ELSEVIER

Decision Support Systems 30 (2001) 451–468

Decision Support Systems

www.elsevier.com/locate/dsw

Re-engineering knock-out processes

W.M.P. van der Aalst *

Faculty of Technology and Management, Department of Information and Technology, Eindhoven University of Technology, P.O. Box 513, NL-5600 MB, Eindhoven, Netherlands

Abstract

The core of many workflow processes in banks, insurance companies, governmental departments, and administrations of multinationals is formed by a set of tasks that are used to classify cases into two groups: accepted and rejected. Each of these tasks has two possible outcomes: OK or NOK (i.e., Not OK). If for a specific case all tasks result in OK, the case is accepted, otherwise it is rejected. In this paper, we concentrate on the order in which these tasks need to be executed to yield an 'optimal' process with respect to the utilization of resources and flow time. Both sequential and parallel routing are considered. The effect of combining tasks is also investigated. A step-wise approach consisting of 11 concrete re-engineering rules is given. The approach is supported by a simulation toolbox ExSpect/KO. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Workflow management; Business process re-engineering; Decision support systems; Simulation; Performance evaluation

1.Introduction

Today's successful corporations consider the business processes to be the crown jewels of the organization and are constantly challenged to improve the performance and efficiency of these processes. There are several reasons for the increased interest in business processes. First of all, management philosophies such as Business Process Re-engineering (BPR) and Continuous Process Improvement (CPI) stimulated organizations to become more aware of their business processes. Secondly, today's organizations need to deliver a broad range of products and services. As a result, the number of processes inside organizations has increased. Consider, for example, mortgage loans. A decade ago, there were just a few types of mortgages, at the moment numerous types are available. Not only the number of products and services has increased, also the lifetime of products and services has decreased in the last three decades. As a result, today's business processes are also subject to frequent changes. Moreover, the complexity of these processes increased considerably. Finally, today there are generic tools such as Workflow Management (WFM) and Enterprise Resource Planning (ERP) systems, which allow for the definition, execution, registration, and control of workflow processes. These tools have triggered many organizations to rethink their business processes.

The increased interest in business processes, in particular the re-engineering of workflow processes, has uncovered the Achilles' heel of business process management: the lack of concrete quantitative guidelines for the design of processes in the service

* Tel.: +31-40-247-4295.
E-mail address: w.m.p.v.d.aalst@tm.tue.nl (W.M.P. van der Aalst).

PII: S0167-9236(00)00136-6

industry (i.e., banks, insurance companies, and administrations). Existing approaches are either not applicable by end-users or focus on qualitative aspects. *Queuing theory* [26], in particular the analysis of queuing networks [16,18], provides many results and has been applied in the domain of (flexible) manufacturing systems [12], computer networks [10], and telecommunication systems [35]. However, the results described in literature are only applicable to specific situations, require advanced tools, and do not provide concrete guidelines for re-engineering. Nevertheless, it is clear that queuing theory could provide a firm theoretical basis for re-engineering efforts. *Simulation* is a very flexible technique. Modern simulation packages allow for both the visualization and performance analysis of a given process [6,22]. Unfortunately, it takes a lot of time to build a simulation model, accurate interpretation of simulation results requires statistical knowledge, and simulation only supports 'what–if analysis', i.e., it does not suggest improvements. Literature on BPR [1,15, 20,21,24,29,30,32] and WFM [19,23,25,27,28,37] focuses on the organizational and technical aspects rather than formulating quantitative rules for re-engineering. For example, the 36 'process improvement rules' provided by Poyssick and Hannaford in Ref. [32] are of a qualitative nature and do not give any concrete support for the design of the control of a given business process. The paper by Buzacott [11] is one of the few papers targeting at quantitative redesign rules for business processes.

In this paper, we focus on the re-engineering of *knock-out processes*. For this type of process, we try to bridge the gap between queuing theory and simulation on the one hand and qualitative approaches on the other hand. A knock-out process is a workflow process with a specific structure [4]. As any workflow, a knock-out process is *case-based*, i.e., every piece of work is executed for a specific *case*. Examples of cases are a mortgage, an insurance claim, a tax declaration, an order, or a request for information. Cases are often generated by an external customer. However, it is also possible that a case is generated by another department within the same organization (internal customer). The goal of a knock-out process is to decide whether the case should be accepted or rejected. To make this decision, a number of tasks need to be executed. Each task has two possible results: OK or NOK (i.e., not OK). If for a specific case a task results in NOK, the case is rejected immediately. Only if all tasks have a positive result, the case is accepted. Many workflow processes have parts, which can be considered to be knock-out processes. Handling an insurance claim, a request for a loan, a job application, and the reviewing of paper for publication in a journal are typical examples of processes with a knock-out structure. This paper provides a set of rules for the redesign of knock-out processes. The rules are easy to apply and do not require advanced tool support. However, the rules are heuristics and do not give decisive answers to all questions. Therefore, the approach presented in this paper is supported by the simulation toolbox *ExSpect / KO* specifically developed for the analysis of knock-out processes. The toolbox is implemented using the Petri-net-based simulation package ExSpect [8,9].

The paper is organized a follows. First, a formal definition of a so-called knock-out problem and the associated class of knock-out processes are given. Then, using a three-step approach, issues such as the ordering of tasks, the combining of tasks, and the parallel execution of tasks are addressed. Finally, the toolbox ExSpect/KO is described. To illustrate the approach, the process of handling a request for a mortgage is used throughout the paper.

2. Knock-out processes

A knock-out problem is a business situation where for each case a pre-specified set of tasks needs to be executed. As indicated in the introduction, the processing of a task stops immediately if in one of the tasks a reason for rejection is detected. Only cases that successfully pass all tasks are accepted. A knock-out problem consists of an arrival process (i.e., new cases), a set of tasks, a set of resource classes, and a set of precedence constraints. The arrival process is specified by the arrival rate, i.e., the average number of new cases that arrives each time unit. In this paper, we assume a Poisson arrival process [16,26]. Tasks are mapped onto *resource classes*, i.e., a task requires a resource of a specific resource class. Per resource class, the number of available resources is given. In a typical organization

with knock-out processes, e.g., a bank, an insurance company, or a public administration, the resources are mainly human. However, because the approach is not restricted to these organizations, we prefer the term resource. To facilitate the allocation of *work items* to resources, resources are grouped into classes. A work item is a concrete piece of work, i.e., a task which needs to be executed for a specific case. A resource class is a group of resources with similar characteristics. There may be many resources in the same class, but a resource may only be a member of one resource class. If a resource class is based on the capabilities (i.e., functional requirements) of its members it is called a *role*. If the classification is based on the structure of the organization, such a resource class is called an *organizational unit* (e.g., team, branch or department). A work item, which is being executed by a specific resource, is called an *activity*.

Each task has an average processing time, a failure probability, a reject probability, and a set-up time ratio. At some points in this paper, the service time distribution of tasks is assumed to be negative exponential. However, for most of the results we do not need this assumption. The failure probability specifies the likelihood that a task has to be redone because of some human or technical error. The set-up time ratio is the average percentage of time that is spent on preparations like reading the case data. The reject probability is the percentage of cases that do not pass the task successfully. If the task has a reject probability of 0.22, then for 22% of the cases the outcome of this task is NOK and in 78% the result is OK. Throughout this paper, we assume that the reject probability of a task is independent of its location in the process, i.e., reject probabilities are mutually independent. In general, tasks cannot be executed in an arbitrary way. Some tasks need results produced by other tasks. Therefore, constraints with respect to the ordering of tasks are specified in the so-called precedence relation. In principle there are three ways to execute tasks: tasks can be executed sequentially, tasks can be combined into composite tasks, and tasks can be executed in parallel. If tasks are combined into a composite task, only one set-up is needed. Clearly, the set-up time reduction can improve performance. However, the whole composite task needs to be executed before the decision, OK or NOK, can be made. Moreover, a failure results in a rollback of the whole composite task. Parallel routing can reduce the flow time because multiple resources can work on a case at the same time. However, just as with composite tasks, all parallel flows need to be executed before the decision, OK or NOK, can be made. Only after synchronization, it can be decided whether the case requires further processing or not. This is a reasonable assumption since generally it is not possible to withdraw a case from one parallel branch if a reason for rejection was detected in another parallel branch. It may also take some time to synchronize parallel flows. Therefore, we specify a fixed synchronization time. The data associated to a knock-out problem can be formulated by an 11-tuple capturing all aspects discussed.

