

Optimistic Distributed Execution of Business Process Models *

Alois Ferscha
Institut für Angewandte Informatik
Universität Wien
Lenaugasse 2/8, A-1080 Vienna, AUSTRIA
Email: ferscha@ani.univie.ac.at

Abstract

For the modeling of large and complex systems of business processes, a flow oriented, graphical modeling framework based on Petri nets has emerged, tapping the potentials of a qualitative and a quantitative analysis based on one and the same model. For the quantitative analysis of business process models (BPMs) representing realistically sized enterprise organizations, traditional evaluation techniques (like discrete event simulation) tend to become practically intractable. To be able to cope with very complex models, therefore, we have developed a distributed execution mechanism based on the Time Warp distributed simulation protocol. A corresponding software tool was implemented based on the MPI communication library, thus portable to almost any distributed or parallel computing platform.

In case studies performed on a 134 node Meiko CS-2 multiprocessor investigating real and hypothetical business organizations, this work demonstrates that parallel/distributed simulation techniques make the execution of very large models feasible: in situations where the simulation model has reached a complexity prohibitive to an execution on a single processor computer (e.g. due to memory constraints), the decomposition into smaller submodels to be executed on a parallel processor or a network of workstations remains the only means to get the simulation done. As such, a whole new class of (complex) BPM simulations becomes practically tractable, and traditional simulations can be accelerated dramatically. As an example, a BPM of a document flow system comprising 64 offices with an average of 1000 documents per office gains a 250 to 300 fold acceleration of the overall execution speed using 32 processors of the CS-2.

* This work was partially supported by the Oesterreichische Nationalbank under grant No. 5069, and the Human Capital and Mobility program of the EU under grant CHRX-CT94-0452 (MATCH).

1 Introduction

Business Process Reengineering has emerged as a modeling and analysis methodology for making the organization of an enterprise more efficient. For the purpose of reengineering, a business process *model* (BPM) has to be developed, the evaluation of which in almost any case demands simulation. Even for enterprises of moderate size, the BPMs tend to become prohibitively complex, precluding a sequential simulation as an efficient evaluation method. Distributed simulation, or generally distributed model execution, is demanded to cope with this complexity.

Besides modeling techniques involving high level data flow diagrams in the process oriented enterprise view, entity relationship diagrams in the data oriented enterprise view and state transition diagrams in the time and control view of an enterprise, a *flow oriented* enterprise view has emerged [1] [2] [3]. Within this modeling approach, a (*business*) *process* is described as a set of partially ordered steps to reach a (business) goal. A component of a process is called a *process element*, representing an atomic action with no internal substructures. An *agent* is an actor (human or machine) who performs one or more process elements. A coherent set of process elements to be assigned to an agent as a unit of functional responsibility is also called a *role*, and the a product created or modified by the enactment of a processes element is referred to as *work*. A *business process model* (BPM) finally is an abstract description of business processes constituted by their respective process elements, together with their assignment to agents. The assignment describes the dependencies, coordination and interaction among agents, the most natural way of interaction being the flow of work among agents. For this reason, we also refer to BPMs as “Workflow Models” [4]. Like the assignment of process elements to the responsibility of agents, a BPM must also appropriately describe the input and output relation of work for every

process element. This necessitates the preservation of causal (flow) relationships defined in processes, but also the work scheduling and management strategy applied by an agent to execute the assigned process elements. This is essential especially for agents acting in several roles (involved in more than one process). We have successfully used the Petri net (PN) formalism [5] for the purpose of expressing both *qualitative* and *quantitative* properties of systems of business processes. The former has been investigated thoroughly in the context of Petri nets, like e.g. in [6] and [7]. For the latter, the *quantitative* analysis, temporal system behavior becomes an issue of importance. For a BPM to be adequate with respect to causal dependencies of processes as determined by their temporal appearance and duration, it must reflect delays and durations of process element executions. Consequently, the modeling formalism has to provide sufficient expressiveness to characterize the time dynamics of the system. The modeling power of *timed* Petri nets meets those requirements to an outstanding extent [1], as we have practically demonstrated in [2]. For the quantitative analysis of timed Petri net BPMs we have preliminarily chosen discrete event simulation, but encountered that the complexity of business process systems of realistic size tends to become overwhelming, making traditional sequential simulation techniques practically intractable.

This work presents the use of distributed simulation methods to accelerate the BPM evaluation. Our approach is to decompose a BPM based on the structural properties obtained from a preanalysis of the underlying timed Petri net. Submodels are executed by *logical processes* allocated to different nodes of a multiprocessor [8], employing the Time Warp parallel simulation protocol for event synchronization.

2 Modeling Business Process Systems

The use of the Petri net formalism to support a flow oriented system view of BPMs has been motivated in early work on office information flow modeling [9] and the scheduling of office procedures [10]. Qualitative and quantitative analysis of Petri net based BPMs was exercised in [2]. Recently, further reasons for the use of Petri nets for workflow modeling have been given in [3]. Since this work is focused on the acceleration of the simulation and/or execution of BPMs in distributed and parallel computing environments, we are concerned here mainly with the integration of the static description of BPMs and their behavioral and temporal dynamics, when modeled with Petri nets. In a natural correspondence among PNs and business

process concepts, the static structure the business process topology is expressed in a PN graph, whereas the behavioral aspect is encoded in the initial marking and the transition firing rules of the PN.

