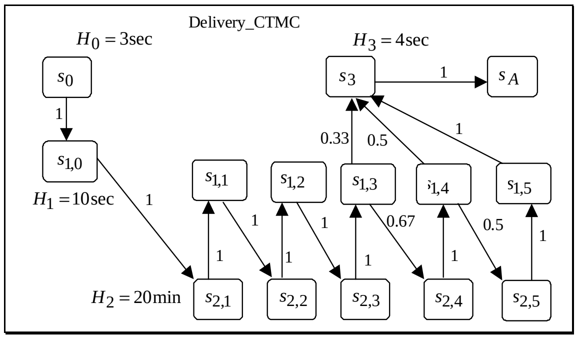


Figure 6 : Expanded CTMC of workflow type Delivery

4.2 Non-exponentially distributed component downtimes and activity turnaround times

In our availability model, i.e., the CTMC model sketched in Section 3.4, we assumed that the downtimes of the servers and the time from a server restart until the next crash are exponentially distributed with mean values MTTR and MTTF of the server type, respectively. Especially for server downtimes, this assumption is quite unrealistic. The duration of service maintenances or reboots after soft crashes, for example, are nearly constant. There is a mechanism to expand the CTMC such that non-exponentially distributed state residence times can be approximated. The mechanism is known in the literature as the *Phase Method* [22]. It approximates a positive random variable (i.e., the state residence time) by a mixture of Erlangian distributions with the same scale parameters, i.e., sums of independent exponentials with the same means. As a special case we are able to automatically substitute a state of the CTMC by a sequence of *h* states whose combined residence time is Erlang-*h*-distributed and thus approaches a constant with increasing *h*.

Figure 7 shows an example of such a substitution. The transition rates λ and μ describe the transition rates of the outgoing transitions of the state to be substituted. Note that the mean state residence time of a CTMC state is the reciprocal of the sum of the transition rates of its outgoing transitions. In the CTMC of the availability model, for a system configuration $Y = (Y_1,...,Y_k)$ and a state that represents X_1, ..., X_k currently operational servers of server types 1 through *k* a CTMC state has actually up to *2k* outgoing transitions to "adjacent" system states. For the purpose of emulating non-exponential state residence time by the Phase Method only the aggregate transition rate needs to be considered. Thus, in the state substitution λ is $\sum_i X_i / MTTF_i$ and μ is $\sum_i (Y_i - (X_i - 1)) / MTTR_i$.

The Phase Method can also be used to expand the CTMC of the flow process described in section 3.2.1 when the residence times of the execution states of a workflow type are not exponentially distributed (e.g., when the turnaround time of an activity is uniformly distributed or nearly constant).

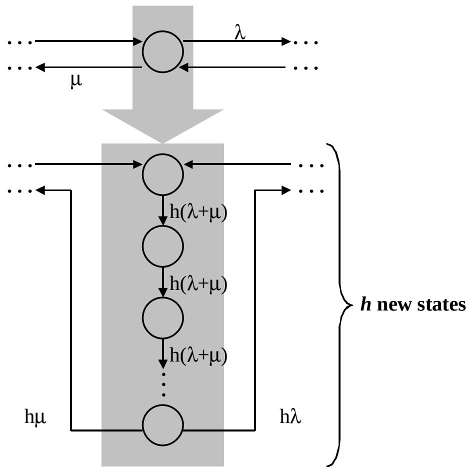


Figure 7 : Substitution of a CTMC state for non-exponential downtimes

5 EVALUATION

In this section, we present a systematic evaluation of the Goliat configuration tool. Section 5.1 describes the experimental testbed that we have developed based on the Mentor-lite WFMS prototype. Section 5.2 discusses the various parameters that we investigated in our experiments. The experimental results are presented in Section 6.

5.1 Experimental testbed

The experimental testbed is a combination of simulation and a real WFMS. The core of the experimental testbed is the workflow engine of our WFMS Mentor-lite [24]. The workflow engine is responsible for the interpretation and execution of the workflow specification. Depending on the workflow specification, the workflow engine starts activities and sends synchronization messages to other engines in the case of distributed workflow execution. So the entire workflow execution “logic” is processed by the full-fledged code of a real WFMS, i.e., the Mentor-lite prototype.