Definition 1 (KOPROB). A *knock-out problem KOPROB* is an 11-tuple (T, R, ra, pt, fp, rp, sr, nr, st, ar, pr), where:

- T, the set of *tasks*,
- R, the set of *resource classes*,
- $ra \subseteq T \rightarrow R$, the *resource assignment* function,
- $pt \in T \rightarrow R^+$, the *processing time* of each task,
- $fp \in T \rightarrow [0,1]$, the *failure probability* of each task,
- $rp \in T \rightarrow [0,1]$, the *reject probability* of each task,
- $sr \in T \rightarrow [0,1]$, the *set-up time ratio* of each task,
- $nr \in R \rightarrow N$, the *number of resources* for each task,
- $st \in R^+$, the *synchronization time*,
- $ar \in R^+$, the *arrival rate*,
- $pr \subseteq T \times T$, the *precedence relation.*

Consider the following knock-out problem. Before a bank lends money on mortgage for buying a house. The bank executes five tasks to check out the potential mortgagee and the house (s)he is planning to buy:

(A) Check salary of mortgagee: is the income sufficient to pay off the debt in a reasonable period?
(B) Check current debts: does the mortgagee have serious debts?

Table 1
Data for the knock-out problem

task	ra	pt	fp	rp	sr
A	X	35	0.05	0.10	0.30
B	X	30	0.05	0.15	0.30
C	Y	20	0.10	0.20	0.10
D	Y	15	0.10	0.15	0.10
E	Z	20	0.05	0.20	0.10

(C) Check mortgage history: does the mortgagee have a history of non-payment?
(D) Check collateral: is the value of the real estate as indicated by the mortgagee?
(E) Check insurance: is there a reason why the mortgagee will not be able to get a life insurance?

If result of any of these checks is NOK, the request is rejected. Only if all checks are positive, the mortgage is accepted. Clearly, the process is a knock-out process. There are five tasks. For convenience, we call them A, B, C, D, and E. Tasks A and B are executed by financial experts (resource class X). Tasks C and D are executed by employees of the mortgage department (resource class Y). Task E is executed by insurance experts (resource class Z). The resource classes X, Y, and Z contain, respectively, six, four, and three employees allocated to this knock-out process. There are only two precedence relations: both task D and task E need the result of task C, i.e., $pr = \{(C, D), (C, E)\}$. Each day (8 h), 40 new requests arrive, i.e., 5 h^{-1} ($ar = 5$). The time needed to synchronize the result of multiple tasks is just 6 min ($st = 0.1$). The remaining information about the knock-out problem is given in Table 1.
Note that the probability that a case is accepted is $\prod_{t \in T}(1 - rp(t))$. Therefore, only 42% of the requests for a mortgage are accepted and 58% are rejected. Also, note that a knock-out problem just specifies information about the environment, the resources, and the tasks. It does not specify the process, i.e., the routing of cases. For the specification of a knock-out process, we need to define the routing of a case.

Definition 2 (KOPROC). Given a KOPROB with a set of tasks, a *knock-out process* (*KOPROC*) is a term of the language defined by the following grammar, where PROC is the start symbol, every task appears exactly once, and the precedence relation is satisfied:

PROC:: = SPROC|[PPROC]
SPROC:: = TASK|TASK.PROC
PPROC:: = PROC|PROC,PPROC
TASK:: = ATASK|(CTASK)
CTASK:: = ATASK|ATASK,CTASK
ATASK:: = $t_1|t_2|\ldots|t_n$

The grammar given in Definition 2 uses the standard EBNF (Extended Backus Naur Form, cf. Ref. [38]) notation where a vertical bar indicates a choice, i.e., an expression of type PROC is either a expression of type SPROC or an expression of type PPROC enclosed by brackets. An expression of type PPROC is a list of expressions of type PROC separated by commas, etc. The brackets are used to denote parallel composition. The parenthesis's are used to form composite tasks. The dots are used to denote sequential composition. Consider for example the knock-out process shown in Fig. 1. This process corresponds to the knock-out problem described earlier and can be represented by the term A.[B,(C,D)].E. First task A is executed. Then in parallel task B and the combined task CD are executed. Finally, task E is executed. This process is consistent with the knock-out problem formulated earlier because all tasks appear once and the two precedence constraints are not

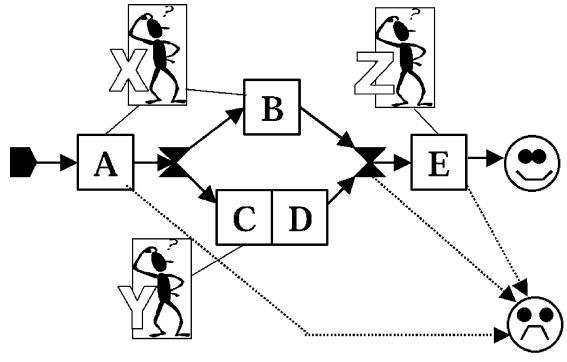


Fig. 1. A knock-out process.

violated (C is executed before D and E). Other examples of knock-out processes, which are consistent with the knock-out problem formulated earlier, are: A.B.C.D.E (a purely sequential process), C.[A,B,D,E] (first task C, then all remaining tasks in parallel), (A,B,C,D,E) (one huge combined task), and C.E.[D,(B,A)].

Knock-out processes can be supported by any of the available workflow management systems equipped with the basic routing primitives (e.g., OR-split, OR-join, AND-split, and AND-join [28]). To illustrate this we have realized the knock-out process shown in using Staffware. Staffware [36] is one of the leading workflow management systems with more than 550,000 user worldwide. Fig. 2 shows the knock-out process A.[B,(C,D)].E in the design tool of Staffware.

We make the following assumptions with respect to the order in which work items are handled and the way the work items are distributed over the resources:

- If the execution of a task fails, the resource is not released, i.e., a new run of the task is executed by the same resource without any delay.
- Work items are executed in first-come-first-served (FCFS) order per resource class, i.e., per resource class there is one central queue of work items which may correspond to multiple tasks ($ra^{-1}(r)$).
- There are no limitations with respect to the amount of work-in-progress, i.e., there is infinite buffer capacity in-between tasks.
- If tasks are combined into a composite task, the composite task is executed in one long run by one resource which is not released until all subtasks are completed.

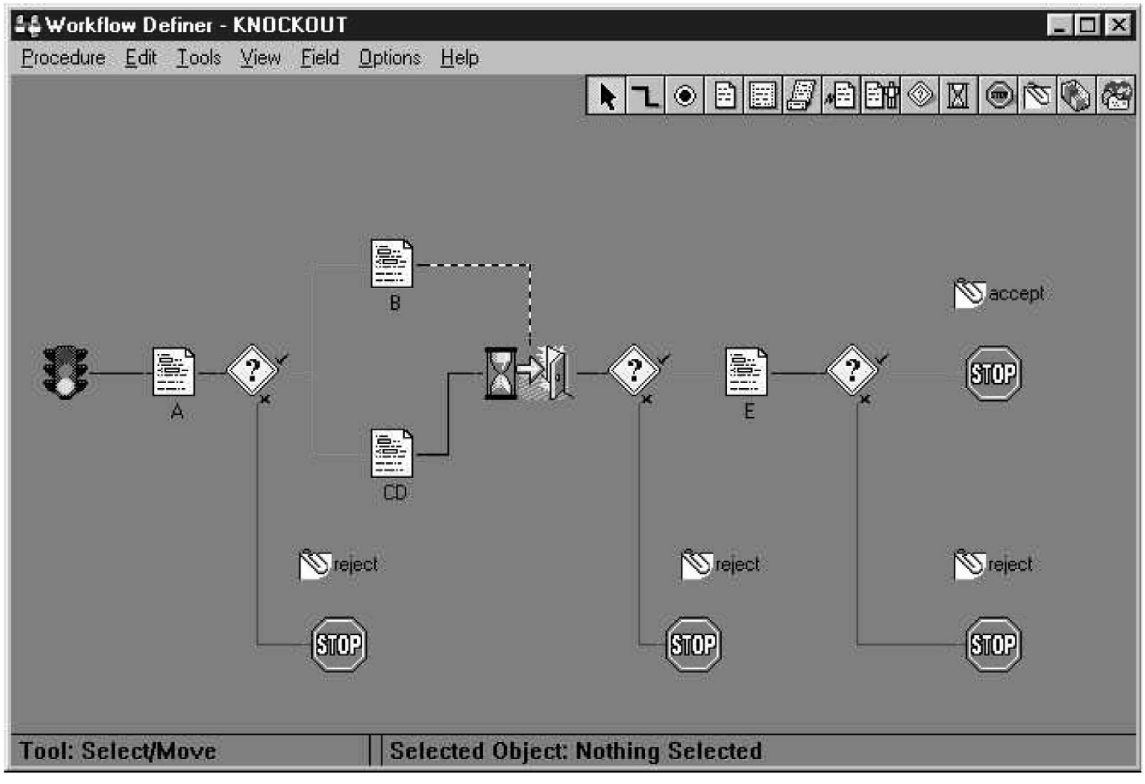


Fig. 2. Knock-out process A.[B,(C,D)].E modeled with the Staffware Graphical Workflow Definer.

