Approaching the Distributed Simulation of Related Business Processes

by

Felix Elliger

A Thesis Submitted to the
Business Process Technology Group
in Partial Fulfillment of the Requirements for the Degree of

Master of Science
in
IT Systems Engineering

at the Hasso Plattner Institute at the University of Potsdam
November, 2011

Advisor: Prof. Dr. Mathias Weske

Abstract. Business process simulation is a powerful tool for the analysis and improvement of business processes based on their models. It is, however, not sufficient to regard a single business process as an isolated unit. Instead, a process is related to other process by sub-process invocations or resource constraints. This thesis addresses the problem of concurrently simulating multiple related business processes along with their resources. Therefore, we develop a formal model of simulation scenarios and provide a mapping onto extended timed colored Petri nets. As such scenarios may grow large, e.g. for the simulation of a whole department or company, we consider the distribution of the simulation as a reasonable measure for achieving a performance improvement. Thus, the generated Petri net must be divided into partitions. Within this thesis, two algorithms for partitioning the gathered Petri nets are proposed. Their effect on the simulation performance is evaluated using a conservative distributed simulation engine. The architecture of the accompanying prototypical implementation is sketched and selected components are described in more detail. In conclusions, we infer on the potential of distributing business process simulations.

Zusammenfassung. Geschäftsprozesssimulation ist ein mächtiges Instrument für die Analyse und Verbesserung von Geschäftsprozessen auf Basis ihrer Modelle. Es ist jedoch unzureichend einen einzelnen Geschäftsprozess als isolierte Einheit zu betrachten. Vielmehr ist dieser durch Subprozess-Aufrufe und Resourcenbeschränkungen mit anderen Prozessen verbunden. Diese Arbeit adressiert das Problem der gleichzeitigen Simulation mehrerer, verbundener Geschäftsprozesse und ihrer Ressourcen. Zu diesem Zweck, entwickeln wir ein formales Modell für Simulationsszenarien und stellen eine Abbildung auf erweiterte, zeitbehaftete, gefärbte Petri-Netze zur Verfügung. Da diese Szenarien möglicher Weise sehr groß werden, zum Beispiel für die Simulation einer gesamten Abteilung oder eines ganzen Unternehmens, wird die Verteilung der Simulation als sinnvolle Maßnahme zur Verbesserung der Performanz betrachtet. Dazu ist es notwendig das generierte Petri-Netz in Partitionen zu zerlegen. In dieser Arbeit werden zwei Algorithmen zur Partitionierung der Petri-Netze beschrieben. Ihr Einfluss auf die Performanz der Simulation wird anhand einer konservativen verteilten Simulationseinheit evaluiert. Die Architektur der begleitenden prototypischen Implementierung wird skizziert und ausgewählte Komponenten im Detail beschrieben. Abschließend werden Schlussfolgerungen zum Potential der verteilten Geschäftsprozesssimulation gezogen.

Acknowledgements

In first place, I would like to thank Frank Puhlmann who had the initial idea for the topic and supported me whenever a problem occurred. Second, I appreciate the supervision by Prof. Mathias Weske who kept asking the right questions and always had some guiding advice. Since writing a thesis over half a year is challenging, a comfortable working atmosphere is of highest importance. Therefore, I would like to thank all the people at the inubit AG for making this place as enjoyable as it still is. Especially Sebastian and Carsten, who always have a joke right at hand when I need to free my mind, have become more than just colleagues. Besides those who made working a lot easier, I would like to thank my parents, as all this would have never been possible without them. In the last and, for me, most important place, I would like to thank Josy, who came into my life at the right time. Thank you Josy, for always keeping me grounded, and staying supportive even in times of high workload.

Contents

1. Introduction **1**
1.1. Questions and Requirements . . . 2
1.2. Contribution . . . 4
1.2.1. Research contribution . . . 5
1.2.2. Prototypical Implementation . . . 5
1.3. Structure of the Thesis . . . 6

2. Related Work And Preliminaries **9**
2.1. Related Work . . . 9
2.2. Business Process Management . . . 10
2.3. Simulation in the Context of BPM . . . 13
2.4. Petri nets . . . 14
2.5. Distributed Discrete Event Simulation . . . 18

3. Simulation Scenarios and Their Formal Representation **23**
3.1. Conceptual Architecture . . . 23
3.2. Simulation Modeling . . . 24
3.2.1. Models and Their Configuration . . . 25
3.2.2. Formal Model for Simulation Scenarios . . . 30
3.3. Mapping Simulation Scenarios to Petri Nets . . . 35
3.3.1. Mapping Time . . . 37
3.3.2. Mapping Simulation Business Process Models . . . 38
3.3.3. Mapping Human and Non-Human Resources . . . 42
3.3.4. Remarks and Additional Mappings . . . 45

4. Simulation Engine and Model Partitioning **47**
4.1. A Logical Process for Conservative Distributed DES . . . 47
4.2. Model Parallelism and Partitioning Rules . . . 54
4.3. Bottom-Up Partitioning . . . 59
4.4. Top-Down Partitioning . . . 62

5. Prototypical Realization **67**

6. Evaluation **73**

6.1. Example Scenarios 73

6.1.1. Example 1: TurboSoft Inc. 73

6.1.2. Example 2: Product Supply Chain 74

6.2. Statistics 76

6.3. Result Analysis 81

7. Summary and Outlook **83**

Bibliography **85**

A. TurboSoft Inc. Processes **91**

1. Introduction

Imagine you were the manager of a manufacturing company. Your simple goal is to increase the winnings of your company. Therefore, you want to create highest quality products at their lowest possible production costs. Further, you want to produce sufficient items to fulfill every order as fast as possible. However, as the number of orders is rapidly increasing, you are close to losing potential customers due to a bottleneck in your production process.

What could you do to overcome this bottleneck? Employ more workers? Buy more machines? Change the production process? On the one hand, there exists a variety of opportunities. On the other hand, it would be way too risky and too expensive to test each of these opportunities in your real company. Therefore, you need to decide for efficient measures. Finding these measures is a difficult and responsible task. However, if you can not test the opportunities in the real company, it would be helpful to try them out fictively.

A suitable tool for this kind of problem, is business process simulation. As stated by White and Ingalls in [65], "Simulation is a particular approach to studying models, which is fundamentally experiential or experimental'. Applying this statement to our example, your company is the model under study. Using modeling notations like organizational charts and the Business Process Model and Notation (BPMN) [22], it is possible to describe a company's structure and processes by models, which can be used as inputs for a business process simulation engine. Further, models provide means for virtually employing more or less workers and machines, or changing the processes, without affecting the day-to-day business.

Since a process cannot, in general, be regarded as an isolated unit, it is not sufficient to simulate only one process at a time. Instead, a process might invoke other processes or is competing for the use of a specific resource with other processes. Therefore, the related processes need to get simulated at the same time. Regarding our example, this implies, if you want to improve a certain step in your production, you need to consider all processes that are related to the production to derive appropriate measures.