In addition to the Mentor-lite workflow engine in the core, the experimental testbed consists of the following components:

- A synthetic *load generator* starts new workflow instances following a given distribution for the arrivals of instances of different workflow types. We use Poisson arrivals as the most common traffic pattern. The arrival rate is a variable parameter of the experiments.
- A *monitoring module* observes the waiting times and the service times of service requests at the different servers of the WFMS and logs start and stop points of the activities.
- An *activity module* simulates the activities. The activity module receives incoming data from the workflow engine, sleeps for some time that corresponds to the human interaction time, and delivers result data back to the engine. If the activity is an automatic one the activity module additionally initiates a service request on an application server.
- A *crash module* simulates the downtime of servers. The mean time to failure (MTTF) and the mean time to repair (MTTR) of each server type are specified as parameters.

A distributed configuration of Mentor-lite consists of different workflow servers (i.e., instances of the workflow engine), application servers, and one communication server (i.e., ORB). Each server of the first two categories can be dedicated to a specified set of workflow activities or external applications, on a per type basis. Each of these dedicated servers and also the communication server can be replicated across multiple computers for enhanced performance and availability. The computers themselves are simulated using the CSIM library [4, 18] that provides primitives for “virtual resources” and performs all the necessary bookkeeping (for computing resource utilization, queue lengths, waiting times, etc.). The usage periods of the virtual resources are derived in different ways depending on the server type: for the workflow engine they are dynamically obtained, by online instrumentation, from the actual execution path lengths of the Mentor-lite code while workflow instances are processed; for application servers the simulated resource usage time is derived from the simulation parameters; and for communication servers the simulated usage periods have been calibrated by the results of offline measurements of the Orbix ORB used by the full-fledged Mentor-lite system.

So the entire testbed can be seen primarily as a simulation, with the Mentor-lite code embedded and really executed. This way the simulation is automatically validated through the online measurements of Mentor-lite path lengths. With this hybrid approach we aimed at easily reproducible experiments with high statistical confidence and flexible control over hardware resources (hence the virtual resources provided by CSIM), while at the same time capturing the real execution paths and timing behavior of our own workflow engine.

5.2 Parameter settings

5.2.1 Benchmark workflow

In the experiments we used only one workflow type, the benchmark workflow presented in Section 3.1. We set the parameters of the workflow as follows:

- Mean turnaround time of activities: The mean turnaround times of the activities of the benchmark workflow are shown in Table 1. The activities NewOrder and Payment are interactive activities, i.e., activities that require human interaction. These activities are executed on the user's client machine and do not induce any load on the application servers.

Activity	*Type*	*Mean Turnaround Time*
NewOrder	interactive	30 min
CreditCardCheck	automatic	10 sec
Notify	automatic	5 sec
FindStore	automatic	3 sec
CheckStore	automatic	5 sec
CreditCardPayment	automatic	12 sec
Payment	interactive	2 h

Table 1: mean turnaround time of activities

- Distribution of loop iterations: The number of iterations of the control flow loop in the Delivery subworkflow follows a discrete uniform distribution with lower bound 1 and upper bound 3.
- *Frequency of credit card payment:* The mode of payment influences the behavior of a workflow instance at two control flow splits that are jointly responsible for the lifetime of the instance and the load that the workflow instance induces on the application servers. In the case of credit card payment, the workflow results in two short-running automatic activities that stress the application (i.e., CreditCardCheck and CreditCardPayment), whereas payment by bill spawns an interactive activity. In the experiments, we set the relative frequency of credit card payment to 50%.
- *Frequency of credit card failures:* After the activity CreditCardCheck, the control flow is split depending on the result of the activity. If the check failed the workflow instance terminates immediately. So, the workload includes also workflow instances with extremely short lifetime. In the experiments we set the relative frequency of credit card failures to 10%.