For any KOPROC it is possible to calculate the probability that a specific task t is executed. We will use $\mathrm{ep}(t)$ to denote this probability. The actual processing time of a task is given by $\mathrm{apt}(t)$. If t is combined and is not the first subtask of the composite task, then $\mathrm{apt}(t) = \mathrm{pt}(t)(1 - \mathrm{sr}(t))$, otherwise $\mathrm{apt}(t) = \mathrm{pt}(t)$. The utilization of a resource class is given by:

$$\mathrm{ut}(r) = \frac{\mathrm{ar} \sum_{t \in \mathrm{ra}^{-1}(r)} \frac{\mathrm{ep}(t)\mathrm{apt}(t)}{1 - \mathrm{fp}(t)}}{\mathrm{nr}(r)}$$

This measure is an extension of the standard measure for utilization in a queuing network with Markovian routing [16,18]. Consider the knock-out process shown in Fig. 1: $\mathrm{ep}(A) = 1.00$, $\mathrm{ep}(B) = 0.90$, $\mathrm{ep}(C) = 0.90$, $\mathrm{ep}(D) = 0.90$, and $\mathrm{ep}(E) = 0.52$. The utilization of the three resource classes is as follows: $\mathrm{ut}(X) = 0.90$, $\mathrm{ut}(Y) = 0.69$, and $\mathrm{ut}(Z) = 0.30$. A process is called non-stable or infeasible if the amount of work arriving each time unit exceeds the capacity of one of the resource classes.

Definition 3 ((stability)). A KOPROC is *stable* if and only if for all resource classes $r \in R$, the utilization is less than 100% ($\mathrm{ut}(r) < 1$).

The process shown in Fig. 1 is stable because the utilization of each of three resource classes is less than 1. However, an intriguing question is whether it is possible to improve the process with respect to resource utilization and flow times. In the remainder of this paper, we will supply some heuristics and tools to answer this question. We will concentrate on the following three *key performance indicators*:

- *Resource utilization*: the average percentage of time resources are occupied (occupation rate).
- *Maximal throughput*: the maximum number of cases the process can handle (on average) per time unit (capacity).
- *Flow time*: the average time needed to process a case (processing time, includes both waiting and service time).

If the resource utilization of a resource class is 0.8, then, on average, resources are idle 20% of the time. The maximal throughput of a workflow process is determined by the capacity of the most restricting bottleneck. A process with a maximal throughput of 20 can handle, on average, up to 20 new cases per time unit. If more than 20 new cases arrive each time unit, the process will be overloaded and unable to process all requests. The flow time is the average time the workflow process takes to completely process a case, i.e., it is based on the time that passes from the time of arrival of a new case to the time of its completion.

In this paper we focus on the *steady-state behavior* of workflow processes having a knock-out structure, i.e., we assume a steady flow of new cases and a fixed number of resources. However, for operational support the steady-state behavior is of secondary importance. For concrete operational decisions (e.g., accept new cases or not) the current state and the transient behavior are of prime importance. In [33] we coined the term *short-time simulation* for the situation where the simulation is based on operational control data (e.g., work-in-progress and expected arrival patterns of new cases based on historical data) and analysis is limited to a given time horizon (e.g., one month). In fact, we have established a run-time connection between the workflow management system COSA [14] and our simulation tool ExSpect [9] to allow for this type of simulation. Since today's business processes are subject to frequent changes, it is important to have a direct link between the run-time system and the decision support system. The interested reader is referred to [33] for more information. However, to goal of this paper is to identify re-engineering rules. Since we are primarily interested in rules of thumb rather than complex rules for operational control, the focus on the steady-state behavior is justified.

The remainder of this paper is organized as follows. First, we consider the ordering of tasks for a knock-out problem without considering the combination of tasks and parallel processing. Subsequently, we focus on the guidelines for combining tasks into composite tasks. Then we consider the pros and cons associated to putting subsequent tasks in parallel. The resulting propositions and heuristics are consolidated into a three-step approach to handle knock-out problems. This approach is based on a number of simplifying assumptions. Therefore, we also present the simulation toolbox ExSpect/KO.

3. Swapping tasks

Many knock-out processes are characterized by the fact that the degree of freedom with respect to the order in which tasks can be executed is quite high. Most tasks correspond to checks which can be executed in any order. A task is selective if the reject probability is high. A task is expensive if the average processing time is high. Clearly, it is wise to start with selective tasks that are not expensive and postpone the expensive tasks which are not selective as long as possible. This rule of thumb is outlined in the following proposition.

Proposition 1. *Tasks sharing the same resource class should be ordered in descending order using the ratio [rp(t)(1 − fp(t))] / (pt(t)) to obtain an optimal process with respect to resource utilization and maximal throughput.*

Proof. Consider two subsequent tasks t_1 and t_2. The average workload generated by these two tasks for a case waiting for task t_1 is $[(\mathrm{pt}(t_1))/(1-\mathrm{fp}(t_1))] + [(1-\mathrm{rp}(t_1))(\mathrm{pt}(t_2))/(1-\mathrm{fp}(t_2))]$. By reversing both tasks the average workload corresponding to the two tasks is $[(\mathrm{pt}(t_2))/(1-\mathrm{fp}(t_2))] + [(1-\mathrm{rp}(t_2))(\mathrm{pt}(t_1))/(1-\mathrm{fp}(t_1))]$. Hence, if $[(\mathrm{pt}(t_2))/(1-\mathrm{fp}(t_2))] + [(1-\mathrm{rp}(t_2))(\mathrm{pt}(t_1))/(1-\mathrm{fp}(t_1))]$ is smaller than $[(\mathrm{pt}(t_1))/(1-\mathrm{fp}(t_1))] + [(1-\mathrm{rp}(t_1))(\mathrm{pt}(t_2))/(1-\mathrm{fp}(t_2))]$, then swapping results in a lower resource utilization. By solving this inequality we obtain the ratio. □

It should be noted that the results presented in this section only hold for knock-out processes where there is no parallelism and no tasks are combined into composite tasks. Nevertheless, we will use the knock-out process shown in Fig. 1 (which contains both parallelism and a composite task) to illustrate some of the results. The issues of composition and parallelism are addressed in subsequent sections.

Proposition 1 also has an alternative interpretation. If is interpreted as the processing costs instead of the processing time, then Proposition 1 shows how the tasks should be ordered to minimize the average costs. Note that such calculation only makes sense if the costs are truly variable or the savings can be redirected to other processes.

Task A and task B in the knock-out process shown in Fig. 1 are both executed by a resource of the resource class X. Applying Proposition 1 shows that resource utilization and maximal throughput will improve if task A and task B are swapped. Task C and task D are both executed by resources of class Y. In the remainder of this section we ignore the fact that there is a precedence constraint which states that task C needs to be executed before task D and focus on the desired ordering of these two tasks. Since the ratio mentioned in Proposition 1 is equal to 0.54 for both tasks C and D, it is not clear what the best order is. Therefore, we explore the effect of changing the ordering of tasks on the average flow time.

Proposition 2. *Tasks sharing the same resource class and having identical exponential service times (including failures) should be ordered using the reject probability (in descending order) to obtain an optimal process with respect to resource utilization, maximal throughput, and flow time.*

Proof. In this paper we assume a Poisson arrival process and a FCFS (first-come–first-served) queuing discipline. Also, note that in this section we do not consider composition and parallelism. Therefore, the workflow process can be described by a product form queuing network [16,18] if the routing is Markovian, the service times are exponential, and the average service time is the same for every class of clients, i.e., the queuing network corresponds to the first class of BCMP networks (first described by Baskett et al. [10]). For this class of BCMP networks the service discipline at each station is FCFS and the service time distributions must be identical and exponential for all classes of clients (cf. page 250 in Ref. [10] or Chapter 4 of Ref. [18]). Please note that the term “client” is the standard term used to denote customers or objects flowing through the network. These clients correspond to cases. The mapping of the workflow process onto such a BCMP network is as follows. All tasks sharing the same resources are mapped onto one FCFS station. If multiple tasks are mapped onto one station, then several classes of clients are used to allow for a Markovian routing. After the first service, the case switches from one client class to another. Within one class the routing is Markovian, but a case can be transferred from one class to another after executing a specific task. This

way it is possible to enable different successor tasks for tasks mapped onto the same FCFS station. Note that it is essential that the average processing time is the same for all tasks mapped onto the same station. If this is not the case, the resulting network does not have a product form solution. Under the assumptions given it is easy to calculate the average flow time. Since both the resource utilization and the average time spent on each case are minimized by starting with the task with the highest reject probability, it is straightforward to show that the average flow time is also minimized. □

If all tasks are about the same size, the ordering given in Proposition 1 is also optimal with respect to the flow time. However, if the sizes of the tasks differ considerably and the resource utilization is low, it is better to use the following heuristic. In this paper, the term 'heuristic' is used for statements that do not hold for every knock-out problem. For each of the heuristics presented in this paper, we need additional assumptions to give a formal proof of their correctness. Nevertheless, they can be used as rules of thumb for the re-engineering for knock-out processes.