However, simulating a whole department or even a complete company is a challenging task. First, a user has to be enabled to model such scenarios, consisting of processes, human resources and non-human resources. The models have to describe, e.g., the number of created instances, execution times of tasks, and working times of employees. Second, a formalism must be available that is able to express the concepts described by the models and is, at the same time, suitable for simulation. Third, a mapping must exist, that maps the models, their properties, and their dependencies onto the formalism. Fourth, there has to be a simulation engine that runs the simulation in an appropriate time, although handling thousands of process instances, and delivers meaningful results to the user.

This thesis describes an approach for handling the simulation of multiple related processes under consideration of their required human and non-human resources. We develop a formal model for such simulation scenarios and provide a mapping onto the formalism of timed colored Petri nets. Further, we describe a simulation engine that is capable of distributing the simulation of the generated net among a set of simulation engines. We assume the distribution to be a reasonable step in order to gain performance compared to the classic simulation running on a single engine. This gain has already been documented by Ferscha for single process simulations in [17], where they observed a 200 to 250 fold gain over the single-threaded execution. Using a prototypical implementation of our approach, an evaluation using realistic examples is conducted and conclusions are drawn.

The remainder of this chapter is structured as follows. Section 1.1 states requirements imposed on an appropriate solution to the described problem. Based thereon, section 1.2 presents a detailed outline on the research contribution of this thesis. Concluding this chapter, section 1.3 outlines the structure of this thesis.

1.1. Questions and Requirements

When approaching the concurrent simulation of multiple related processes, the level of supported constructs and concepts can be almost arbitrarily chosen. This section defines the scope of our approach. The listed requirements serve as a basis for the investigations described in the next chapters. We partitioned them into *modeling*, *functional*, and *technical* requirements.

Modeling Requirements Since simulation is only an approximation of the real world behavior, we must provide sufficient means for defining reasonable simulation models.

MR1: Resource Modeling There must exist a graphical language for specifying organizational structures, composed by units, roles, and persons. Further, a mechanism for determining inventories must be provided.

MR2: Process Modeling Since processes are the core element of our simulation, a graphical, and thereby user-friendly, process modeling language must be available. The language has to provide constructs for expressing the following workflow patterns as described by Russel et al. in [46]: *Sequence*, *Parallel Split*, *Synchronization*, *Exclusive Choice*, *Simple Merge*, *Structured Loop*, *Cancel Activity*, and *Deferred Choice*. Processes can be linked to other processes, i.e. representing a sub-process relation or a message exchange. In addition, the tasks of the processes can be connected to human and non-human resources, specifying their resource relations.

MR3: Time Modeling The purpose of basically every kind of simulation is to reveal the evolution of a system over time. Therefore, means for modeling times and time constraints must exist. For human resources, working times have to be specifiable determining their availability for work. For tasks, the duration of a single execution is relevant. Please note: These execution times might be affected by several factors, and are not a priori constant. Further, the number and frequency of created process instances might vary over time, e.g. there are less instances over night than at daytime. Therefore, the user must be enabled for the specification of such facts. The specification of times must be available in a comprehensible format.

MR4: Probability Modeling Decisions made by humans or based on process data are usually hard, or even impossible, to be simulated. To overcome this obstacle, probabilities need to be configurable for the respective decision points.

MR5: Scenario Modeling To determine the extent of a single simulation run, a way for grouping process models, organizational charts, and inventory lists together to form a *simulation scenario* must be provided. Further, the start and end point of simulated time in terms of concrete dates has to be specified.

Functional Requirements For executing a simulation consisting of multiple related processes and resources, the following functional requirements must be fulfilled.

FR1: Model Formalism A formalism must be found, that is powerful enough, though

still appropriate for automatic simulation, to represent the modeled concepts, entities, and relations.

FR2: Mapping Function For translating the original simulation models into models of the formalism, a mapping must be provided.

FR3: Simulation Engine The formal description of the simulation scenarios form the basis for the simulation. Therefore, a simulation engine is required that knows the execution semantics of the formalism.

FR4: Reporting After a successful execution of a simulation run, the results of the simulation must be delivered to the user. This report contains, among other issues, the number of created process instances for each process.

Technical Requirements As mentioned in the introduction, we assume the distribution of the simulation among a set of computational resources to be a reasonable step for improving the performance of the simulation. However, distribution imposes technical requirements to be considered.

TR1: Model Partitioning The different computational nodes, each of them running a simulation engine as claimed by *FR3*, must be assigned to handle a specific part of the simulation model. Therefore, the model must be reasonably partitioned, such that the model parallelism is exploited to the best possible extent.

TR2: Synchronization The concurrently running engines must synchronize their execution, in order to deliver the same causally correct results as the single-engine approach would do.

TR3: Communication For synchronization purposes the engines must be enabled to communicate with other engines if this is necessary.

1.2. Contribution

This section describes the contribution of the thesis and the corresponding prototypical implementation. Section 1.2.1 outlines the contribution to research. In Section 1.2.2, we describe the functional extent of the prototype which has been built in context of this work.

1.2.1. Research contribution

As outlined in the motivation for this thesis, the main goal is the development of an approach for enabling an efficient simulation of multiple related business processes and their participating resources. It is, however, not sufficient to put an exclusive focus on the simulation itself, but to investigate the fields of business process modeling, simulation modeling, and model formalisms in general. Therefore, the contribution of the thesis comprises as follows:

Literature Analysis The fields of business process simulation and distributed simulation in general are not new to research, but have been investigated for quiet a long time. This creates a broad grounding for this work. Therefore, we analyze and summarize the current state of research especially in these two areas.

Formal Model of Simulation Scenarios Based on our observations made in the existing literature, we specify formal descriptions of processes, resources and finally simulation scenarios.

Mapping Onto Petri Nets We describe a mapping of the supported modeling concepts onto the formalism of timed colored Petri nets. Further, we state reasons for choosing Petri nets as a formalism.

Partitioning Algorithms Taking the Petri net representation and the findings from literature as a starting point, we describe concrete algorithms for partitioning the generated net.

Algorithm Evaluation The described algorithms are evaluated using two realistic simulation examples. The results will be compared to those of the single-engine simulation and conclusions are drawn.

1.2.2. Prototypical Implementation

Driven by the research contribution of the thesis, the implemented prototype provides the following features.

Simulation modeling Via web-based graphical interfaces, the user is enabled to model processes and organizational structures, specify simulation parameters, and compose simulation scenarios of multiple models.

Petri net mapping The specified simulation scenarios are automatically mapped onto

timed colored Petri nets, which provide the required formalism for the specification of our algorithms.

Simulation Engine The prototype implements a simulation engine for the discrete simulation of the generated Petri nets. For our prototype, we consider the distribution of the simulation among multiple CPU cores of a single machine. However, a physical distribution onto multiple machines should be attainable by minor implementation efforts. For excluding implementation issues as a source of error during evaluation, we use the same engine for both, the single-threaded and the distributed simulation.