For experiments with distributed workflow execution, we partitioned the benchmark workflow into two partitions. One partition includes all activities of the subworkflow Delivery; the rest of the activities forms the second partition. For each of the partitions there is one dedicated type of workflow servers and one dedicated type of application servers. In the alternative case of centralized workflow execution (i.e., the entire workflow is processed by a single workflow engine), the assignment of the application servers is the same as in the distributed case (i.e., there are also two types of application servers). All workflow servers and application servers communicate with each other via one type of communication servers.

5.2.2 Service times of server types

The service times of a request to the workflow servers are given by the CPU times that the Mentor-lite workflow engine needs for such a request in its real execution, which is measured online.

The service times of the communication servers leans have been determined offline from (relatively coarse) measurements of Orbix, a commercial CORBA implementation [12] that provides the communication middleware for Mentor-lite. Orbix also includes support for distributed transactions by its Object Transaction Service (OTS) component, which is used by Mentor-lite for reliable message exchanges between workflow engines. Table 2 gives the most important types of Orbix requests for the execution of (distributed) workflows with Mentor-lite and the setting of their mean service times in the experiments.

Request	*Mean Service Time*
Bind the communication server	100 msec
Bind an activity on application server	500 msec
Start an activity via IDL	75 msec
Data exchange via IDL	75 msec
Delete a registry entry at the ORB	50 msec
Begin of transaction in OTS	100 msec
Rollback a transaction in OTS	100 msec
Commit a transaction in OTS	100 msec

Table 2: mean service time of communication servers

The activity module of the experimental testbed induces exactly one service request for each automatic activity on an application server. The service time of such an service request is the mean turnaround time of the activity as given in Table 1.

5.2.3 Downtimes of server types

There are many possible causes of server failures: soft crashes caused by Heisenbugs which require rebooting and log-based recovery, hard crashes such as disk failures which require restoring data from a backup, and possibly even external denial-of-service attacks that result in heavy performance losses so that the system becomes unusable for its actual users. Heavily loaded servers are more likely to fail than lightly loaded ones. Finally, there are periodic server shutdowns for maintenance (e.g., software upgrades).

Server type	*MTTF*	*MTTR*
WFS 1	1 d	20 min
WFS 2	2 d	30 min
CS	6 h	15 min
AS 1	1 d	30 min
AS 2	12 h	20 min

Table 3: MTTF and MTTR of crashes of server types

Especially for frequent server downtimes, the accuracy of Goliat's steady-state performability model is crucial but also the most difficult metric to predict. We set the MTTF and MTTR of soft and hard crashes of servers to reflect an extremely stress tested system. Table 3 shows the values that we calibrated the crash module with. In addition, the crash module initializes a server once a week (i.e., with a continuous uniform distribution with values between 6 and 8 days) to model service shutdowns for a duration of 60 minutes. Each of these planned outages affects one server of a server type at a time; the outages of different servers are spread evenly across the entire one-week period.

6 EXPERIMENTAL RESULTS

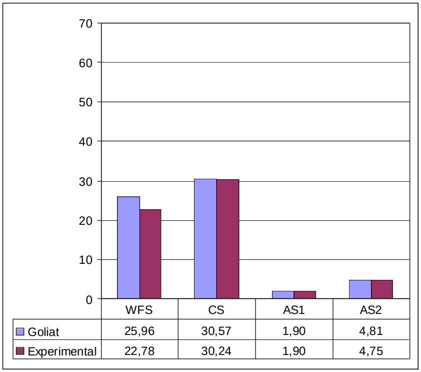
We studied two major degrees of freedom in our experiments:

1) Centralized vs. distributed workflow execution:

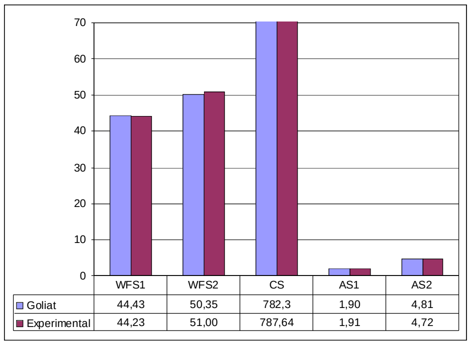
> The proper configuration of a WFMS critically depends on whether the workflow execution is distributed over several workflow engines. In particular, the communication servers have to sustain a higher load because of the synchronization messages that are exchanged between the workflow engines. Moreover, different workflow partitions are typically executed by different workflow servers. So we have to add a second type of workflow servers in experiments with distributed workflow execution.