Heuristic 1. *Tasks sharing the same resource class should be ordered in descending order using the ratio* $rp(t)/\{C + [pt(t)/(1 - fp(t))]\}$ *to obtain an optimal process with flow times, where the resulting resource utilization is acceptable and C is a suitably chosen constant. C should be of the same order of magnitude as the average waiting time per task.*

It is easy to see that Proposition 2 is a special case of this heuristic: If the ratio $[(\mathrm{pt}(t))/(1 - \mathrm{fp}(t))]$ is the same for all tasks sharing a resource class, then the tasks should be ordered using the reject probability. Note that in this particular case the ordering is independent of the constant *C*.

To understand the effect of having smaller and larger tasks sharing the same resources, consider the following example. There are two tasks: T1 and T2. Task T1 costs two time units and rejects 25% of the cases. Task T2 costs four time units but rejects 50%. According to the ratio in Proposition 1, the order in which T1 and T2 are executed does not have any effect on the resource utilization (both alternatives require an average total processing time of five). Let us assume that the waiting time for each task does not change when the order is reversed. In general this is not true, but, since the utilization does not change, the waiting time will not change dramatically. If the average waiting time is W, the average flow time in the situation where task T1 is executed before T2 is $W + 2 + 3/4(W + 4)$. If the order of T1 and T2 is reversed the average flow time is $W + 4 + 1/2(W + 2)$. Hence, the average flow time of the second alternative is time units less. The explanation of this phenomenon is simple. By taking the large but highly selective tasks first, the average number of times a case has to queue is reduced. Based on this observation the constant C is added to the ratio in Heuristic 1. Note that this heuristic should be applied carefully, because changing the order of the tasks will have some effect on the waiting times!

If we apply Heuristic 1 to the example shown in Fig. 1, it is clear that task C should be executed before task D if we consider flow times (although in both cases the ratio in Proposition 1 is 0.54). In this case, any value for the constant C will lead to the same conclusion. However, in general, the constant should be chosen close to the expected waiting time. This waiting time can be estimated based on historical or empirical data or a simple approximation based on some analytical model, e.g., the average queuing time in an M/M/1 queue [26] with the same utilization rate and average processing time. These models can be found in any textbook on queueing theory, e.g., Chapter 1 of Ref. [18]. Please note that the knock-out process shown in Fig. 1 does not have a product-form solution because of the synchronization before executing task E. Therefore, the conditions mentioned in Proposition 2 are not satisfied if all tasks have identical exponential service times.

In Proposition 1, Proposition 2, and Heuristic 1 we considered tasks sharing the same resource class. If tasks use different resource classes, the ratio in Proposition 1 can still be used to minimize the overall utilization. However, the overall resource utilization is of less importance. For example, no resource class should have a utilization of more than 100%. A minimal overall resource utilization may lead to an unbalanced situation with serious bottle-

necks. Therefore, it is important to focus on the effect of utilization on the average flow time. Suppose that for each task the waiting time $wt(t)$ is given by the following formula: $wt(t) = (ut(r)pt(t))/(1 - ut(r)(1 - fp(t)))$. This formula corresponds to the average waiting time for an M/M/1 queue [18,26] extended with rework as a result of failures. Note that if the utilization is close to 100%, the average waiting time tends to be long. Based on this assumption we formulate the following heuristic.

Heuristic 2. *To obtain an optimal process with respect to flow time, tasks in a sequential knock-out process should be ordered in such a way that the following ratio is minimized* $\sum_{r \in R} 1 / [1 - ur(r)] \sum_{t \in ra^{-1}(r)} [ep(t)pt(t)] / [1 - fp(t)]$.

Heuristic 2 gives exact answers if each resource class is a singleton, serves a single task, and service times are negative exponentially distributed. Under these circumstances, the workflow process corresponds to product form queuing network [16]. To be more precise: a BCMP network of class 1 (cf. Ref. [10], page 250). In a way similar to the Proof of Proposition 2, it can be shown that the flow time is minimized under these assumptions. Note that local rules such as Proposition 1 and Heuristic 1 do not apply to the situation with multiple resource classes. Swapping two tasks, which correspond to different resource classes, will not only affect the waiting times of all tasks corresponding to both resource classes; it will also affect the utilization rates of all resource classes executing one or more tasks in-between.

If we abstract from the two precedence constraints (C before D and C before E), there are 120 possible sequential processes that fit the problem description given earlier. Using the ExSpect toolbox described in the second part of this paper, we calculated the ratio mentioned in Heuristic 2 for all of the processes. It turns out that the process E.C.D.B.A has the smallest ratio (3.09). The process with the highest ratio is A.B.C.D.E (12.93). Both schedules are stable. However, process E.C.D.B.A violates the precedence constraint that task C has to be executed before task E. There are 40 sequential processes satisfying the precedence relation, pr. Of these 40 knock-out processes, the process C.E.D.B.A has the smallest ratio (3.31). In this process the workload is more balanced ($ut(X) = 0.51$, $ut(Y) = 0.65$, and $ut(Z) = 0.46$) than in the process A.B.C.D.E ($ut(X) = 0.90$, $ut(Y) = 0.56$, and $ut(Z) = 0.30$).

It is possible to combine both heuristics, i.e., extend the ratio in Heuristic 2 with a factor for the number of times a case has to queue. The extended ratio is:

$$\left(\sum_{r \in R} \frac{1}{1 - ut(r)} \sum_{t \in ra^{-1}(r)} \frac{ep(t)pt(t)}{1 - fp(t)} \right) + \left(C \sum_{r \in R} ep(t) \right)$$

The constant C should be of the same order of magnitude as the average waiting time per task.

4. Combining tasks

In the previous section we assumed tasks to be atomic, i.e., there were no composite tasks. In this section, we investigate the effect of combining tasks. Consider two tasks which are mapped onto the same resource class. If the two tasks are combined into one composite task, they are executed by one resource without interruption. This way no setup is needed for the second subtask. Clearly, set-up time reduction can have a positive effect. However, there are two factors which may have a negative effect on the performance. First of all, if the composite task fails, both subtasks need to be executed even if the error occurred in the second subtask. Secondly, both subtasks are executed even if the first subtask indicates that the case will be rejected. The processing time of a composite task joining tasks t_1 and t_2 is $pt(t_1) + (1 - sr(t_2))pt(t_2)$ and the failure probability of the composite task is $fp(t_1) + fp(t_2) - fp(t_1)fp(t_2)$. It is reasonable to assume that both tasks are executed because both subtasks are intertwined and only at the end it is possible to make some kind of decision. Since tasks are considered to be atomic, we also assume that a failure in one of the two parts leads to a new execution of the whole task. This is the assumption we also made for normal tasks: The execution of a task leads to success of the whole task (commit work) or a failure of the whole task (abort and rollback). This kind of failure semantics is common in workflow management systems. Typically,

the so-called ACID properties need to be preserved, i.e., tasks should be Atomic, Consistent, Isolated, and Durable [4].

Therefore we make the same assumption for composite tasks. The trade-off between the positive and negative effects of combining tasks is outlined in the following proposition.

Proposition 3. *Two subsequent tasks and sharing the same resource class should be combined if and only if* $pt(t_2)sr(t_2) > pt(t_1)fp(t_2) + pt(t_2)(fp(t_1) + rp(t_1) - fp(t_1)rp(t_1))$ *in order to obtain an optimal process with respect to resource utilization and maximal throughput.*

Proof. Consider two subsequent tasks t_1 and t_2. The average workload generated by these two tasks for a case waiting for task t_1 is $[pt(t_1)/[1 - fp(t_1)] + [(1 - rp(t_1))(pt(t_2))]/(1 - fp(t_2))$. By combining both tasks this workload equals $[pt(t_1) + (1 - sr(t_2))pt(t_2)]/[(1 - fp(t_1))(1 - fp(t_2)]$. By comparing both workloads we obtain the inequality which shows on the left-hand side the benefits of combining tasks $pt(t_2)sr(t_2)$, i.e., the setup time reduction, and on the right-hand side $pt(t_1)fp(t_2) + pt(t_2)(fp(t_1) + (rp(t_1) - fp(t_1)rp(t_1))$, the costs of extra work in case of a failure. □

It should be noted that the results presented in this section only hold for knock-out processes where there is no parallelism. Nevertheless, we can apply the result to the knock-out process shown in Fig. 1 (which contains parallelism). If we apply Proposition 3 to this process, it becomes clear that it was not a wise choice to combine task C and task D. The time saved by the setup reduction cannot compensate for the time lost by doing extra work in case of a failure. However, assuming that task B is executed before task A, Proposition 3 points out that it might be a good idea to combine task B and task A. By combining task B and task A, the resulting resource utilization is 96% of the original utilization. The following ratio gives the resource utilization in the combined situation (one composite task) relative to the original situation (two tasks):

$$\frac{pt(t_1) + pt(t_2)(1 - sr(t_2))}{pt(t_1)(1 - fp(t_2)) + pt(t_2)(1 - fp(t_1))(1 - rp(t_1))}$$

This ratio can be obtained by simply dividing the two workloads mentioned in the proof of Proposition 3.