The main PN based BP modeling concepts can more formally be summarized as follows: A BPM is a PN $BPM = (P, T, F, W, \mu^{(0)}, \tau, \mathcal{P})$, where P , the set of places, corresponds to “work deposits”, i.e. stores for pieces of work represented by tokens. The set of transitions $T = (\pi_1, \pi_2, \dots, t_1, t_2, \dots)$ stands for *process elements* π_i , i.e. abstract, atomic processes to be conducted upon invocation (enabling). The transitions $t_i \in T$ are used for defining the precedence of the process elements π_i with respect to their invocation, i.e. the “flow” of work. Transitions $\pi_i \in T$ have assigned enabling delays $\tau(\pi_i)$, for example $\tau(\pi_i) \sim \exp(\lambda_i)$ if the duration of the process execution is modeled by an exponential random variate characterized by the parameter λ_i of the exponential distribution. The enabling delay of process elements can be expressed by arbitrarily complex expressions $\tau(\pi_i)$, yielding either a deterministic or stochastic timing behavior. Transitions $t_i \in T$ do not consume time for their enabling ($\tau(t_i) = \tau_i = 0.0$), but can have assigned a probability $\mathcal{P}(t_i)$ to model random selection of tokens for firing (*random switches*). (Conveniently we shall also refer to transitions $t_i \in T$ as “process elements”.) *Work*, is seen as a product created or modified by the enactment of a process element, i.e. the firing of a transition. A process element may require a *resource* or a *set of resources* for their execution, the execution itself may or may not consume time. F , the flow relation, models the flow of work among process elements $\pi_i \in T$, and $W : F \mapsto \mathbb{N}^+$ is the arc weight function in the usual sense [5]. $\mu^{(0)}$ is the initial assignment of work to work deposits $p_i \in P$.

Analogously to the rules defining the dynamic behavior of PNs we have:

- (i) (*process element enabling rule*) A process element $\pi \in T$ is *enabled* in some marking μ at time \mathcal{T} , iff each of its input deposits holds a “sufficient” amount of work, i.e. iff $\forall p \in I(\pi)$, $\mu(p) \geq w((p, \pi))$ in μ . ($I(\pi)$ denoting the input function.) $E_{\mathcal{T}}(\mu)$ is the set of all transitions enabled in μ at time \mathcal{T} .
- (ii) (*process element selection rule*) Let $E_{\mathcal{T}}(\mu)$ contain only process elements π_i with $\tau(\pi_i) = 0.0$. $\pi_i, \pi_j \in T$ are in *conflict* at time \mathcal{T} ($\pi_i C_{\mathcal{T}} \pi_j$), iff at time \mathcal{T} the actual marking is μ s.t. $\pi_i, \pi_j \in E_{\mathcal{T}}(\mu)$, and $\mu \xrightarrow{\pi_i} \mu'$ might cause that $\pi_j \notin E_{\mathcal{T}}(\mu')$. If ($\pi_i C_{\mathcal{T}} \pi_j$), then π_i is selected for execution

with probability:

$$\mathcal{P}_{\pi_i} = \frac{\mathcal{P}(\pi_i)}{\sum_{j|(\pi_i, C_T, \pi_j)} \mathcal{P}(\pi_j)}.$$

The selected process element *must* execute.

In case $E_T(\mu)$ contains process elements with $\tau(\pi_i) = 0.0$ and $\tau(\pi_j) > 0.0$, the ones with $\tau(\pi_i) = 0.0$ are always selected prior to the other ones.

In case $E_T(\mu)$ contains *only* process elements with $\tau(\pi_i) > 0.0$, the remaining enabling time $RET(\pi_i)$ is computed from $\tau(\pi_i)$, the total enabling time, minus the time expired since the enabling instant of π_i . The process element $\pi | \min_i RET(\pi_i)$ is selected for execution.

- (iii) (*process element execution rule*) When a process element $\pi \in T$ executes in μ , it creates μ' ($\mu \xrightarrow{\pi} \mu'$) by removing a certain amount of work from its input deposits and creating a certain amount of work into its output deposits: $\forall p \in I(\pi) \cup O(\pi) \quad \mu'(p) = \mu(p) - w((p, \pi)) + w((\pi, p))$.

A *business process*, or equivalently a *process*, finally appears as a set of partially ordered process elements, described as a spatial region of the PN graph.

3 Distributed Execution of BPMs

An *event driven* simulated execution of BPMs repeatedly processes the occurrence of process elements (events) in simulated time (often called “*virtual time*”, VT) by maintaining a time ordered *event list* (EVL) holding timestamped events scheduled to occur in the future, a (global) *clock* indicating the current time and *state variables* $S = (s_1, s_2, \dots, s_n)$ defining the current state of the system. A *simulation engine* (SE) drives the simulation by continuously taking the first event out of the event list (i.e. the one with the lowest timestamp), simulating the effect of the event by changing the state variables and/or scheduling new events in EVL – possibly also removing obsolete events. This is performed until some pre-defined endtime is reached, or there are no further events to occur.

Certainly, this sequential strategy of simulated execution can become aggressive in the consumption of computational resources, and the use of distributed execution facilities is advised for large and complex BPMs. In order to gain speedup of elapsed simulation time, however, from a parallel execution, there needs to be parallelism among the occurrences of process elements (events) in the BPM. Generally, for a distributed execution of BPMs we are interested in the

possibility of executing events of a spatially decomposed BPM simultaneously in different processor, i.e. to exploit potential model parallelism between process elements that are located in different partitions of the BPM (see [11] for model parallelism and decomposition issues). Naturally, process elements that do not “share” input places and can become enabled at the same instant of time are potentially parallel and can be executed concurrently, but only if it is guaranteed, that executing one will not disable the other. let $I(\pi_i)$ be the set of input places to p_i , then two process elements $\pi_i, \pi_j \in T$ are called *parallel* ($\pi_i || \pi_j$), iff

- (i) $I(\pi_i) \cap I(\pi_j) = \emptyset$
- (ii) $\exists \mu$ s.t. $\pi_i, \pi_j \in E(\mu)$
- (iii) given $\pi_i, \pi_j \in E(\mu)$, then $\mu \xrightarrow{\pi_i} \mu'$ cannot cause that $\pi_j \notin E(\mu')$.

Necessary and sufficient conditions for the detection of model parallelism in BPMs can be derived using the computation of P-invariants and T-invariants of the underlying PN, and has been studied in [12]. For the rest of this paper, we shall study BPMs with sufficient model parallelism, and investigate the performance effect of the degree of model parallelism.