Partitioning algorithms The prototype implements two partitioning strategies for distributing the generated Petri net.

Evaluation framework For analyzing the performance of the different partitioning algorithms, there is a simple framework for evaluating the simulation runs in terms of their execution time.

Visualization of simulation results Although this work is focused on the investigations with respect to the distribution strategies of the simulation, the prototype provides a simple visualization of the gathered simulation results. This visualization enables the user to see, e.g., the number of generated process instances, instance execution times, and, furthermore, provides a simple mechanism for pointing the user to problematic points in the simulated processes.

1.3. Structure of the Thesis

The remainder of this thesis is structured as follows. Chapter 2 describes preliminaries for this thesis. Related work is discussed, an introduction to business process management and simulation is given, and Petri nets are defined. Concluding this chapter, we concisely describe discrete event simulation and its distribution. In chapter 3, the first part of our contribution is presented. After a sketch of the abstract system architecture, the formal model for simulation scenarios is presented. The chapter is completed by the mapping of simulation scenarios onto timed colored Petri nets. Subsequently, chapter 4 commences by describing the simulation engine that enables distribution. This chapter further investigates the basics for model partitioning and presents two concrete algorithms for partitioning timed colored Petri nets. In chapter 5, we provide insights on some vital components of our accompanying prototypical implementation. Using this prototype

and the algorithms developed in chapter 4, evaluation results for two realistic use cases are shown in chapter 6 and conclusions are drawn. Conclusively, chapter 7 summarizes this thesis and gives an outlook on future work.

2. Related Work And Preliminaries

Within this chapter, we thoroughly describe the functional and technical grounding for this thesis. Section 2.1 gives a concise overview on related work. In section 2.3 we integrate the technique of simulation into the discipline of business process management described in section 2.2. Section 2.4 introduces the concept of Petri nets along with its extensions of timed and colored Petri nets. Concluding this chapter, section 2.5 gives a general introduction to discrete event simulation and its distribution.

2.1. Related Work

Simulation is a well-established technique applied in versatile contexts. Using simulation for the analysis of business processes has already been documented in the 1970s by Shannon in [48]. Concise introductions are given, e.g., by Shannon in [49] and White and Ingalls in [65]. Recently, a list of problems of business process simulation approaches has been published by van der Aalst in [57]. Among other issues, the oversimplification of simulation models is remarked which includes the pure focus on single processes. This result is supported by a tool survey conducted of Jansen-Vullers and Netjes ([30]). Besides business process simulation tools, the survey analyzed some general purpose simulation tools, which might be configured for multi-process simulation. However, these tools are based on proper modeling languages resulting in high effort for translating the original models to models of the tool.

Using business process simulation as a powerful tool for business process re-engineering has been documented, e.g., by Tumay ([52]) and Hlupic and Robinson ([24]). In addition these papers argue that *discrete event simulation (DES)* is the most suitable and powerful simulation technique for business processes. A conceptual introduction to DES is given by Schriber and Brunner in [47]. The distribution of simulation is thoroughly described by Fujimoto in [19]. Prominent works on simulation engines for distributed discrete event simulation have been published by Misra ([34]), Chandy ([7]), and Jeffer-

son ([25]). Overviews on synchronization strategies have been published by Ferscha in [16] and Zarei in [66].

The suitability of Petri nets as a formalism for business processes has been thoroughly investigated by van der Aalst ([55]). Dijkman et al. ([13]) provide a mapping of BPMN constructs to Petri nets covering only the structure of the process model. The mapping is enriched by considering simulation parameters in [29]. However, this mapping of Krumnow et al. does not cover all modeling constructs used in this thesis. Another well-documented formalism is the π-calculus, originally proposed by Milner et al. in [33]. Its suitability is shown by Puhlmann in [42] who further presents a mapping of workflow patterns to the calculus in [43].

For both formalisms an approach for applying the DES technique is available. Wang and Wysk ([62]) give the approach for the π-calculus. In [9], Chiola and Ferscha already describe the distributed discrete event simulation of timed Petri nets. These are, however, generic approaches to DES and are only shown in a superficial, i.e. non-algorithmic, manner for only very small examples. It lacks a bridge between business processes and an effective automated partitioning for their distributed simulation. In general, this partitioning is independent of the chosen formalism. A first approach for distributing the execution of business process models is shown in [15]. However, the proposed process models are low-level Petri nets, whereas this thesis considers high-level modeling languages. Another side-effect of using low-level Petri nets is the inability of distinguishing between process instances which is necessary for a meaningful reporting of simulation results. An exhaustive list of reporting issues is given by Anupindi et al. in [3].

The modeling of resources and their assignment to tasks is a vital part of our approach. Zur Mühlen ([35]) gives an overview on this topic. The necessity of considering human resources for simulation is discussed by Baines et al. in [5]. Graphical meta-models for resources and patterns for their assignment to process activities are presented by Russel et al. in [45]. There is, however, no formalization of the depicted meta-models. Further, an approach for capturing human work behavior is proposed by van der Aalst et al. in [60].

2.2. Business Process Management

The discipline of business process management (BPM) is defined by van der Aalst et al.([58]) as follows.

Definition 1 (Business Process Management (BPM)) *"Supporting business processes using methods, techniques, and software to design, enact, control, and analyze operational processes involving humans, organizations, applications, documents and other sources of information."*

A basic concept of BPM "is the explicit representation of business processes with their activities and the execution constraints between them." ([64]) This leads to the definition of a business process.

Definition 2 (Business Process (cf. [64])) *A* business process *consists of a set of activities, jointly realizing a business goal, that are performed in coordination in an organizational and technical environment.*

Business processes are subject to analysis, change, and implementation. The life-cycle of a business process is captured in figure 2.1, depicting an iterative process of four phases that represent the relevant concepts in BPM.

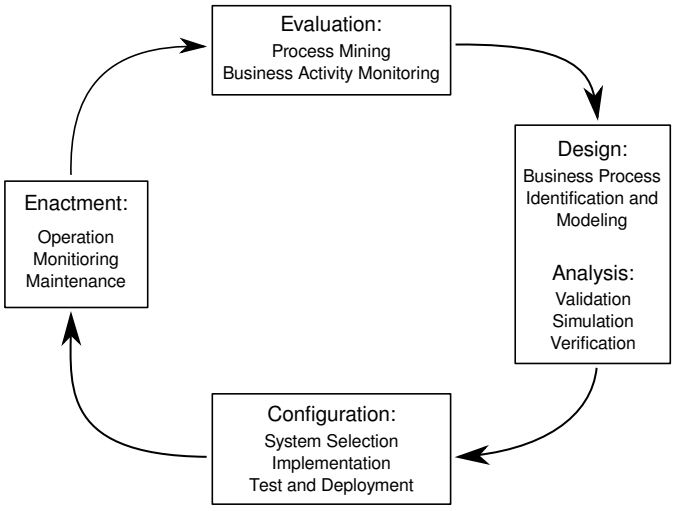


Figure 2.1.: Business Process Life-cycle (cf. [64])

In its first iteration, the life-cycle starts in the *Design and Analysis* phase. This phase targets the identification of processes and organizational structures along with their explicit representation using process models. Usually, graphical modeling languages, e.g. YAWL ([54]), event-driven process chains ([27]), BPMN ([22]), or UML activity diagrams ([21]), are used to achieve process models. We will have a closer look at the Business Process Model and Notation (BPMN) later in this section. Another part of this first phase is the analysis of the identified processes. Analytical tasks assure the quality and correctness of the identified processes and can be used to improve them.