The effect of combining tasks on the flow time is less clear. However, the following proposition shows the effect of the flow time given specific circumstances.

Proposition 4. *Consider two subsequent tasks and sharing a resource class containing one resource and having identical exponential service times (including failures). The tasks should be combined if and only if* $pt(t_2)sr(t_2) > pt(t_1)fp(t_2) + pt(t_2)(fp(t_1) + rp(t_1)\text{-}fp(t_1)rp(t_1))$ *in order to obtain an optimal process with respect to resource utilization, maximal throughput, and flow time.*

Proof. In Proposition 3, it was already shown that resource utilization is minimized and throughput is maximized by applying the criterion. In the Proof of Proposition 2, we showed that a workflow process can be described by a product form queuing network [16,18] if the routing is Markovian, the service times are exponential, and the average service time is the same for every class of client, i.e., the queuing network corresponds to the first class of BCMP networks [10]. Note that, in this section, we assume that there is no parallelism. $ut_{split} = [ar(2 - rp(t_1))pt(t_1)]/(1 - fp(t_1))$ is the resource utilization if both tasks are executed sequentially. (Note that we use the assumption that $pt(t)/[1 - fp(t_1)] = pt(t_2)/[1 - fp(t_2)]$, i.e., both tasks have identical average service times including failures.) $ut_{comb} = [ar(pt(t_1) + (1 - sr(t_2))\,pt(t_2))] / [(1 - fp(t_1))(1 - fp(t_2))]$ is the resource utilization if both tasks are combined. The average flow time if both tasks are executed sequentially is $\{[1/(1 - ut_{split})][pt(t_1)/(1 - fp(t_1))] + \{(1 - rp(t_1))[1/(1 - ut_{split})][(pt(t_2))/(1 - fp(t_2))]\} = ut_{split}/(ar(1 - ut_{split}))$. The average flow time if both tasks are combined is $[1/(1 - ut_{split})]\{(pt(t_1) + (1 - sr(t_2))pt(t_2))/[(1 - fp(t_1))(1 - fp(t_2))]\} = ut_{comb}/[ar(1 - ut_{comb})]$. Since $\{ut_{comb}/[ar(1 - ut_{comb})]\} < \{ut_{split}/[ar(1 - ut_{split})]\}$ if and only if $ut_{comb} < ut_{split}$, we conclude that combining only makes sense (with respect to the average flow time) if the resource utilization decreases. Therefore, tasks should be combined if and only if $pt(t_2)sr(t_2) > pt(t_1)fp(t_2) + pt(t_2)(fp(t_1)) + rp(t_1) - fp(t_1)rp(t_1))$. □

The crux of the proposition is the fact that the choice between one composite tasks and two separate tasks corresponds to the choice between one visit to an M/M/1 queue and two visits to an M/M/1 queue with about half the processing time. Note that it is vital that the average processing times of both tasks are identical. Otherwise there is no product form solution and it is not possible to apply the simple calculations. For this specific situation, the utilization rate of the resource is decisive. The higher the utilization rate, the longer the flow time. The effect of multiple resources on the trade-off between one composite task and multiple separate tasks is limited, because at any moment in time there is just one resource working on a specific case.

Heuristic 3. *If there are multiple resources per class but the tasks have about the same size, the rule used in Propositions 3 and 4 can be applied to obtain a minimal average flow time.*

Since task A and task B are of about the same size, it is wise to combine them in one composite task BA. If the condition mentioned in Heuristic 3 is not satisfied, the effect of combining tasks can have a more positive result than indicated by Proposition 4.

Heuristic 4. *If there is a relevant difference in size of the two tasks in Proposition 4, the rule should be applied carefully to obtain a minimal average flow time. In this case, splitting a composite task can have a negative effect on the average flow time even though the resulting utilization rate decreases.*

Consider the following (rather extreme) situation. A task of 99 min and a task of 1 min are combined into one composite task of 101 min, i.e., the set-up time reduction does not compensate the negative effects mentioned earlier ($101 > 99 + 1$). Clearly, the utilization rate of the resources involved increases. However, cases just have to queue once instead of twice. Note that cases are executed in FCFS order. If the effect of the increased utilization rate on the waiting time is limited, then the waiting time is almost divided by two by combining the tasks.

5. Putting tasks in parallel

The parallel execution of tasks may reduce the flow time considerably. Consider for example the construction of a building (e.g., building a house); only by doing things in parallel it is possible to build a house in a couple of weeks. Today's organizations are forced to minimize the flow time to reduce inventory or to improve the responsiveness to the customer. In fact, there are business processes where the average flow time is less than sum of all processing times. The latter cannot be achieved by just adding resources, it forces the organization to put tasks in parallel. Consider two parallel flows with flow times of respectively 3 weeks and 4 weeks. The total flow time will be 4 weeks. If the two flows are executed sequentially, the flow time will be 7 weeks. If there is no conditional routing (i.e., a fixed set of tasks needs to be executed), the flow time is minimized by putting as much tasks in parallel as possible without violating the precedence constraints. However, for knock-out processes it is more difficult. If two tasks are executed in parallel and one of the tasks returns NOK, the other task will be executed anyway and only after synchronization the case is rejected, i.e., processing completion is delayed until the moment of synchronization. This has a negative effect on the resource utilization and as a result can have a negative effect on the flow time.

Proposition 5. *Putting two subsequent tasks t_1 and t_2 in parallel increases the resource utilization and reduces the maximal throughput.*

Proof. Consider two subsequent tasks t_1 and t_2. The average workload generated by these two tasks for a case waiting for task t_1 is $\{pt(t_1)/[1 - fp(t_1)]\} + (1 - rp(t_1))[pt(t_2)/(1 - fp(t_2)]$. By putting both tasks in parallel the average workload equals $\{pt(t_1)/[1 - fp(t_1)]\} + \{pt(t_2)/[1 - fp(t_2)]\}$, i.e., for any positive value of $rp(t_1)$ the workload increases. □

In Proposition 5 it is assumed that both tasks have a positive rejection ratio. Therefore, it is easy to see that the amount of work is increased by putting tasks in parallel (rejected cases still consume capacity).

The following ratio shows the *increase* in resource utilization after putting both tasks in parallel:

$$\frac{pt(t_2)(1-fp(t_1))rp(t_1)}{pt(t_1)(1-fp(t_2))+pt(t_2)(1-fp(t_1))(1-rp(t_1))}$$

This ratio can be obtained by dividing the two workloads mentioned in the Proof of Proposition 5 and subtracting 1. Note that the ratio considers the increase in resource usage.

Suppose that there are two tasks with identical processing times and failure rates, and the reject probability of the first task is, then the increase in resource utilization is $x/(2-x)$.

The assumption that tasks in one parallel branch are executed while a task in another parallel branch already detected sufficient grounds for rejection is based on today's workflow management technology. There are several problems that make it difficult to stop all processing after a NOK is detected in one of the parallel branches. First of all, most workflow management systems do not have a mechanism for removing work items. Second, even if the workflow management system does have a mechanism for removing work items (e.g., Staffware [36]) it will not be able to stop a resource working on one of these work items thus leading to all kinds of synchronization problems. Moreover, if there are several parallel branches, say n, each containing a considerable number of tasks, say m, then there may be hundreds of potential states m^n thus making the withdrawal of work items very complicated and error-prone. The basic assumption used in today's workflow management systems is that parallel tasks are really independent of each other. Therefore, it is difficult, if not impossible, to model dependencies between parallel flows other than a straightforward synchronization.

Although the resource utilization increases by putting tasks in parallel, the organization may be willing to add resources to improve flow times. Therefore, it is interesting to investigate under what conditions the flow times are reduced. The positive effect of parallel routing is a more effective use of resources, i.e., the increased flexibility allows for a better deployment. In a sequential process, it is possible that one moment one resource (executing task T1) is overloaded and the other (executing task T2) is waiting for work, and the next moment the situation is reversed. In a parallel process this is not possible because the order in which tasks T1 and T2 need to be executed is not fixed. However, if both tasks need to be executed by resources of the same resource class, then it is not possible that cases are queuing for task T1 while there is sufficient free capacity for processing T2. In the latter case, both in the parallel process and in the sequential process, resources can work as long as there are cases waiting for either T1 or T2.

Heuristic 5. *The positive effect of two subsequent tasks in parallel is limited if both tasks require resources from the same resource class.*

Therefore, it is not wise to put tasks which are mapped onto one resource class in parallel. It will only increase the utilization and therefore also the average flow time. It is not possible to give general guidelines for deciding whether tasks should be executed in parallel or not. Parallel routing complicates analytical models. Synchronization results in the blocking of tasks, which complicates the formulation of simple and elegant results [16]. Therefore, we restrict ourselves to the rather weak statements in the following heuristic. In fact, to analyze the effect of parallel routing in more detail we advise to use the simulation toolbox ExSpect/KO.