An event driven, *distributed* simulated execution of BPMs commonly involves the division of the BPM into a set of communicating *logical processes* (LPs), exploiting the inherent parallelism among the respective model components. A *logical process simulation* (LP simulation) of BPMs is thus viewed as the cooperation of an arrangement of interacting LPs, each of them simulating a subspace of the space-time which we will call an event structure *region*. A *communication system* (CS) provides the possibility to LPs to exchange local data, but also to synchronize local activities. In every LP_i a simulation engine SE_i executes event occurrences *local* to region R_i , (and potentially generates *remote* event occurrences) thus progressing a *local clock* (local virtual time, LVT) in each LP_i . Since each LP_i (SE_i) has access only to a statically partitioned *subset of the state variables* $S_i \subset S$, disjoint to state variables assigned to other LPs, synchronization of local system states is necessary: a *communication interface* CI_i attached to the SE takes care for the propagation of effects causal to events to be simulated by remote LPs, and the proper inclusion of causal effects to the local simulation as produced by remote LPs. Therefore, Two kinds of events are processed in each LP_i : *internal events* which have causal impact only to $S_i \subset S$, and *external events* also affect $S_j \subset S$ ($i \neq j$) the local states of other LPs.

Several approaches for the distributed execution of discrete event systems have appeared in the field of *parallel and distributed simulation* [13], establishing two schools of thought: the conservative Chandy/Misra/Bryant (CMB) approach, and the optimistic Time Warp (TW) based parallel simulation protocols. Both schools have established LP synchronization schemes that operate in a *causally correct* way, i.e. LPs that despite the asynchronous execution of their components would generate consistent simulation sample paths. CMB protocols execute events in each LP that occur in the respective submodel in non-decreasing order of their occurrence timestamp while strictly preventing the possibility of any event causality violation across LPs. The primary motivation for TW protocols is that events which occur in different LPs – irrespective of their occurrence time stamp – might not affect one another, thus giving raise to their parallel, out-of-timestamp-order execution. No general rule of superiority of the two strategies can be formulated [8], but since the timing in BPMs is typically of stochastic nature, the CMB approach with its policy of blocking a SE even if the probability for causality violation is vanishingly small appears overly pessimistic. In the sequel of this work we develop a TW based distributed execution environment for BPMs.

3.1 TW for Distributed BPM Execution

A distributed execution engine for BPMs under TW can be considered to consist of two principal components: (i) a logical process based *simulation engine* which ensures the causal consistency of the simulation by implementing a causality violation detection and recovery policy called *rollback*, a means for saving and restoring simulation states (as demanded by the rollback procedure), and a mechanism for committing states obtained in the various LPs by progressing *global virtual time* (GVT), and (ii) the business process *model* itself, encompassing a detailed description of event dependencies and their occurrence. A clear separation of the simulation engine and the simulation model preserves the possibility to link various model implementations to the same simulation engine, and vice versa.

The implementation of a TW based LP is outlined in Figure 1. During initialization (s1) the assigned submodel of the BPM description file is read, and model initialization produces a list of initial internal events *ie* which are inserted into *EVL*. The resulting initial state is subsequently logged on the state stack. The main simulation loop is then executed until GVT reaches the value *ENDTIME* (s2). Incoming messages are read from the LP's input buffer (s3), and their time

```

LP(i) {
s1  iel=initial(BPM);
    while(ie=next_ie(iel)) chronological_insert(ie,EVL);
    log_new_state();
s2  while (GVT < ENDTIME) {
s3    while (m=read_next_input_buffer_message()) {
s4      if (ts(m) < LVT) {
          /* rollback – process straggler */
s4.1      dual=dual_exists(m,IQ);
s4.2      if ((positive(m)&& !dual) ||
            (negative(m)&& dual)) {
s4.3          LVT=restore_state_before(ts(m));
s4.4          ant_evl=generate_antimsg(LVT);
s4.5          while(ee=next_ee(ant_evl))
              chronological_insert(ee,OQ);
            }
          }
s5      if (!remove_dual(m,IQ)) insert(m,IQ);
    }
s6    advance_GVT(); fossil_collection();
s7    if (memory_used > MEMORY_LIMIT) continue;
s8    e = get_first_EVL_or_IQ();
    if (e) {
          /* process event e */
s8.1      new_ev, pre_evl, ext_evl = modified_by_e(e);
s8.2      while(ie=next_internal_e(new_evl))
          chronological_insert(ie,EVL);
          while(ie=next_preempted_int_e(pre_evl))
          remove_event(ie, EVL);
s8.3      LVT = ts(e);log_new_state();
s8.4      while(ee=next_external_e(ext_evl))
s8.5      if (!dual_update(ee,OQ))
          chronological_insert(ee,OQ);
    }
s9    fill_OB(OQ);
    send_out_contents(OB);
  }
s10 analyse_stack();
}

```

Figure 1: Time Warp Distributed Execution Engine stamps are checked against the current LVT (s4). In the case of a causality violation, the rollback mechanism is invoked to restore the first consistent state prior to the time stamp of the straggler message (s4.1 – s4.3), and antimessages generated in the course of the rollback (s4.4) are inserted into the output queue OQ (s4.5). As a final action of the input processing part, each incoming message is inserted into the input queue IQ, or annihilated if it finds its dual counterpart in the input queue (s5).

In step (s6) the algorithm tries to progress GVT, and if successful, cleans up fossils from the state stack. GVT is calculated in a distributed way, involving the sending of packets containing a vector GVT estimates from all LPs: If a GVT calculation packet has been received in LP_i during the input phase, then the **advance_GVT**() code segment is executed. Using the information contained in the packet, LP_i calculates a new GVT estimate, updates its own information in the GVT packet and forwards it to its successor.