After its design, the *Configuration* phase identifies possible realizations of the process. The concrete implementation of a process differs according to its degree of automation or computer support. Until this point in the life-cycle we used an artifact, namely the process model, that serves as a kind of blueprint for all instances of this process.

Definition 3 (Business process instance (cf. [64])) *A* business process instance *is a concrete case in the operational business of a company following the rules depicted by its corresponding business process.*

Once the process is implemented and configured, it is ready for execution (*Enactment* phase), i.e. process instances are created. During execution, monitoring components collect information on the running process instances. The gathered information can be used in the *Evaluation* phase for further process improvement and the determination of *key performance indicators* ([40]).

As already mentioned above, there exists a variety of business process modeling languages. A prominent example is the *Business Process Model and Notation(BPMN)*. BPMN is a graphical, graph-based language, supporting the representation of various control flow constructs. Besides the modeling of tasks and their control flow dependencies, BPMN supplies a simple mechanism for referring to organizational resources and artifacts, like process data. A sample BPMN process model is depicted in figure 2.2.

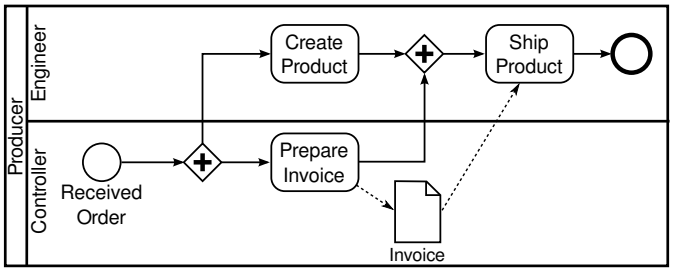


Figure 2.2.: A BPMN process example

The sketched process is instantiated on the reception of an order, i.e. the occurrence of an event. Concurrently, an engineer produces the requested item and a controller creates an invoice. The simultaneity of the tasks is indicated by the diamond incorporating a plus-symbol. If both persons have completed their tasks the product is shipped and the process instance terminates. The box around the actual process, called *Pool*, describes its organizational context. By placing tasks into different *Lanes* of the pool, the responsibilities for the task execution are defined. We further observe a data dependency between *Prepare Invoice* and *Ship Product*.

Since its version 2.0, each BPMN element has clearly defined execution semantics, which enables an unambiguous execution and simulation of processes specified as BPMN process models.

2.3. Simulation in the Context of BPM

As stated by White and Ingalls in [65], "Simulation is a particular approach to studying models, which is fundamentally experiential or experimental". Models are appropriate abstractions of real world facts, created for a specific purpose, e.g. the understanding of complex issues.

As we can see from figure 2.1 in the previous section, business process simulation (BPS) is a vital part of the business process life-cycle. As such, it is concerned with the validation and improvement of business processes and can be used effectively for re-engineering already existing processes ([52]).

Simulation is a rather practical approach to the evaluation of models. A simulation engine is used to test the models in a virtual environment. For business processes, this environment usually consists of entities representing humans or machines that are required to conduct the process. Due to this virtual execution of processes it is possible to arbitrarily change the processes and their environment without affecting the day-to-day business. This makes BPS a primary tool for *what-if-analyses*.

In general, the simulation engine embodies a notion of time, which is advanced during simulation. The progression of time can be either continuous or discrete. A famous simulation approach applying a continuous notion of time are *System Dynamics* ([44]). In a process model, however, discrete points in time can be identified where the state of a process instance changes, e.g., by the begin or completion of a task. Therefore, *discrete event simulation(DES)* is a well-suited approach to BPS. Detailed information on DES is presented in section 2.5.

In contrast to simulation, analytical approaches try to capture the behavior of a system or process using mathematical equations. Resolving these equations leads to the requested results. A prominent example of such techniques in the context of business processes analysis is shown by Magnani and Montesi in [31]. Creating and resolving such mathematical equations is an elaborate task, which is usually unaffordable for complex systems.

Generally speaking, although analytical techniques deserve their existence and return accurate results, simulation is a fair approach for running experiments on business processes while still providing meaningful results.

2.4. Petri nets

This section recalls the basic definitions of Petri nets and its extensions of Timed and Colored Petri nets. The original concept has been introduced by Petri ([41]) in 1962 and has, since then, been applied in several contexts and extended by further concepts, e.g. time. A comprehensive introduction to basic Petri nets, their properties, and applications is presented by Murata in [36].

Definition 4 (Petri net) *A* Petri net*, or simply a* net*, is a tuple* $N = (P, T, F)$ *with* P *and* T *as finite disjoint sets of places and transitions, and* $F \subseteq (P \times T) \cup (T \times P)$ *as the flow relation.*

Places are graphically represented as circles, transitions as rectangles, and the flow relation as directed edges between them, composing a bipartite graph. For a node $n \in P \cup T$ of net N we define the set of *inputs*, also called *preset*, as ${}^{\bullet[N]}n = \{x | (x, n) \in F\}$, and the set of *outputs*, also called *postset*, as $n^{\bullet[N]} = \{x | (n, x) \in F\}$. For notational simplicity, we will use ${}^{\bullet}n$ and $n^{\bullet}$ if it is unambiguous to which net we are referring.

Definition 5 (Labeled net) *A* labeled net *is a tuple* $N = (P, T, F, \mathcal{T}, \lambda)$*, where*

- (P, T, F) *is a Petri net,*
- $\mathcal{T}$ *is set of labels, with* $\tau \in \mathcal{T}$*, and*
- $\lambda : T \to \mathcal{T}$ *is a function assigning labels to transitions.*

If $\lambda(t) = \tau$, t is a τ-transition; otherwise, t is *observable*.

The state of a net is represented by a function $M : P \to \mathbb{N}$, assigning a number of *tokens* to each place. M is called a *marking* of the net. A transition $t \in T$ is *enabled* in a marking M, if, and only if, $\forall p \in {}^{\bullet} : tM(p) \geq 1$. An enabled transition can be fired. The *firing* of a transition t removes a token from each place $p_{in} \in {}^{\bullet}t$ and produces a token for each place $p_{out} \in t^{\bullet}$, thereby, creating a new marking M'. We denote this fact by writing $M[t\rangle M'$. Assuming an initial marking M_i, a marking M_n is *reachable* if there are transition $t_0, t_1, \ldots, t_j$ and markings $M_0, M_1, \ldots, M_{n-1}$, such that $M_i[t_0\rangle M_0[t_1\rangle M_1[t_2\rangle \ldots M_{n-1}[t_j\rangle M_n$. The sequence $t_0, t_1, \ldots t_j$ is called a *firing sequence*. Two transitions t_1 and t_2 are *mutually exclusive* if there is no reachable marking M, such that t_1 and t_2 are both enabled in M. Two transitions t_1 and t_2 are in a *structural*

conflict, if they share an input place, i.e. ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$.