Heuristic 6. *Putting subsequent tasks in parallel can only have a considerable positive effect if the following conditions are satisfied:*

- *Resources from different classes execute the tasks.*
- *The flow times of the parallel subprocesses are of the same order of magnitude.*
- *The reject probabilities are rather small.*
- *There is no overloading of any resource class as a result of putting tasks in parallel, i.e., the resulting utilization rates are acceptable.*
- *The time needed to synchronize is limited.*

If the flow times of parallel processes are unbalanced, there is little to gain by executing tasks in parallel. Consider two parallel flows with flow times of respectively one day and four weeks. The total flow time will be about 4 weeks. However, if the

two flows are executed sequentially the total flow time will also be about 4 weeks (worst case). In fact, if the flow of one day is highly selective, the average flow time will be reduced considerably if this one-day flow is executed first. If tasks are highly selective, it is always better to execute them sequentially. Clearly, parallel routing should only be considered if the organization can handle the extra work and the effort to synchronize flows is limited.

6. An approach to re-engineer knock-out processes

In the previous three sections we have seen three types of rules: (1) rules for ordering tasks in a knock-out process, (2) rules for combining and splitting tasks, and (3) rules for making knock-out processes more or less parallel. These steps can be executed in any order. However, for re-engineering knock-out processes we propose the following approach:

1. Determine the ordering of tasks according to Propositions 1 and 2, and Heuristics 1 and 2.
2. Combine tasks according to Propositions 3 and 4, and Heuristics 3 and 4.
3. Determine, using Proposition 5, and Heuristics 5 and 6, whether it makes sense to handle tasks in parallel.

Let us apply the approach to the example shown in Fig. 1. First, we determine the order of tasks. According to Proposition 1, task B has to be executed before A. According to Heuristic 1, C should be executed before D. Heuristic 2, taking into account the precedence constraints, shows that the optimal ordering of tasks is C.E.D.B.A. Process C.E.D.B.A is the sequential process with the smallest ratio (3.31). Then, we investigate the usefulness of composite tasks. In principle it is only possible to combine tasks C and D and tasks B and A. There are two reasons for not combining tasks C and D. First of all, task E needs to be executed between tasks C and D according to the ordering of tasks established in step 1. However, we may change the order if there is a lot to gain by joining these tasks. This is not the case, because there is a second reason for not combining tasks C and D. The time saved by the setup reduction cannot compensate the time lost by doing extra work in case of a failure. However, it is wise to combine task B and task A. By combining task B and task A, the resulting resource utilization decreases 4% compared to the situation where both tasks are not combined. Step 2 of our approach suggests that C.E.D.(B,A) is good starting point for investigating the effect of parallel routing (step 3). Since task C has to be executed before tasks E and D, only the flows E, D and (B.A) can be executed in parallel, e.g., C.[E,D,(B,A)], C.E.[D,(B,A)], C.D.[E,(B,A)], or C.(B,A).[E,D] are good candidates. However, task E is selective (20% is rejected) and the utilization of resource class Z is limited even if E is executed directly after task C. Therefore, we conclude that it is not wise to put E in parallel with other two flows (i.e., D and (B.A)). The only candidates for parallel execution that remain are D and (B.A). All conditions stated in Heuristic 6 are satisfied. Therefore, it may be a good idea to put D and (B.A) in parallel. In Section 7 we will use the simulation toolbox ExSpect/KO to investigate this in more detail. For the moment, we conclude that the 'optimal knock-out process' is either C.E.[D,(B,A)] or C.E.D.(B,A). Fig. 2 shows the knock-out process C.E.[D,(B,A)].

7. ExSpect/KO: a toolbox for analyzing knock-out processes

To support the approach presented in this paper, we have developed the simulation toolbox *ExSpect/KO*. This toolbox has been developed using *ExSpect 6.2* [8,9]. Before we present the functionality of the toolbox and apply it to the example, we briefly discuss the background of ExSpect and its applications in the domain of workflow management.

The author of this paper has been involved in the development of ExSpect for the last 10 years. Members of the department of Mathematics and Computing Science of Eindhoven University of Technology (including the author) have constructed the first versions of the software package ExSpect. In 1995 the development of ExSpect moved to the Dutch consultancy firm Bakkenist (150 consultants) where it is used as the standard modeling and analysis tool.

ExSpect is based on high-level Petri nets, offers a complete functional programming language, and supports complex simulation studies. Several authors have indicated that Petri nets constitute a good formal basis for workflow management [2–5,17]. In fact the leading BPR, ERP, and WFM tools on the Dutch workflow market are based on Petri nets. Because of this common basis it is possible to exchange workflow process definitions. Workflow processes modeled with the BPR tool *Protos* (Pallas Athena, [31]), the WFM system *COSA* (COSA Solutions/Ley, [14]), and the ERP system *Baan* (Baan Company, [7]) can be exported to ExSpect. This way it is possible to simulate the workflow processes specified in any of these tools. During the simulation of a workflow process with ExSpect, the workflow is animated and key performance indicators (e.g., average flow time and utilization rate) are measured. The fact that these tools are based on Petri nets also allows for advanced verification techniques. The tool *Woflan* (Eindhoven University of Technology) can be used to verify the correctness of workflow process definitions using state-of-the-art analysis techniques [2]. A detailed discussion of these tools is outside the scope of this paper. However, the link between the toolbox described in this paper and commercially leading tools illustrates the applicability of our approach. In fact, workflow designers using Protos, COSA or Baan/DEM can directly benefit from the results presented in this paper.

ExSpect/KO consists of two parts. First of all, there is a module (ko_sort.ex) to generate an optimal ordering of tasks using Heuristic 2. The ratio's mentioned in this paper were calculated with this module. Secondly, there is a second module (ko_sim.ex) which consists of building blocks for the simulation of knock-out processes. Fig. 3 shows a screenshot of ExSpect/KO. ExSpect/KO provides both animation and performance measures. The animation shows tasks as squares with one input arc and two output arcs (OK and NOK). AND-splits and AND-joins are represented by a sand timer. Also special building blocks for the arrival process, the resource manager, and measurement systems have been added. Figure 4 shows a selected part of the simulation dashboard. In total eleven measurements are reported, ranging from average flow times to average resource utilization. The simulation is divided into subruns and confidence intervals for the key performance indicators are generated automatically. For each subrun there are approximately 10,000 new arrivals and the average number of cases in the workflow is typically in the range 150–170 cases. Therefore, it is safe to assume that the subruns are mutually independent (this has been checked using the Von Neumann ratio) and the individual subrun results are assumed to be normally distributed (Law of large numbers). Based on these assumptions we calculate 90%-confidence intervals for flow times using the Student's *t*-distribution.

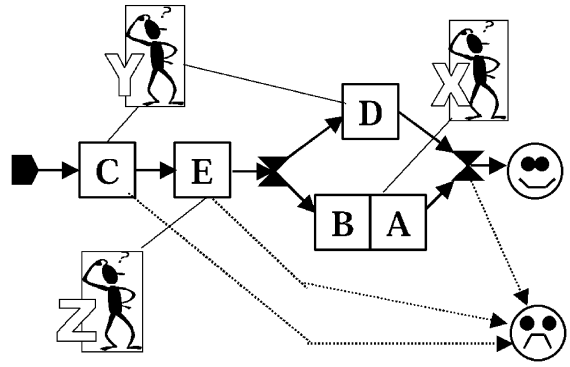
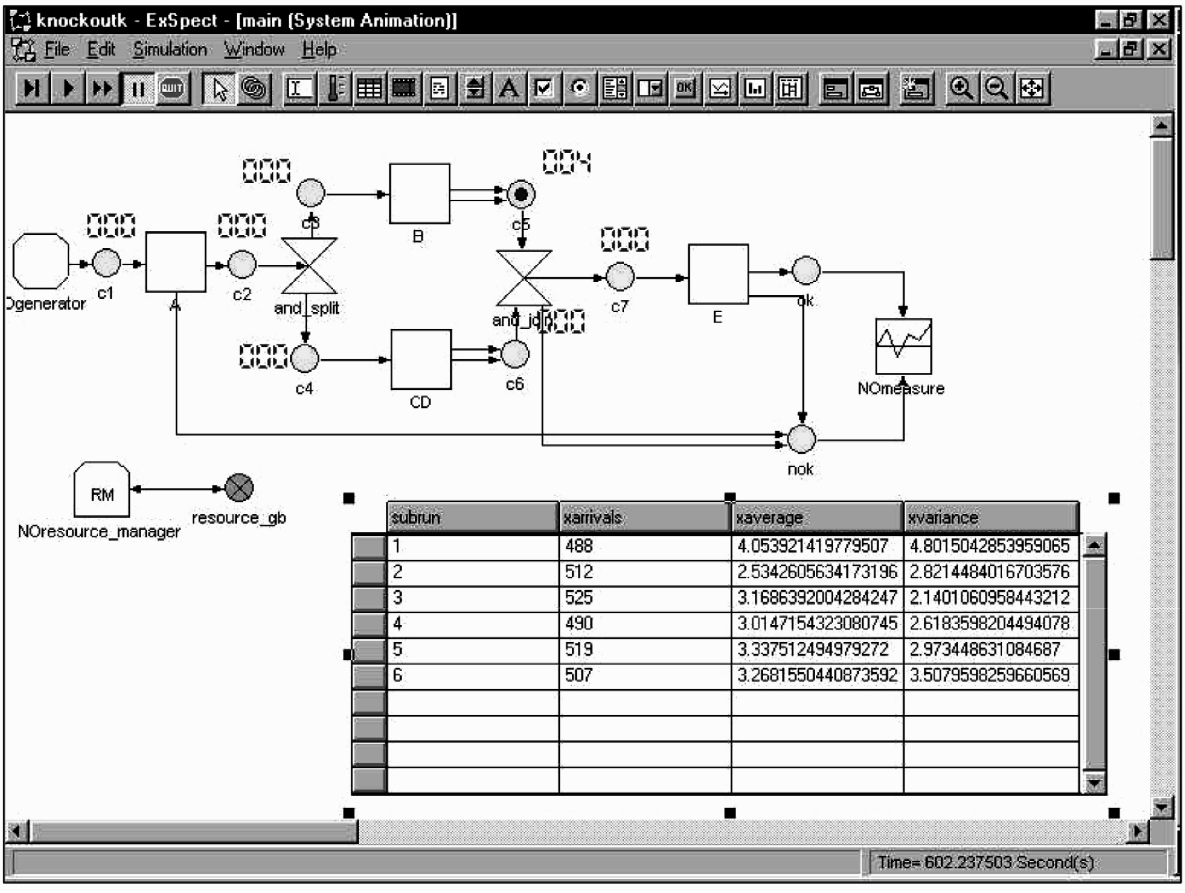