2) Performance of a stable system vs. performability regarding transient outages:

> We evaluate the configuration tool's predictions for the system performance assuming that no servers will fail as well as the predictions for the system performability when transient server outages are taken into account. In the experiments we study both situations by switching the crash module of the testbed off or on.



(a) centralized workflow execution



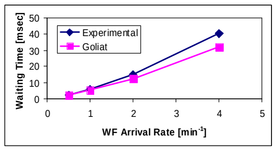
(b) distributed workflow execution

Figure 8 : Service requests per workflow instance

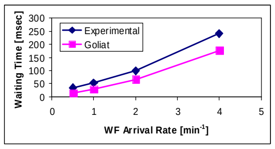
In the following, we present performance as well as performability results for centralized workflow execution and performance results for distributed workflow execution. In all experiments we measured the mean waiting times of service requests at the various server types as a function of the arrival rate workflow instances for a given system configuration. We varied the system configuration from experiment to experiment to are able to show the most meaningful results.

First, we discuss the accuracy of the approximation of the induced load per workflow instance using the CTMC model of Sections 3 and 4. Figure 8 shows Goliat's predictions for the expected number of service requests of a single workflow instance to the various server types versus the mean values of the requests that were actually observed in the experiments. For both, centralized (Figure 8(a)) and distributed (Figure 8(b)) workflow execution, the analytic predictions match the measurements very well. The increased number of service requests to the workflow servers and the communication server under distributed workflow execution is caused by Mentor-lite's polling of the transactional message queues via OTS when waiting for synchronization messages. Indeed, there are probably many issues for optimizing the Mentor-lite code, but this is not the subject of this paper.

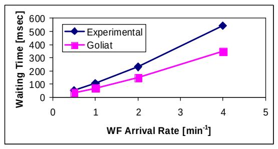
Figure 9 shows the results of our first, performance-oriented, experiment, where we switched the crash module off and executed the workflow instances in a centralized manner. The system configuration was set to 1 workflow server, 1 communication server, and 1 server of each application server type. The charts present the mean waiting times given observed in the experiments versus the predicted values from Goliat's analytic models, as a function of the arrival rate of new instances of the example workflow type. Goliat underestimates the mean waiting time of the service requests for all server types. But the underestimation is not critical and the approximation of the overall system performance, i.e., the mean waiting time averaged over the service requests of all server types (Figure 9(e)) is in an acceptable range reasonably close to the experimental results.



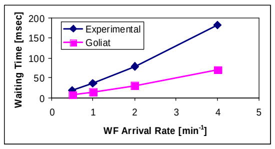
(a) workflow server



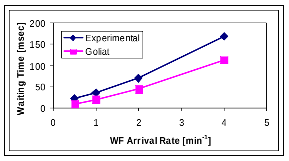
(b) communication server



(c) application server 1



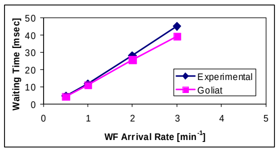
(d) application server 2



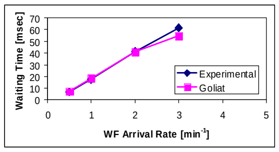
(e) overall system performance

Figure 9 : Server performance of the stable system under central workflow execution