For these low-level Petri nets, the tokens are indistinguishable, i.e. it is impossible to distinguish different process instances in such a net. For such reasons, there exist several types of higher-level Petri nets.

As stated by Jensen in [26], *Colored Petri nets* "allow the modeller to make much more succinct and manageable descriptions than can be produced by means of low-level nets." The *coloring* of tokens allows for specifying process data, manipulate this data and impose further conditions on the control flow. Thereby, Colored Petri nets (*CPN*) provide sufficient means for representing multiple distinct process instances within a single net.

Definition 6 (Multi-Set(cf. [26])) *A* multi-set *m, over a non-empty set S, is a function $m : S \to \mathbb{N}$, where the non-negative integer $m(s) \in \mathbb{N}$ is the number of appearances of s in the multi-set m. We write $s \in m$ if $m(s) \geq 1$.*

Definition 7 (Labeled Colored Petri net(cf. [64], [26])) *A* Labeled Colored Petri net(CPN) *is a tuple $CPN = (P, T, F, \mathcal{T}, \lambda, \Sigma, \gamma, \mu, \epsilon)$, where*

- *$(P, T, F, \mathcal{T}, \lambda)$ is a labeled net,*
- *Σ, is a finite set of* types*, called* color sets*, which are also finite and non-empty,*
- *γ, is a* color function *$\gamma : P \to \Sigma$, that associates each place with a color set,*
- *μ, is a* guard function *$\mu : T \to BooleanExpr$ that maps each transition to a predicate, and*
- *ϵ, is an* arc expression function *$\epsilon : F \to Expr$, assigning an expression to each arc that evaluates to a multi-set over the color set of the place connected by the respective arc.*

Associating the variables of the arc expressions to color values of tokens is called a *binding*. The available color values and their frequency are determined by function ϵ. Of course, these bindings along with the guard expression defined by μ, influence the enabling and firing of transitions. Further the definition of a *marking* of the net is influenced by the token colors. We will give an example later. For the formal definitions of the enabling of transitions and markings in a CPN, we refer the reader to [26]. We denote the multi-set of *enabling bindings* for transition t by $\mathsf{EB}(t)$.

Although CPNs are a powerful modeling language, they do not, by definition, explicitly provide means for specifying the time consumption of transitions. Therefore, definition 7 must be further extended. The most general type of timed Petri nets are *Stochastic Petri nets*, as described, e.g., by Marsan in [32], which allow random firing delays for

transitions. Furthermore, van der Aalst ([53]) presents a combination of timed and colored Petri nets.

Definition 8 (Labeled Timed Colored Petri net) *A* Labled Timed Colored Petri net(TCPN) *is a tuple* $TCPN = (P, T, F, \mathcal{T}, \lambda, \Sigma, \gamma, \mu, \epsilon, TS, \mathcal{D}, \delta)$*, where*

- $(P, T, F, \mathcal{T}, \lambda, \Sigma, \gamma, \mu, \epsilon)$ *is a labeled colored Petri net,*
- TS *is the time set,*
- $\mathcal{D}$ *is the set of distribution functions, where* $d \in \mathcal{D}$ *is defined as* $d : [0,1] \times TS \rightarrow TS$
- δ *is a function* $\delta : T \rightarrow 2^{TS \times TS \times \mathcal{D}}$ *that associates a transition with one or multiple distribution functions.*

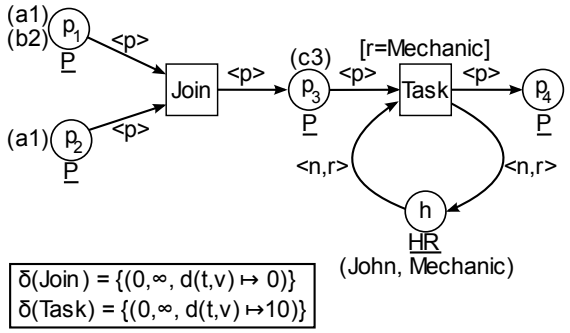
Let us have a closer look into the definitions of $TS, \mathcal{D}$, and δ. TS defines an arbitrary set of time stamps, where $\forall x, y \in TS : x + y \in TS$. We will, however, assume that $TS \subseteq \mathbb{N} \cup \{0, \infty\}$ representing an arbitrary time unit. A function $d \in \mathcal{D}$ takes a time stamp $t_1 \in TS$ and a random value $v \in [0,1]$ as inputs, and returns a time $t_2 \in TS$, with $t_1 + t_2 \geq t_1$. As these functions are assigned to transitions by function δ, they provide firing delays, where $t_1 + t_2$ defines the point in time when the transition completes firing. t_2 is the *firing delay*. In general, we denote the firing delay of a transition t with $fd(t)$.

By the definition of δ we are able to associate a transition with multiple of such distribution functions. Therefore, for a transition $t \in T$, $\delta(t)$ is a subset of $TS \times TS \times \mathcal{D}$, where a tuple (t_1, t_2, d) denotes that within the time interval $[t_1, t_2]$ function d is applied to determine $fd(t)$. This notion implies that the time intervals in $\delta(t)$ are non-interleaving, i.e. for a concrete point in time there is at most one distribution function to be applied. For all τ-transitions $t_\tau \in T$, $\delta(t_\tau) = \{(0, \infty, d : [0,1] \times TS \mapsto 0)\}$, i.e. τ-transitions have no delay.

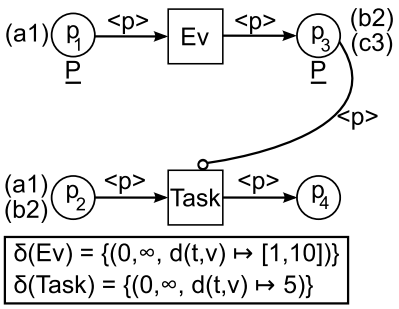
In addition to the enabling constraints imposed for CPNs, we define that a transition $t \in T$ of a timed colored Petri net, is only enabled in a marking M at a point in time $t_0 \in TS$, if $\exists (t_1, t_2, d) \in \delta(t) : t_1 \leq t_0 \leq t_2$; otherwise, t is not enabled.

As an illustrative example for the concepts defined before, Figure 2.3a depicts a simple timed colored Petri net. It represents a simple join of two branches followed by a task execution. For a better explanation, we have also labeled the places (p_1 to p_4,h).