Fig. 3. An alternative knock-out process.

If we apply ExSpect/KO to the process A,[B,(C,D)].E (i.e., the process shown in Fig. 1), the following results are reported. Average flow time is more than 3 h. For a simulation run where 100,000 cases are processed the 90% confidence interval for the average flow time is [3.05,3.50], and the utilization rates of X, Y and Z are respectively 0.91 ([0.90,0.92], 90% confidence interval), 0.79 ([0.78,0.80]), and 0.31 ([0.30,0.32]). It takes about 10 min to simulate a run of 100,000 cases on a Pentium 200 Mhz computer.

For the process C.E.[D,(B,A)] (i.e., the process shown in Fig. 2), ExSpect/KO reports the following results. Average flow time is 1.59 h (100,000 cases, [1.57,1.61], 90% confidence interval) and the utilization rates of X, Y and Z are respectively 0.54 ([0.53,0.55], 90% confidence interval), 0.69 ([0.68,0.70]), and 0.47 ([0.46,0.48]).

If we apply the ExSpect/KO to the process C.E.D.(B,A), the following results are reported. Average flow time is 1.64 h (100,000 cases, [1.63,1.65],



subrun	xarrivals	xaverage	xvariance
1	488	4.053921419779507	4.8015042853959065
2	512	2.5342605634173196	2.8214484016703576
3	525	3.1686392004284247	2.1401060958443212
4	490	3.0147154323080745	2.6183598204494078
5	519	3.337512494979272	2.973448631084687
6	507	3.2681550440873592	3.5079598259660569

Fig. 4. A screenshot of ExSpect/KO.

90% confidence interval) and the utilization rates of X, Y and Z are respectively 0.47 ([0.46,0.48], 90% confidence interval), 0.69 ([0.68,0.70]), and 0.47 ([0.46,0.48]).

Based on these simulation results we conclude that both alternatives found by applying the step-wise approach presented in this paper improve the performance indeed. The average flow time is only half of the original situation and the workload is more balanced over the three resource classes. The step-wise approach could not answer the question whether alternative C.E.[D,(B,A)] is better than alternative C.E.D.(B,A) or vice-versa. Fortunately, we can use ExSpect/KO to make a more detailed analysis. As the simulation results show, C.E.[D,(B,A)] leads to significantly smaller flow times than C.E.D.(B,A), because D and (B,A) are executed in parallel (Fig. 5). However, the utilization rate of the resource class X increases about 6%. If we increase the arrival rate sufficiently, then one would expect alternative C.E.D.(B,A) to have a smaller average flow time than alternative CE[D,(B,A)] because of the increasing utilization. If we increase the arrival rate by 10%, then the average flow time for the process C.E.[D,(B,A)] is 1.73 h (110,000 cases, [1.69,1.76], 90% confidence interval) and the average flow time for the process C.E.D.(B,A) is 1.77 h (110,000 cases, [1.74,1.81], 90% confidence interval), i.e., the first alternative still seems to be slightly better. If we increase the arrival rate by 25%, then the average flow time for the process C.E.[D,(B,A)] is 2.17 h (125,000 cases, [2.11,2.24], 90% confidence interval) and the average flow time for the process C.E.D.(B,A) is 2.18 h (125,000 cases, [2.13,2.23], 90% confidence interval), i.e., both alternatives seem to be comparable. The fact that a considerable in-

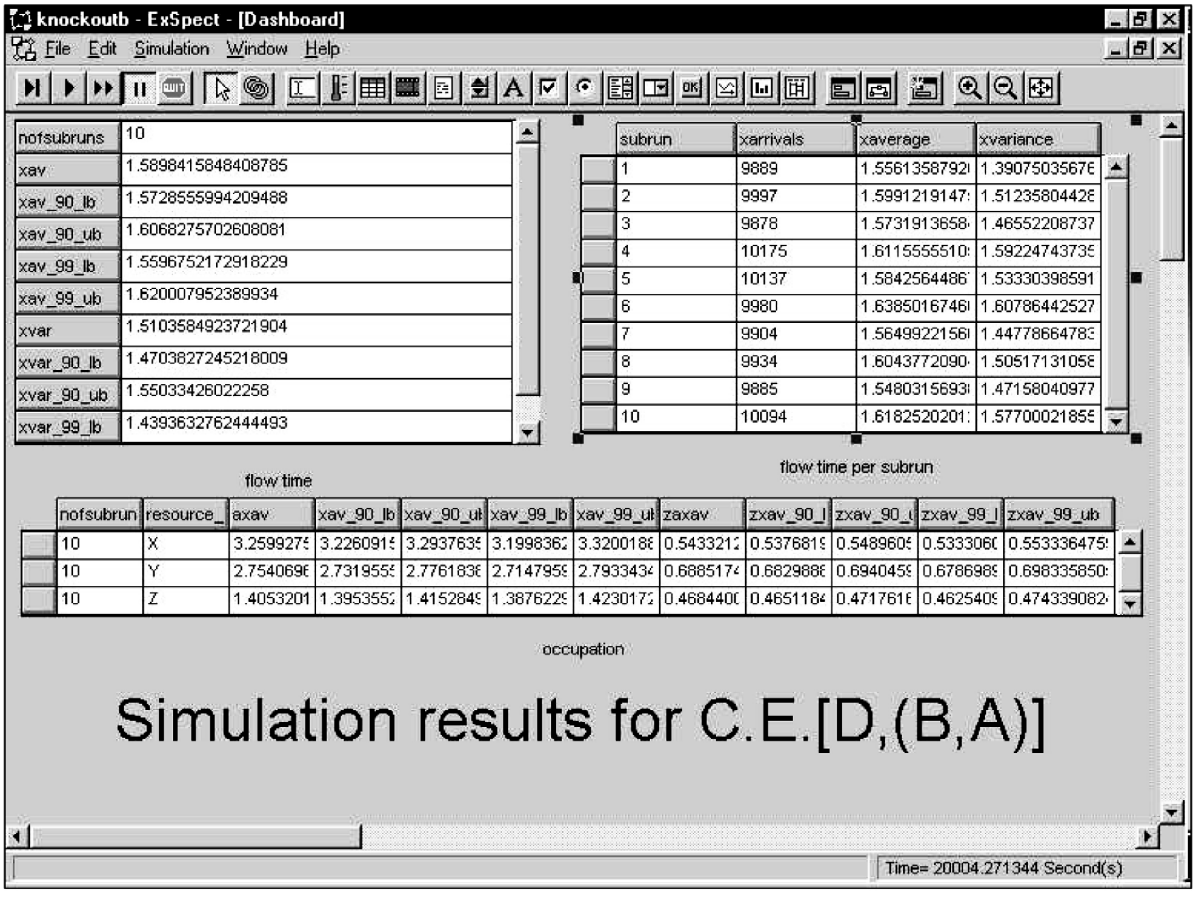


Fig. 5. Screenshot of the dashboard showing some of the results for the process shown in Fig. 3.

crease in arrivals does not lead to superiority of alternative C.E.D.(B,A) over C.E.[D,(B,A)] can be explained by the observation that resource class Y is the bottleneck and that the utilization rate of Y is the same for both alternatives. Note that some of the confidence interval overlap. By using longer simulation runs the length of these intervals can be reduced. However, the goal of this section was to illustrate the method and not to provide a detailed analysis of the example.