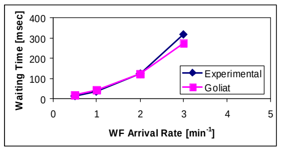
The performance results of the experiments with distributed workflow execution (and still deactivated crash module) are given in Figure 10. Because of the much higher load at the communication server when executing the workflows in a distributed manner (see Figure 8), we increased the number of replicas of the communication server type to 5. The number of servers of the additional workflow server type that is responsible for the execution of the subworkflow *Delivery* is set to 1. The measured as well as the predicted values of the mean waiting times of the application server types are the same as in the experiments with central workflow execution because the load at the application servers is independent of the execution strategy. Goliat's approximation of the mean waiting times is as accurate as in the previous experiment for the workflow servers and becomes even better for the communication servers. Goliat slightly overestimates the waiting times at the communication servers when the arrival rate is low because the round-robin assignment of the service requests among the 5 server replicas in the simulations balances the load better than the analytical model assumes. In contrast to the actual round-robin strategy, the model would partition the overall arrival stream into five streams so that each of the server replicas receives one fifth of the load, which could still lead to significant fluctuations of the load at each server. The round-robin assignment, however, smoothes the arrivals at each server to a much better extent. This effect can also be seen in the following experiments.



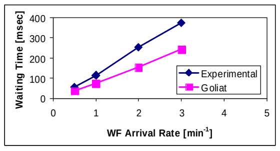
(a) workflow server 1



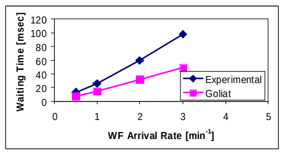
(b) workflow server 2



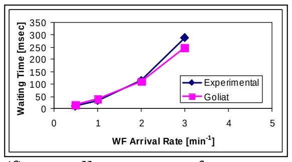
(c) communication server



(d) application server 1



(e) application server 2



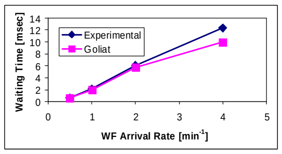
(f) overall system performance

Figure 10 : Server performance of the stable system under distributed workflow execution

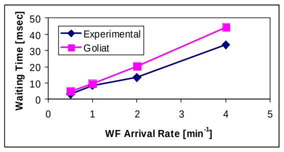
Figure 11 shows the results of our third setting where we increased the number of server replicas to 3 for each server type to meassure the performability under centralized workflow execution with transient server outages. With the exception of the workflow server type, the mean waiting times of the service requests are overestimated as discussed before. The mean waiting times of the workflow servers are slightly underestimated because, unlike for the other server types, the service requests are not scheduled in a round-robin manner; rather all service requests of a workflow instance are served by the same workflow server replica. The underestimation increases with the server's load because the model neglect transient queueing effects when a server fails and its active workflow instances have to be reassigned to available servers.

Finally, we ran a simulation with a configuration that was suggested by Goliat's recommendation component. We specified the goals as follows: the system availability should be at least 99.99 %, and the overall mean waiting time of service requests should be no higher than 10 milliseconds under a load of 2 workflow instances per minute. Goliat's recommendation for a minimum cost configuration that can satisfy these goals was: 3 workflow servers, 5 communication servers, and 3 application servers of both types. It turns out, by running additional experiments, that 4 replicas would be sufficient, but this is indeed the true minimum, and also none of the other degrees of replication can be reduced. So Goliat misses the true minimum-cost configuration by only one server, $^1/_{14}$ of the total cost.

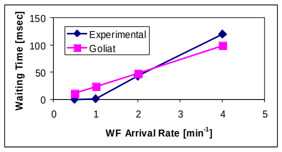
To summarize, Goliat's internal analytic models exhibit acceptable accuracy for both performance and performablity predictions. Goliat underestimates the mean waiting times of a server type when there is only one server of this type. On the other hand, it tends to overestimate the overall waiting times with increasing number of server replicas. Goliat's performability predictions result in a recommended system configuration that truly minimizes the number of server replicas while satisfying all specified goals.