The underlined letters below each place indicate its type, i.e., p_1 to p_4 have the same type, whereas h is of another type. Consequently, the tokens consist of different types and structure. Type P refers to a color representing process IDs; type HR refers to a color representing human resources with their name and role. The respective tokens are represented as tuples in round brackets next to the places. Arc expressions are defined



(a) A Timed Colored Petri net



(b) An Extended TCPN

Figure 2.3.: Petri net examples

in arrow brackets close to each edge. Evaluating the arc expression $< p >$ for place p_1 returns the (multi-)set $\{a1, b2\}$; for place p_1 it returns $\{a1\}$. For a valid enabling of a transition, a variable can only be bound to one value, i.e. transition $Join$ is only enabled for the binding $< p = a1 >$ and not for $< p = b2 >$. The firing of $Join$ puts a token $(a1)$ to place p_3, that already contains a token $(c3)$. Having these two tokens at place p_3, transition $Task$ is enabled twice, i.e. $\mathsf{EB}(Task) = \{< p = a1, n = John, r = Mechanic >, < p = b2, n = John, r = Mechanic >\}$.

For transition $Task$ we specify a guard condition, denoted by square brackets, which requires the role of the human resource token to equal "Mechanic". From the definition of δ, we see that $Join$ is not time consuming whereas $Task$ has a constant firing delay of ten time units.

In literature, there exist multiple approaches for defining a transition firing in a timed Petri net. Most approaches consider an atomic firing of transitions, i.e., if the transition fires, tokens are removed and produced in one immediate step. The approach of Jensen ([26]) associates a time stamp with each token indicating when this token is "ready to use". If a transition t fires, i.e. available tokens are consumed at a point in time t_1, new tokens are immediately created, having a time stamp $t_1 + fd(t)$. In [18], Ferscha requires a transition t to be enabled for the time $fd(t)$, before the firing takes place. Other approaches, as described by Zuberek in [67], assume the firing to consist of two steps, i.e., the tokens are removed from the input places of t, remain "inside" t for the time $fd(t)$, and are finally put into the output places.

Definition 9 (Extended Timed Colored Petri net) *An* Extended Timed Colored Petri net(ETCPN) *is a tuple* $ETCPN = (P, T, F, Esc, \mathcal{T}, \lambda, \Sigma, \gamma, \mu, \epsilon, TS, \mathcal{D}, \delta)$*, where*

- $(P, T, F, \mathcal{T}, \lambda, \Sigma, \gamma, \mu, \epsilon, TS, \mathcal{D}, \delta)$ *is a labeled timed colored petri net, and*

- $Esc \subseteq P \times T, Esc \cap F = \emptyset$ *is a set of* escape arcs.

Escape arcs enable the *inhibition* of transitions. A place p is an *escape place* of transition t, if there exists an escape arc $(p, t) \in Esc$. Escape arcs are visually represented by edges ending with a circle. Combining the concepts of escape arcs and colored tokens a complex transition firing mechanism is created.

As an illustration, figure 2.3b depicts a small ETCPN. Place p_3 is an output of transition Ev and at the same time an escape place for transition $Task$. For the three process instances, `a1`, `b2`, and `c3`, the following scenarios apply:

Race condition For process instance `a1`, both transitions are concurrently enabled. Transition Ev has a firing delay between 1 and 10 time units; $Task$ has a constant firing delay of 5 time units. Depending on the concrete firing delay of Ev, $Task$ fires successfully or gets preempted.

Inhibition For process instance `b2` place p_3 already contains a token. Thereby, transition $Task$ will not even start firing for this instance. We say $Task$ is *inhibited* for process instance `b2`.

Preemption We assume that transition $Task$ has already started firing for process instance `c3`, applying the two-step firing approach ([67]). In this case, the arrival of a token with the same instance ID at place p_3 leads to the immediate cancellation of the transition firing, i.e., no token will be produced for place p_4. For approaches considering atomic transition firings, *preemption* and *inhibition* are conceptually equal.

2.5. Distributed Discrete Event Simulation

In *discrete event simulation (DES)* the change of a system's state over time is represented as a sequence of *events*. Each event occurs at a discrete point in time. Between the occurrence times of two consecutive events the state of the simulated system does not change. Therefore, the simulation time is not continuously advanced by a constant step size, but to the closest point in future where the next event will occur. If multiple events are scheduled at the same point in time, they are sequentially executed. The sequential order is determined by random or by applying a prioritization scheme.

Since an event represents a change in the system's state, the occurrence of an event usually leads to the scheduling of new future events. For example, when simulating a

simple web server, an event representing an incoming request will eventually lead to an event representing the sending of the respective response. In addition, the firing of an event may invalidate other, already scheduled, events, e.g. if two events require exclusive access to a shared resource.

From the implementation point of view, a simulation engine following the DES paradigm provides the following data structures:

Clock For tracking the simulated time a clock is required. The clock c is advanced in discrete, usually unequal, steps using the function $\mathsf{advance}(c, t)$, where t is a time stamp.

Event An event describes the change of the system's state at a specific point in simulated time. Therefore, an event ev specifies an action, denoted by $\mathsf{action}(ev)$, and a time stamp, denoted by $\mathsf{time}(ev)$. Furthermore, a function $\mathsf{isValid}(ev)$ can be used do determine if a respective event is still valid with respect to the current state.

Event List The event list is a sorted list of events similar to a priority queue. The primary metric for sorting the events is their time stamp. For events with equal time stamp further metrics might be applied; otherwise they are randomly enqueued. For an event list evl and an event ev the following operations are defined:

- $\mathsf{schedule}(evl, ev)$, inserts the event into the event list.
- $\mathsf{dequeue}(evl)$, removes and returns the first event in the event list, i.e. the event with the nearest time stamp.
- $\mathsf{deschedule}(evl, ev)$, removes the event from the event list. This function is used if an event got invalidated.
- $\mathsf{isEmpty}(evl)$, checks if the list is empty.

Using these data structures, the basic algorithm for discrete event simulation is sketched in algorithm 1.

Although DES is a powerful approach, large simulation models generating a vast amount of events require much computational resources and a probably unaffordable amount of real world time for their simulation. To overcome these obstacles, a lot of investigations have been conducted in the field of *parallel* and *distributed* discrete event simulation. As explained by Ferscha ([16]), the term *parallel* refers to a multi-processor approach where synchronization is achieved by the use of shared memory. In contrast, the term *distributed* refers to a multi-processor approach where synchronization is manually achieved using a synchronization protocol. With respect to this classification, this thesis is con-

Algorithm 1: Basic DES algorithm

Prerequisites: evl is an empty event list; clock is a clock; endTime is the time until the system is simulated

```
scheduleInitialEvents (evl) ;
while ¬ isEmpty (evl) ∧ clock < endTime do
    ev ⟵ dequeue (evl) ;
    advance (clock, time (ev)) ;
    perform action (ev) ;
    scheduleNewEvents (evl) ;
    descheduleDevalidatedEvents (evl) ;
end
```

cerned with *distributed discrete event simulation(DDES)*, as it is the more general case also allowing the physically distributed execution on multiple machines. We will, however, use both terms interchangeably with the meaning of distributed simulation.