(s7) checks if sufficient memory (specified in MEMORY_LIMIT) is available to perform the local simulation of the next event. If not, the simulator loops back to the input section (s3) to await the arrival of further messages or GVT calculation packets until sufficient memory is freed through the occurrence of a rollback (arrival of a straggler message) or fossil collection (arrival of a GVT calculation packet). In the case that no events are scheduled for local simulation in the IQ or EVL (i.e. the partition has become depleted of units of work, i.e. tokens), the simulator also loops back to the input section (s3). The occurrence of the next scheduled local event is simulated by modifying the LPs state variables accordingly (s8.1). **modified_by_e**(e) returns three lists of events to the simulation engine: (i) a list of the new internal events resulting from the occurrence of the event in the model which are to be scheduled for future simulation, (ii) a list of previously scheduled events which have now been pre-empted by the occurrence of the event and (iii) a list of new external events (message arrivals in other partitions) which must be sent to the respective processors. The state of the simulator is updated to reflect the occurrence of the event by inserting the internal events into the EVL and removing the pre-empted events. External events are inserted into the output queue (OQ) or alternately annihilated if a dual message is present in the OQ (lazy cancellation) (s8.5). In (s8.4) the current state of the execution is saved by copying the EVL to the state stack as well as the state variables used by the model.

Finally, messages stored in the OQ with time stamps less or equal to the current LVT are moved to the output buffer and sent to the respective LPs in (s9) and control loops back to the input phase. The current simulation run terminates when GVT reaches ENDTIME.

4 A Small Business Process Model

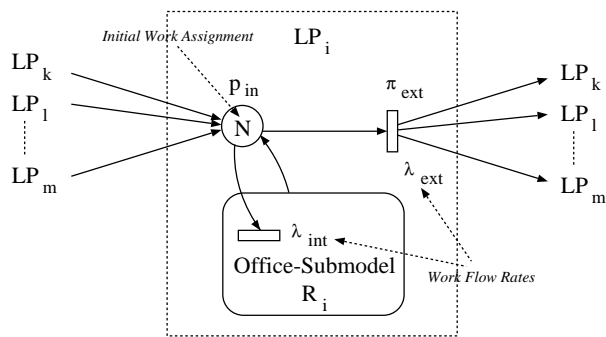


Figure 2: A simple BPM of a Document Flow System

To demonstrate the potential performance gain of a distributed BPM execution on one hand, and the relative performance of TW on the other hand, we study the BPM in Figure 2 developed for the document flow in a large office system. In the modeling process, each office was represented by a BPM “region”, containing a work deposit P_{in} for incoming documents (work), and a process element π_{ext} propagating documents to other offices (regions). The initial amount of work in P_{in} represents the number of documents n initially in the In-box of office (region) R_i . In the experiments we study an office system consisting of 64 offices with the same business process structure, connected to each other by the arcs originating from respective π_{ext} processes, and discharging into P_{in} deposits, i.e. the execution of π_{ext} in region R_i causes a document to be transferred to the successor region R_j . If R_i and R_j are handled by LPs assigned to different processors, the TW protocol generates and sends a message for each document flowing from R_i to R_j , whereas documents flowing within one region do not invoke any message sending.

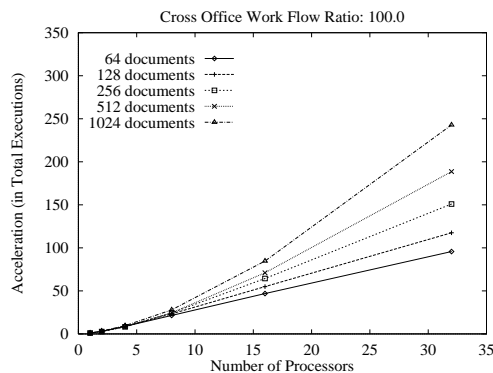


Figure 3: Acceleration of the Distributed Execution of the Document Flow BPM on the CS-2

The performance improvements of a distributed execution of the document flow BPM in Figure 2 using the TW based simulation engine on the Meiko CS-2 are presented in Figure 3. The plot reflects a set of experiments involving 2 to 32 processors for a 64 office document flow BPM at varying initial work assignments ($D = 2^6, \dots, 2^{10}$ documents initially in each office). The cross office work flow ratio among offices (i.e. the ratio $\lambda_{int} : \lambda_{ext}$) is kept at 100, i.e. an office on average processes 100 documents before passing one on to another office. The plot explains performance in terms of “acceleration”, which is the total number of committed process element executions per processor per second, and shows that a 250 fold accelera-

64 Initial Documents, 1 per Office

$\frac{\lambda_{int}}{\lambda_{ext}}$	1 proc	2 procs	4 procs	8 procs	16 procs	32 procs
10000	393.0	1451.0	4941.7	14412.0	27798.3	51082.7
Accel.	1.0	3.7	12.6	36.7	70.7	* 130.0
1000	396.0	1439.7	4940.0	14861.7	28592.7	46987.7
Accel.	1.0	3.6	12.5	37.5	* 72.2	118.7
100	391.3	1456.7	5285.7	15902.0	27003.7	36910.0
Accel.	1.0	3.7	13.4	* 40.6	69.0	94.3
10	419.3	1541.3	5603.7	15825.7	20757.0	31600.7
Accel.	1.0	3.7	* 13.5	37.7	49.5	75.4
1	406.3	1519.7	5120.7	13311.0	19255.0	26290.0
Accel.	1.0	* 3.7	12.6	32.8	47.4	64.7
0.1	395.0	1458.0	4703.7	11986.0	18303.0	23883.0
Accel.	1.0	3.7	11.9	30.3	46.3	60.5

Figure 4: Impact of Cross Office Document Flows

tion is possible with 32 processors for a heavily loaded (1024 initial documents) BPM. This means, that for every successful execution of a process element in one and the same BPM if assigned to a single processor, 250 executions could be executed in the same time frame if decomposed and simulated on 32 processors. This effect mainly results from the decomposition of the simulation model into smaller submodels which reduces memory access overhead at quadratic order. (The cost for search and insert operations in EVL is $O(\log_2(\Pi/N))$, for system calls to allocate memory it is $O(\Pi/N)$, where Π is the number of documents in the system.)

Generally, the highest accelerations are achieved in cases with a large document (token) count whereas the lowest accelerations occur in cases with small document populations: models with a large number of tokens exhibit a higher level of *model parallelism* and thus a larger number of possible *concurrent process execution*. This is a desirable trait of the TW protocol.