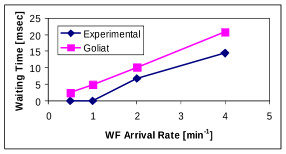
(a) workflow server



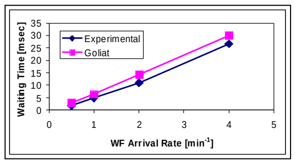
(b) communication server



(c) application server 1



(d) application server 2



(e) overall system performability

Figure 11 : Server performability under centralized execution

7 CONCLUSION

In [8] 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 of an assessment and configuration tool. As an initial step towards evaluating the viability of our approach, we have defined a WFMS benchmark [7] that we used for measurements of our own prototype, Mentor-lite, under different configurations. The current paper is the final step in this research program. We have presented the fully implemented Goliat tool, and have systematically evaluated its accuracy and practical viability using a comprehensive experimental testbed that combines simulated with online measurements of the real Mentor-lite code. We have also presented novel extensions to Goliat's models that are important to capture the behavior of a workflow system in a realistic manner. The presented experiments have shown that Goliat's predictions of both performance and performability are indeed reasonably accurate.

Apart from certain fine-tuning issues, the work on Goliat is completed. We plan to make both the Goliat software and the Mentor-lite system available as shareware source code. Our own plans for future work include adapting Goliat to a commercial workflow system and conducting more tests towards industrial-strength practicality.

REFERENCES

[1] G. Alonso, D. Agrawal, A. El Abbadi, C. Mohan, Functionality and Limitations of Current Workflow Management Systems, IEEE Expert, 12(5), 1997

[2] T. Bauer, P. Dadam, A Distributed Execution Environment for Large-Scale Workflow Management Systems with Subnets and Server Migration, CoopIS, 1997

[3] T Bauer, P. Dadam, Distribution Models for Workflow Management Systems - Classification and Simulation (in German), Informatik Forschung und Entwicklung, 14(4), Springer, 1999

[4] Mesquite Software Homepage, http://www.mesquite.com

[5] Data Engineering Bulletin Special Issue on Infrastructure for Advanced E-Services, 24(1), 2001

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

[7] M. Gillmann, R. Mindermann, G. Weikum, Benchmarking and Configuration of Workflow Management Systems, CoopIS, 2000

[8] M. Gillmann, J. Weissenfels, G. Weikum, A. Kraiss, Performance and Availability Assessment for the Configuration of Distributed Workflow Management Systems, EDBT, 2000

[9] C. Hagen, G. Alonso, Highly Available Process Support Systems: Implementing Backup Mechanisms, SRDS, 1999

[10] D. Harel, E. Gery, Executable Object Modeling with Statecharts, IEEE Computer, 30(7), 1997

[11] G. Haring, C. Lindemann, M. Reiser (Eds.), Performance Evaluation: Origins and Directions, Lecture Notes in Computer Science (LNCS), Vol. 1769, Springer, 2000

[12] IONA Technologies PLC, User Manuals of Orbix 2.3, 1997

[13] J. Klingemann, J. Waesch, K. Aberer, Deriving Service Models in Cross-Organizational Workflows, RIDE, 1999

[14] M. Kamath, G. Alonso, R. Günthör, C. Mohan, Providing High Availability in Very Large Workflow Management Systems, EDBT, 1996

[15] A. Lazcano, G. Alonso, H. Schuldt, C. Schuler, The WISE approach to Electronic Commerce, Int'l. Journal of Computer Systems Science & Engineering, 15(5), 2000

[16] F. Leymann, D. Roller, Production Workflow, Prentice Hall, 1999

[17] P. Mills, C. Loosley, A Performance Analysis of 40 e-Business Web Sites, White Paper, Keynote Systems Inc., 2001

[18] H.D. Schwetmann, Model-based Systems Analysis Using CSIM18, WSC, 1998

[19] G. Shegalov, M. Gillmann, G. Weikum, XML-enabled Workflow Management for E-Services across Heterogeneous Platforms, VLDB Journal Special Issue on E-Services, 10(1), Springer, 2001

[20] H. Schuster, J. Neeb, R. Schamburger, A Configuration Management Approach for Large Workflow Management Systems, WACC, 1999

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

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

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

[24] J. Weissenfels, M. Gillmann, O. Roth, G. Shegalov, W. Wonner, The Mentor-lite Prototype: A Light-Weight Workflow Management System, ICDE, California, 2000