In order to get simulated in a distributed manner, the simulation model must be partitioned. The partitions are simulated by so-called *logical processes*(LP) that are communicating using messages. A reasonable partitioning of the simulation model exploits its parallelism as good as possible, i.e. if two parts of the simulation model are completely independent of each other, they should be placed in different logical processes. Moreover, the partitioning is influenced by the chosen synchronization protocol.

The purpose of a synchronization protocol is to ensure the semantical and chronological order of the simulated events, thereby ensuring the correctness of the distributed simulation result. A compact overview on existing protocols is given by Zarei in [66]; a more explanatory summary can be found in [16]. Over time, four basic categories of strategies have evolved.

Conservative Protocols The philosophy behind conservative synchronization strategies is to ensure that after simulating an event it is not possible to receive a message, i.e. from another logical process, having a smaller time stamp. This chronological order is preserved by blocking the execution of a logical process until it is *safe to process* the next event. The original approaches have been described by Bryant, Chandy, and Misra ([6, 7]). Conservative protocols are straight-forward in their implementation. Their drawback, however, is they are too pessimistic and, thereby, obstruct the full exploitation

of model parallelism. In addition, if there are circular dependencies between the logical processes, the simulation might deadlock when multiple processes are waiting on each other. For the latter problem, distributed deadlock detection and recovery (cf. e.g. [8]) and deadlock avoidance (cf. e.g. [34, 38]) algorithms have been proposed.

Optimistic Protocols In contrast to the conservative approaches, in optimistic protocols, the logical processes continue simulating despite the probability of the occurrence of a message with a time stamp "in the past". If such a message occurs, the simulation is *rolled back* to the point in time this, so-called, *straggler message* occurred. These rollbacks require the simulation engines to persist past states of the system in order to restore them if necessary. Therefore, although optimistic approaches allow for a better exploitation of model parallelism, they are much more resource consuming than conservative approaches, and, in case of rollbacks across multiple logical processes, generate an additional communication overhead. The original approach for these protocols has been presented by Jefferson in [25].

Hybrid Protocols To avoid tremendous rollback cascades while still reducing the blocking times of logical processes, hybrid protocols, incorporating conservative and optimistic aspects, have been developed. Examples are proposed, e.g., by Steinman ([50]) and Dickens ([12]).

Adaptive Protocols As an optimization of the general hybrid approaches, adaptive protocols try to adapt the level of optimism while the simulation is running. An example for this class has been published by Ferscha and Chiola in [15].

The distributed simulation of Petri nets and their extension of timed Petri nets has been thoroughly investigated by Chiola and Ferscha in [9] and [10]. A logical process for such simulation consists of three parts: a spatial *region* of the Petri net, a *simulation engine* implementing the synchronization protocol, and a *communication interface* for sending messages to and receiving them from other LPs. The communication interfaces can be connected using directed *communication channels* for exchanging messages.

3. Simulation Scenarios and Their Formal Representation

The current and the following chapter describe our approach to the concurrent simulation of multiple related processes. After a short outline on the conceptual architecture in section 3.1, section 3.2 defines our notion of simulation scenarios. Based thereon, we present a mapping of simulation scenarios onto timed colored Petri nets in section 3.3.

As mentioned in section 2.1, Petri nets are not the only formalism at hand, that can be used for the formal representation of business processes. However, Petri nets have been proven to be a well-suited and well-understood formalism for the analysis and simulation of business processes ([56]). Further, from a conceptual point of view, the partitioning of the simulation model, for its distributed execution, is independent of the concrete formalism. Therefore, and due to previous experiences using them, we chose Petri nets as the formalism for our approach.

3.1. Conceptual Architecture

In [29], Krumnow proposes an architecture blueprint for a business process simulation engine. Although, the general structure also applies for our approach, we need to refine this architecture for the following reasons.

First, disregarding the fact that the blueprint is tailored towards the use of BPMN as a modeling language, process models are the only considered model type. This is not sufficient in our case. Besides considering organizational models, the simulation scenario is a model itself. Second, the blueprint represents a solution for a single-engine simulation. As we seek the distribution of simulation among multiple simulation engines, we have to introduce a component that partitions the generated Petri net. Third, since there will be multiple engines, we need a component for initiating the simulation and collecting the results for the user. Considering these adaptions, we receive the conceptual

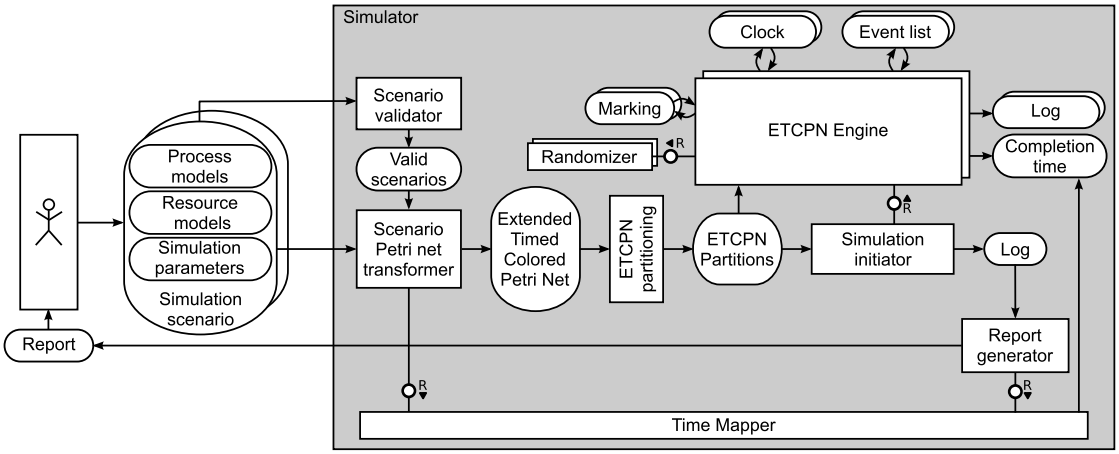


Figure 3.1.: Conceptual simulation architecture as FMC block diagram [28]

architecture depicted in figure 3.1.

As we can see from the figure, a simulation scenario consists of process models, resource models, and also incorporates the simulation parameters. This is not the case for the blueprint in [29], where they are separately stored. During the transformation of a valid scenario into a Petri net, the times configured by the user need to be mapped from the user-friendly format to the format of the simulation engine, which is a simple integer value. See section 3.3 for details on the mapping and section 3.2 for time modeling. Depending on the size and structure of the generated net, it is split into one or multiple *partitions* (cf. chapter 4). Each partition is simulated by a simulation engine following the DES approach. Each engine has its own local clock, event list, marking, and log. The *simulation initiator* is responsible for distributing the partitions to the engines and collecting their simulation results. After the complete log data is gathered, a report is generated and returned to the user. As the report should contain user-friendly time representations, the *time mapper* is used again.