The impact of varying cross office work flow rates among offices on TW performance is an important issue. Note that according to the definition of the *process element selection rule (ii)*, it is possible to simulate the behavior of BPM regions with varying degrees of “locality” by appropriately adjusting the ratio $\lambda_{int} : \lambda_{ext}$ (which we call Internal/External Event Ratio): a large ratio simulates a model where the majority of process executions affect only processes within the LP’s office-submodel, whereas a small ratio causes frequent document flows out of the submodel, yielding increased communication overheads for TW. Figure 4 explains a tendency that with higher internal/external event ratios higher accelerations are possible: In the case of 2 processors, the highest acceleration occurs at a ratio of 1, for 4 processors at 10, for 8 at 100, for 16 at 1000 and for 32 at 10000. Obviously, the benefits of frequent LVT synchronization (i.e. fewer and less “severe” rollbacks) is gradually outweighed by the increasing costs of communication and strain on the communication network. While the acceleration for all cases in the 2 processor case tend to remain close

to the value 4, the acceleration for 32 processors vary greatly in respect to both the ratio and the model parallelism, besides seldom reaching the ideal value of $N^2 = 1024$.

5 A Large Business Process Model

Having seen a convincing performance improvement in the distributed document flow BPM execution above, we are further interested in the performance impact of less regular BPM models, i.e. cases where the decomposition into submodels is not obvious (like offices in the previous case). To this end, we have studied a much larger BPM, and systematically investigating the model partitioning issue.

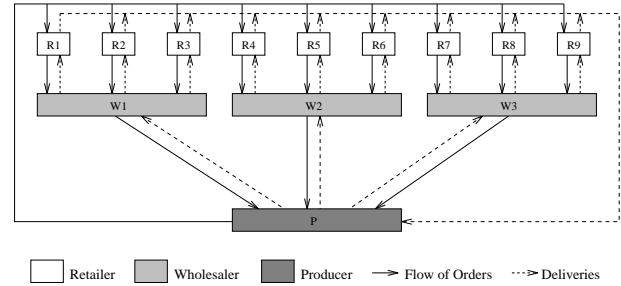


Figure 5: Schematic Structure of a Complex BPM

The model depicted in Figure 5 represents 13 interacting business organizations. Nine of the organizations are retailers (R1 – R9) which order products on demand from the respective wholesalers (organizations W1 – W3), which in turn order from the producer (P). After an order is received by the producer, the producer delivers to the respective wholesaler, who then forwards to the retailer. Each organization consists of the following work groups: (i) Incoming Orders (ii) Outgoing Orders (iii) Shipping and (iv) Receiving as illustrated in Figure 6 for the workflow within a wholesaler. The structure of retailers and of the producer is essentially the same.

For the case study, enabling delays of process elements are assumed to be exponential, with distribution parameters λ as well as the mean enabling delays, E , in minutes (m), hours (h) or days (d) given in Figure 6. Five workers are employed in the department Incoming Orders where orders from the retailers are processed. Three workers take incoming orders over the telephone, whereas the others process orders received by telefax or mail. The internal delays for the receipt of a telephone call, a telefax or letter are assumed to be 3 minutes, 5 minutes or 1 day respectively. Incoming calls are placed in a queue (delay 11s) and the next available worker accepts the