8. Conclusion

In this paper we investigated the class of knock-out processes. Knock-out processes are a particular type of workflow processes where the goal is to accept or reject cases by executing a specified set of tasks. These processes are called knock-out processes because if one tasks indicates that a case should be rejected, processing is stopped and the task is rejected immediately. For this particular class of processes we have given easy-to-apply rules for re-engineering. These rules are used to determine the ordering of tasks, to indicate useful combinations of tasks, and to find out which tasks should be executed in parallel. To support the approach, the simulation toolbox ExSpect/KO has been developed.

One might argue that only few processes encountered in practice have exactly the knock-out structure described in this paper. Consider for example the reviewing process of a paper for this journal: A reviewer can choose from five possible recommendations (ranging from unconditional acceptance to re-

jection) rather than two (OK or NOK) and not every negative review automatically leads to a rejection. Nevertheless, the re-engineering rules can be applied to selected parts of the reviewing process and the rules can be extended to incorporate more complex routing structures. Many processes encountered in practice contain routing patterns similar to the ones investigated in this paper. Workflow processes often contain subprocesses generating a decision based on a number of checks, e.g., handling insurance claims, processing job applications, and evaluating applications for building permits.

One should think of the reengineering rules as rules of thumb that can be used to generate attractive alternative designs. For a more detailed analysis of different designs one should resort to simulation [34] or standard queuing network analysis [16,18]. The toolbox ExSpect/KO is tailored towards the simulation of knock-out processes. However, ExSpect also allows for the analysis of arbitrary workflow processes using the more general, but less supportive, toolbox described in Ref. [13]. Moreover, it is possible to automatically download and analyze workflow designs from several commercial workflow products including COSA [14] and Protos [31]. See Ref. [9] for more details or to download a demo version of the ExSpect software.

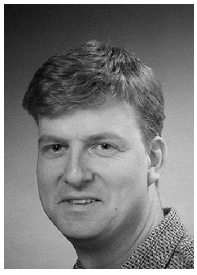
Acknowledgements

Part of this work was done at AIFB (University of Karlsruhe, Germany) and CTRG (University of Colorado, USA) during a sabbatical leave.

References

[1] W.M.P. van der Aalst, K.M. van Hee, Business Process Redesign: a Petri-net-based approach, Computers in Industry 29 (1–2) (1996) 15–26.
[2] W.M.P. van der Aalst, Verification of workflow nets, in: P. Azema, G. Balbo (Eds.), Application and Theory of Petri Nets 1997, Lecture Notes in Computer Science, vol. 1248, Springer-Verlag, Berlin, 1997, pp. 407–426.
[3] W.M.P. van der Aalst, Chapter 10: Three Good reasons for Using a Petri-net-based Workflow Management System, in: T. Wakayama (Ed.), Information and Process Integration in Enterprises: Rethinking documents, The Kluwer International Series in Engineering and Computer Science, Kluwer Academic Publishers, Norwell, 1998, pp. 161–182.
[4] W.M.P. van der Aalst et al., The Application of Petri Nets to Workflow Management, The Journal of Circuits, Systems and Computers 8 (1) (1998) 21–66.
[5] N.R. Adam, V. Atluri, W. Huang, Modeling and analysis of workflows using Petri Nets, Journal of Intelligent Information Systems 10 (2) (1998) 131–158.
[6] R. Ardhaldjian, M. Fahner, Using simulation in the business process reengineering effort, Industrial Engineering (1994) 60–61, 1994.
[7] Baan Company. BaanSeries/Dynamic Enterprise Modeler. http://www.baan.com/, 1999
[8] Bakkenist Management Consultants. ExSpect 6.2 User Manual. Amsterdam, The Netherlands, 1998.
[9] Bakkenist Management Consultants. ExSpect WWW page. http://www.exspect.com/, 1999
[10] F. Baskett, K.M. Chandy, R.R. Muntz, F.G. Palacios, Open, closed and mixed networks of queues with different classes of customers, Journal of the Association of Computing Machinery 22 (2) (1975) 248–260.
[11] J.A. Buzacott, Commonalities in reengineered business processes: models and Issues, Management Science 42 (5) (1996) 68–782.
[12] J.A. Buzacott, D.D. Yao, On queueing networks of flexible manufacturing systems, Queueing Systems (1986) 29–66.
[13] H.A.C. Cornelissen. Construction of a Workflow Management Toolbox in ExSpect. Masters Thesis. Eindhoven University of Technology, 1995.
[14] COSA Solutions/Ley GmbH. COSA 2.0.http://www.cosa.de/, 1999
[15] H. Davenport, Process Innovation: Reengineering Work Through Information Technology, Harvard Business School Press, Boston, 1993.
[16] N.M. van Dijk, Queueing Networks and Product Forms: A Systems Appoach, Wiley, Chichester, 1993.
[17] C.A. Ellis, G.J. Nutt, Modelling and enactment of workflow systems, in: M. Ajmone Marsan (Ed.), Application and Theory of Petri Nets 1993, Lecture Notes in Computer Science, vol. 691, Springer-Verlag, Berlin, 1993, pp. 1–16.
[18] E. Gelenbe, G. Pujolle, Introduction to Queuing Networks, Wiley, Chichester, 1998.
[19] D. Georgakopoulos, M. Hornick, A. Sheth, An overview of workflow management: from process modeling to workflow automation infrastructure, Distributed and Parallel Databases 3 (1995) 119–153.
[20] M. Hammer. Reengineering work: Don't automate, Obliterate. Harvard Business review, pages 104–112, July/August 1990.
[21] M. Hammer, J. Champy, Reengineering the corporation, Nicolas Brealey Publishing, London, 1993.
[22] G.A. Hansen, Automated Business Process Reengineering: Using the Power of Visual Simulation Strategies to Improve Performance and Profit, Prentice-Hall, Englewood Cliffs, 1997.
[23] S. Jablonski, C. Bussler, Workflow Management: Modeling Concepts, Architecture, and Implementation, International Thomson Computer Press, London, 1996.
[24] H.J. Johansson, P. McHugh, A.J. Pendlebury, W.A. Wheeler,

Business Process Reengineering: Breakpoint Strategies for Market Dominance, Wiley, New York, 1993.
[25] S. Khoshanfian, Introduction to Groupware, Workflow, and Workgroup Computing, Wiley, New York, 1995.
[26] L. Kleinrock, Queueing systems, Vol. 1: Theory, Wiley-Interscience, London, 1975.
[27] T.M. Koulopoulos, The Workflow Imperative, Van Nostrand Reinhold, New York, 1995.
[28] P. Lawrence, Workflow handbook 1997,Workflow Management Coalition, Wiley, New York, 1997.
[29] R.L. Manganelli, M.K. Klein, The Reengineering Handbook: A Step-by-step Guide to Business Transformation, Amacom, New York, 1996.
[30] D. Morris, J. Brandon, Reengineering Your Business, McGraw-Hill, New York, 1993.
[31] Pallas Athena. Protos 4.0.http://www.pallas.nl/, 1999.
[32] G. Poyssick, S. Hannaford, Workflow Reengineering, Adobe Press, Mountain View, CA, 1996.
[33] H.A. Reijers, W.M.P. van der Aalst, Short-term simulation: bridging the gap between operational control and strategic decision making, in: M.H. Hamza (Ed.), Proceedings of the IASTED International Conference on Modeling and Simulation, IASTED/Acta Press, Anaheim, USA, 1999, pp. 417–421.
[34] S.M. Ross, A Course in Simulation, Collier Macmillan, London, 1990.
[35] M. Schwartz, Telecommunication Networks, Addison Wesley, Reading, MA, 1987.
[36] Staffware . Staffware 97.http://www.staffware.com/, 1999
[37] WFMC. Workflow Management Coalition Terminology and Glossary (WFMC-TC-1011). Technical report, Workflow Management Coalition, Brussels, 1996.
[38] N. Wirth, What can we do about the unnecessary diversity of notation for syntactic definitions? Communications of the ACM 20 (11) (1977) 822–823.



Wil van der Aalst is a full professor of Information Systems and head of the Department of Information and Technology of the Faculty of Technology Management of Eindhoven University of Technology. He is also a part-time professor at the Computing Science department of the same university and has been working as a part-time consultant for Bakkenist. Wil van der Aalst research interests are information systems, simulation, Petri nets, process models, workflow management systems, verification techniques, enterprise resource planning systems, computer supported cooperative work, and inter-organizational business processes.