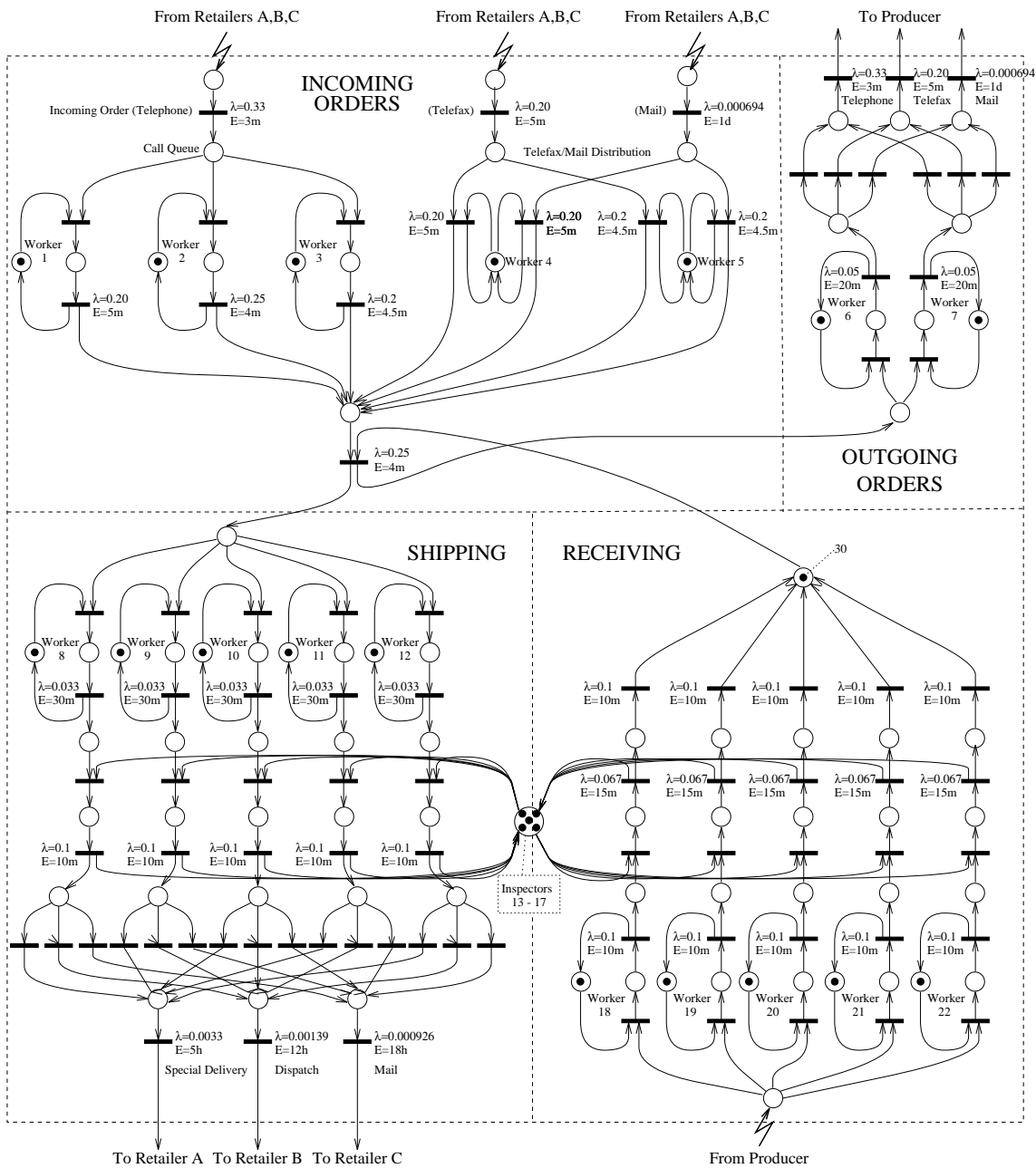


Figure 6: BPM of a Wholesaler

call and processes the order. After a mean processing time of approximately 5 minutes (worker dependent) the order is transferred electronically to the storehouse management (delay 4 minutes). The central transition connecting the four departments guarantees that when items are delivered, they are taken out of the storehouse, delivered, and also backordered from the producer. Thus, an incoming order causes not only items to be delivered, but also creates a new order.

In the department for Outgoing Orders, 2 workers ensure that items are backordered from the producer. Orders are queued (delay 11s) and the first available worker accepts the order. Each worker can process only one order at a time. Outgoing orders can be placed by telephone, telefax or letter. When an incoming order has been processed, it is also relayed to the Shipping department. Five workers are employed here to accept the incoming orders and prepare the

items for shipping to the retailers (mean service time 30 minutes). Shipping is checked by five inspectors who, in addition to controlling shipments also check the correctness of incoming deliveries. The items may be shipped by special delivery (5 hours), normal dispatching (12 hours) or by mail (18 hours). The Receiving department take shipments coming in by all means of transport. Five workers bring the shipments into the storehouse. The inspectors check that the shipment before the items are brought into the central storehouse.

5.1 Partitioning the large BPM

To allow for a distributed execution of the BPM, it must be *partitioned* into regions and assigned to LPs and henceforth processors. Various criteria for determining the optimal partitioning scheme for a given net have been developed and discussed in [14]. The resulting guidelines given there provide a theoretical basis for choosing among the vast number of possible partitioning schemes. Essentially, a good partitioning scheme will keep contiguous, strongly connected parts of the BPM in a single partition, thus reducing the number of messages which must be sent between the LPs. Furthermore, arcs which can be expected to have heavy traffic should not be cut, i.e. the respective place and transition should both remain in the same partition.

In the example BPM, the most logical partitioning of the net is the one that separates the organizations themselves. In the case of the wholesalers (see again Figure 6), this means that only 10 arcs must be cut (3 for incoming orders, 3 for outgoing orders, 3 for shipping and 1 for deliveries). These arcs represent the “natural” communication lines between the organizations in the model. Since there are 13 organizations in the model, a maximum of 13 processors can be used.

To provide for a finer grain partitioning of the model, however, each organization may be subdivided into two partitions (Incoming and Outgoing Orders in the first partition, referred to with subscript “a”, and Shipping and Receiving in the second partition, referred to with subscript “b”). The most problematic feature in the structure of the organization is the placement of the joint quality control place between Shipping and Receiving. Due to the resulting strong interconnectedness of the two departments, both are assigned to the same partition.

In Figure 7, several partitioning schemes are defined for varying numbers of processors. In addition to the ones enlisted in the table we consider the use of 17 and 26 processors: Case 17.1 assigns each of 9 retailers to a single processor and splits wholesalers

Case No.	Processor																											
	1				2				3				4															
4.1	R1	R2	R3	W1	R4	R5	R6	W2	R7	R8	R9	W3	P															
4.2	R1	R2	R3	W2	R4	R5	R6	W3	R7	R8	R9	W1	P															
4.3	R1b	R2b	R3b	W1	R4b	R5b	R6b	W2	R7b	R8b	R9b	W3	R1a -R9a P															
4.4	R1	R2	R3	R4	R5	W1	R6	R7	R8	R9	W2	W3	Pa Pb															
4.5	R1	R2	R3	R4 R5 R6				R7 R8 R9				W1 W2 W3 P																
	1				2				3				4				5											
5.1	R1	R2	R3	W1	R4	R5	R6	W2	R7	R8	R9	W3	Pa				Pb											
5.2	R1	R2	R3	W2	R4	R5	R6	W3	R7	R8	R9	W1	Pa				Pb											
	1			2			3			4			5			6			7									
7.1	R1	R2	R3	W1			R4	R5	R6	W2			R7	R8	R9	W3			P									
7.2	R1	R2	R3 W1			R4	R5	R6 W2			R7	R8	R9	R9 W3			P											
7.3	R1a	R2a	R3a	W1a	R1b	R2b	R3b	W1b	R4a	R5a	R6a	W2a	R4b	R5b	R6b	W2b	R7a	R8a	R9a	W3a	R7b	R8b	R9b	W3b	P			
7.4	R1	R2	R3	W1			R4	R5	R6	W2	R7	R8	R9	W3			Pa			Pb								
	1			2			3			4			5			6			7			8						
8.1	R1	R2	R3	W1			R4	R5	R6	W2			R7	R8	R9	W3			Pa			Pb						
8.2	R1	R2	R3 W1			R4	R5	R6 W2			R7	R8	R9	R9 W3			Pa			Pb								
8.3	R1a	R2a	R3a	W1a	R1b	R2b	R3b	W1b	R4a	R5a	R6a	W2a	R4b	R5b	R6b	W2b	R7a	R8a	R9a	W3a	R7b	R8b	R9b	W3b	Pa		Pb	
8.4	R1	R2	R3	W1			R4	R5	R6	W2			R7	R8	R9	W3a			W3b			13						
	1		2		3		4		5		6		7		8		9		10		11		12		13		14	
11.1	R1	R2	R3	W1a		W1b		R4	R5	R6	W2a		W2b		R7	R8	R9	W3a		W3b		Pa		Pb				
	1		2		3		4		5		6		7		8		9		10		11		12		13		14	
13.1	R1	R2	R3	W1	R4	R5	R6	W2	R7	R8	R9	W3	P				-											
14.1	R1	R2	R3	W1	R4	R5	R6	W2	R7	R8	R9	W3	Pa				Pb											

Figure 7: BPM Partitioning Schemes

and the producer onto two processors, whereas Case 26.1 splits each of the 13 organizations onto two processors. The partitioning schemes chosen for investigation have been selected not solely on the grounds of their suitability. Some less promising schemes have been included in order to demonstrate the effect of partitioning on overall simulation performance.

5.2 Distributed Execution Performance

Each of the 20 cases (partitionings of the BPM) was executed repeatedly (about 30 times) on the Meiko CS-2, until statistical significance was achieved for the performance results. Each execution was carried out to a GVT of 14400 minutes, i.e. 4 days of operation of the business organization system was simulated.

The performance data collected is summarized in Figure 8. Column *Exec. Time* stands for the overall execution time in seconds, *Process Exec.* is the total number of processes executed over *all* processors, resulting in the *Raw Rate*, the raw event execution rate. Since not all process executions ultimately contribute towards reaching the end state of the execution due to TW’s optimism (some executions may be invalidated in the course of rollback), the *Comm. Exec.* reveals the number of committed executions, yielding *Event Rate*,

Case No.	Exec. Time	Process Exec.	Raw Rate	Comm. Exec.	Event Rate	Event Rate per Proc.		
						Min.	Ave.	Max.
4.1	198.2	184826	940.4	101283	523.1	112.4	130.8	174.6
4.2	190.3	168748	891.0	102279	540.1	120.7	135.0	171.2
4.3	226.7	216843	987.9	94818	420.4	73.6	105.1	189.8
4.4	286.7	210398	834.8	114497	466.0	56.0	116.5	168.6
4.5	312.2	220102	704.5	84354	271.9	30.9	68.0	174.7
5.1	163.9	191122	1168.7	113759	694.8	84.1	139.0	162.9
5.2	196.2	213770	1091.5	124314	635.2	74.3	127.0	143.2
7.1	181.7	280236	1544.7	112775	622.6	67.0	88.9	198.1
7.2	170.0	265911	1566.4	104328	616.9	45.1	88.1	198.9
7.3	183.6	268239	1469.8	111748	616.5	53.8	88.1	190.1
7.4	147.0	250029	1703.6	117138	797.4	85.6	113.9	181.7
8.1	136.2	265780	1954.2	118773	873.1	91.8	109.1	195.4
8.2	135.5	269051	1987.0	120278	887.9	91.9	111.0	198.2
8.3	136.9	267718	1962.1	119197	875.1	91.5	109.4	195.0
8.4	175.9	340713	1949.2	107275	620.9	30.0	77.6	195.6
11.1	142.2	352139	2495.7	124832	905.2	39.5	82.3	196.9
13.1	163.2	420815	2574.1	106247	656.1	21.9	50.5	206.3
14.1	141.2	412210	2919.0	121939	864.3	28.1	61.7	192.9
17.1	147.8	564576	3834.3	127480	870.5	27.6	51.2	188.0
26.1	137.0	718526	5241.4	134109	979.7	13.4	37.7	206.0

Figure 8: Distributed Execution Performance

the rate of confirmed process executions. The *Event Rate per Processor* serves as a measure of processor utilization, i.e. how much work done on each processor ultimately contributed towards the final goal.

As a general observation, lower execution times can be achieved with higher processor counts but at the cost of lower processor utilization (i.e. the average event rate per processor): more processors can generate a much higher raw (event) rate, and still exhibit higher performance in the more meaningful true event rate metric. Case 26.1 exhibits the highest performance of all cases with 979.7 committed process executions per second, but Case 11.1, using less than half the number of processors is not far behind with 905.2. Using less than a third the number processors (Case 8.2) still comes close (887.0).

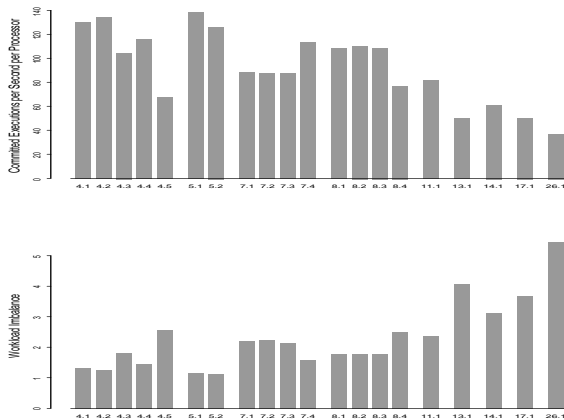


Figure 9: Event Rate / Proc., Workload Imbalance

The higher the processor count, however, the greater the disparity between the raw event rate and the (true) event rate resulting from the fact the an increasing portion of the work accomplished by a processor is subsequently invalidated by rollbacks. This

is mainly due to the increasing imbalance in workload introduced by the partitioning schemes. As a metric for workload imbalance, we consider the ratio of maximum committed process executions per processor per second (i.e. maximum per processor event rate) to the average committed firings per processor per second. Figure 9 summarizes this workload imbalance factor and relates it to the average event rate per processor. It can be seen that Cases 4.1, 4.2, 5.1. and 5.2 have the lowest workload imbalance, whereas the cases with higher processor counts tend to more be imbalanced.

The results obtained tend to confirm the presumption that the wholesalers (W1 – W3) will have roughly 3 times the workload of the retailers (R1 – R9) and that the producer (P) will have three times the workload of the wholesalers. Thus, the placement of the producer in the partitioning scheme is the one most important factor determining the overall performance of the simulation.

For the 4 processor partitioning cases which place the producer in a partition by itself (Case 4.1 and 4.2) outperform the cases where the producer is placed together with other components (Cases 4.3 and 4.5). Splitting the producer onto two separate partitions and consequently having to put the wholesalers together with the retailers (Case 4.4) likewise does not serve to enhance performance.

Cases 5.1 and 5.2 which are analogous to Cases 4.1 and 4.2 except that the producer is split onto two separate partitions outperform the 4 processor cases in average event rate per processor, again confirming that the producer has the heaviest workload of all organizations. Case 5.1 is the most efficient of all partitioning schemes under study (with respect to average event rate per processor) owing most probably to the fact that besides placing the producer in a partition by itself, the wholesalers are placed in a partition with their respective retailers thus reducing the need for communication between retailers and wholesalers.

The Cases 8.1, 8.2 and 8.3 which are analogous to Cases 7.1, 7.2 and 7.3 except that the producer is split onto two separate partitions behave similarly to the 4 and 5 processor cases. Case 7.4 performs best of all 7 processor cases most likely due to the fact that the producer is split. Case 8.4, on the other hand, performs the poorest of all 8 processor cases – a performance penalty for *not* splitting the producer.

For higher processors counts (11 – 26 processors), the disparity in workload distribution becomes increasingly large, thus lowering the average event rate per processor and increasing the strain on the processors simulating the producer.

6 Discussion and Conclusion

Modeling and analysis of business organizations is required for the purpose of redesigning an enterprise's business process to make the organization more efficient (*Business Process Reengineering*), as well as for the purpose of establishing advanced coordination technology in the organization. In this work we have used an abstract framework of business process systems and a modeling formalism based on timed Petri nets. The major advantages of this framework are the high expressive power of the Petri net modeling formalism and a profound mathematical background from the theory of Petri nets together with well established methods which allow for *qualitative* and *quantitative* analysis of BPMs within one and the same framework.

The main contribution of our work is the development of a distributed execution environment for realistically sized and very large BPMs, the primary motivation being to accelerate the execution using multiprocessor facilities. The key achievements are:

- An optimistic distributed BPM execution environment based on asynchronous logical processes synchronized with the Time Warp protocol was developed and implemented.
- The current implementation is in operation on the 134 processor Meiko CS-2 massively parallel computing platform. The software is realized as an SPMD program using the MPI communication library. As such, the code is portable to *almost any* parallel or distributed computing platform.
- We have demonstrated that distributed techniques make executions of very large BPMs tractable: in situations where a model has reached a complexity such that it cannot be executed on a single processor computer (e.g. due to memory constraints), the distributed execution remains the only means. Via distributed execution, a whole new class of (complex) BPMs has become practically tractable.
- Convincing performance improvements could be achieved: Applied in realistic BPM executions (a document flow system comprising 64 offices with an average of 1000 documents per office), a 250 fold acceleration of the overall execution speed using 32 processors of the CS-2 could be gained.
- A detailed case study has revealed the performance sensitivity of our distributed BPM execution environment with respect to various BPM

decompositions. The disparity of workload distribution potentially lowers the average event rate of LPs. Further investigations are necessary to relate the non trivial partitioning issues to automated strategies for performance optimization.

References

- [1] C. A. Ellis and G. J. Nutt, "Modeling and enactment of workflow systems", in *Proc. of the 14th Int. Conf. on Application and Theory of Petri Nets 1993, Chicago, June 1993*, M. Ajmone Marsan, Ed., Berlin, 1993, Lecture Notes in Computer Science 691, pp. 1 – 16, Springer Verlag.
- [2] A. Ferscha, "Qualitative and quantitative analysis of business workflows using generalized stochastic petri nets", in *Connectivity 94: Workflow Management*, G. Chroust and A. Benczur, Eds., Vienna, 1994, pp. 222 – 234, Oldenbourg.
- [3] W. M. P. van der Aalst, "Three good reasons for using a petri-net-based workflow management system", in *Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96)*, S. Navathe and T. Wakayama, Eds., Cambridge, Massachusetts, 1996, pp. 179–201.
- [4] T.M. Kouloupoulos, *The Workflow Imperative*, Van Nostrand Reinhold, New York, 1995.
- [5] T. Murata, "Petri nets: Properties, analysis and applications", *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.
- [6] W.M.P. van der Aalst and K.M. van Hee, "Business process redesign", *Computers in Industry*, vol. 29, no. 1–2, pp. 15–26, 1996.
- [7] W. M. P. van der Aalst, "Verification of workflow nets", in *Application and Theory of Petri Nets 1997*, P. Azema and G. Balbo, Eds., Berlin, 1997, Lecture Notes in Computer Science, Vol. 1248, pp. 62 – 81, Springer Verlag.
- [8] A. Ferscha, "Parallel and distributed simulation of discrete event systems", in *Parallel and Distributed Computing Handbook*, A. Y. Zomaya, Ed., pp. 1003 – 1041. McGraw-Hill, 1996.
- [9] A.W. Holt, "Coordination technology and petri nets", in *Advances in Petri Nets 1985*, G. Rozenberg, Ed., Berlin, 1985, Lecture Notes in Computer Science No. 222, pp. 278 – 296, Springer Verlag.
- [10] H. Fleischhack and A. Weber, "Rule based programming, predicate transition nets and the modeling of office procedures and flexible manufacturing systems", Tech. Rep. Bericht TI 3, Universit"at Oldenbourg, 1989.
- [11] A. Ferscha, "Concurrent execution of timed petri nets", in *Proceedings of the 1994 Winter Simulation Conference*, J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, Eds., 1994, pp. 229 – 236.
- [12] A. Ferscha, "Adaptive model parallelism exploitation in parallel simulation", in *EUROSIM'95*, F. Breitenacker and I. Husinsky, Eds., Amsterdam, 1995, pp. 241 – 248, North-Holland.
- [13] R. M. Fujimoto, "Parallel discrete event simulation", *Communications of the ACM*, vol. 33, no. 10, pp. 30–53, October 1990.
- [14] G. Chiola and A. Ferscha, "Exploiting timed petri net properties for distributed simulation partitioning", in *Proceedings of the 26th Hawaii Int. Conf. on Systems Sciences*. 1993, pp. 194 – 203, IEEE Computer Society Press.