3.2. Simulation Modeling

Before we can use our models as an input for the simulator, we need to define the allowed modeling constructs. Beginning with section 3.2.1, we informally describe the supported modeling elements and simulation parameters. Based thereon, we give formal definitions in section 3.2.2. The modeling requirements $\mathsf{MR1} - \mathsf{MR5}$, stated in section 1.1, serve as guidelines for our definitions.

3.2.1. Models and Their Configuration

Recalling definition 2 of a business process, a business process is more than a pure causal order of tasks. Instead we need to consider the process' environment, where we distinguish human (organizational environment) and non-human (technical environment) resources. In addition, several timing constraints have to be regarded.

Resource Modeling

For the scope of this work, the term *resource* refers to any physical entity that is required, consumed, or produced in the context of the process execution. As mentioned above, we distinguish human and non-human resources.

Non-human resources are characterized by their name and an amount. Examples are simple things like nails, but also complex systems like machines. As these two examples show, we can further distinguish between reusable and consumed or produced non-human resources. Due to their simple nature, non-human resources are modeled using lists, where each entry specifies the resource's name and its available amount.

Human resources are more complex. From an organizational perspective, humans are grouped into teams and organizational roles, which are themselves part of more coarse-grained structures. This structural information can be represented using, e.g., organizational charts[1], satisfying the claim of requirement MR1 for a graphical modeling language. In addition to the organizational structure, we attribute human resources with working times. For this work, we assume that the working times are specified on the basis of weekdays and do not change from one week to another, i.e. a working time specified for Mondays, will be the same on every - simulated - Monday.

There are different possibilities describing the behavior of human-resources during their working times. First, they could arbitrarily choose the next task to perform, disregarding its possible duration and thereby risking extra hours. Second, they could arbitrarily choose a task, but pause its execution at the end of the working time, i.e., there will be no overtimes. Third they could choose the next task according to the amount of working time left.

Of course, there are a lot of more aspects human resources might be attributed with, e.g. technical or functional skills, which will also affect the simulation. In addition, there are

[1] `http://en.wikipedia.org/wiki/Organigram`

several approaches for capturing human behavior in simulation models. For the focus of this thesis, however, we restrict the configuration options for human resources to their working times and refer the interested reader to, e.g., [35], [5], and [60].

Process Modeling

There exist several graphical business process modeling languages. Common examples are event-driven process chains (EPC) ([27]), YAWL ([54]), and the Business Process Model and Notation (BPMN) ([22]). Although they differ with respect to their graphical representation and their expressiveness, they commonly support a large set of core modeling constructs. For the following considerations, we take BPMN as a modeling notation.

As BPMN is a very powerful language, providing also semantically complex constructs, we restrict the set of allowed constructs, to a reasonable but still expressive set. For this thesis the following elements are considered: *Tasks, Call Activities/Collapsed Sub-Processes, Start Events, End Events, Parallel Gateways, Exclusive Gateways, Event-based Gateways, Sequence Flows*, and *Message Flows*. As the simulation is based on resources, the support for *Pools* and *Lanes* is inherently given. The chosen set implies the focus on process orchestrations. Furthermore, the following constraints are imposed:

- The allowed types for events are: *Standard, Message, Timer*, and *Link*.
- *Activities* can be simple or looping.
- *Sub-Processes* must not have any *Intermediate Events* attached to their boundaries. This constraint explicitly precludes the *Cancel Case* control flow pattern ([46]).
- *Message Flow* arcs are only connecting events with other events; i.e. their source and target nodes are no *Tasks* or *Pools*.
- *Gateways* are either split nodes or join nodes, i.e. they either have multiple incoming or multiple outgoing *Sequence Flow*.
- Except for gateways, all nodes have at most one incoming and at most one outgoing *Sequence Flow* arc.

For ensuring the latter two constraints, process models containing diverging constructs can be normalized by the introduction of additional nodes. For the sake of simplicity, we claim that each process has exactly one *End Event* and exactly one *Start Event*, and require the process to be *safe* and *sound*, i.e., it neither contains a deadlock nor a lack of synchronization ([22]). These are important properties for determining the termination of a process instance. However, they explicitly preclude the *Implicit Termination* control

flow pattern ([46]). For detailed definitions of *soundness* and *safety* in the context of Petri nets we refer the reader to [36]. As the BPMN specification ([22]) refers to the concept of tokens, these definitions similarly apply to BPMN process models.

For processes connected by *Message Flow* arcs, we require that it is clear, at any time, which concrete process instances are communicating. This property is similar to the local enforceability of process choreographies ([11]). Figure 3.2 illustrates the properties of *soundness*, *safety*, and message exchanges that are valid w.r.t. our simulation.

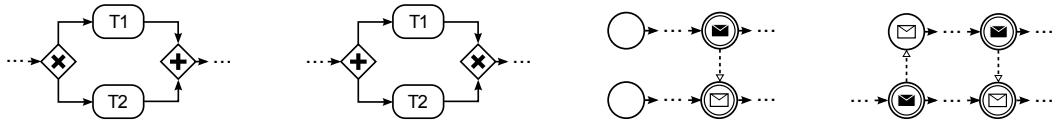


Figure 3.2.: (a) a deadlock example, (b) a lack of synchronization, (c) an invalid communication, and (d) a valid communication

In figure 3.2a, we have an exclusive choice between tasks *T1* and *T2*. However, the gateway joining both branches requires them both to be executed, i.e. the gateway cannot be executed which results in a *deadlock*. In figure 3.2b, the concurrently executed branches are not properly joined by the *Exclusive Gateway*, which is a *lack of synchronization*. Figure 3.2c shows a message exchange between two processes. Since both processes have been started at the time the message is delivered, we cannot automatically identify which process instances are communicating, i.e. this message exchange is invalid for simulation. In contrast, figure 3.2d depicts a valid message exchange. As the first message creates the instance of the upper process, it is obvious to which instance the response, i.e. the second message, is delivered.

Using the defined set of elements and the imposed constraints we are able to express all control flow patterns claimed by modeling requirement MR2.

We assume that a process can be uniquely identified by a respective global identifier. Further, each element of a process can be uniquely identified within its process by a respective identifier. The elements of process models can be attributed with the following aspects:

Execution/Wait Times *Tasks* are annotated with execution times. They can be specified as a constant value or by using a distribution function. Receiving *Intermediate Events* are annotated with durations. If the event is a *Timer Event* only constant

durations are allowed; otherwise, distribution functions can be used.

Branching Probabilities For *Exclusive Gateways* each outgoing *Sequence Flow* is annotated with a probability ranging from 0.0 to 1.0, indicating the likeliness that the respective branch is chosen. The probabilities for a single gateway must sum up to 1.0.

Exception Probabilities Attached *Intermediate Events* are annotated with a probability ranging from 0.0 to 1.0, indicating the likeliness that the event will interrupt the task it is attached to.

Resource Consumption For *Tasks* it can be specified which and how many non-human resources are consumed, or used, by this task.

Resource Production Analogous to the resource consumption, it can be specified that a *Task* returns resources after execution or newly creates non-human resources.

Performer In addition to non-human resources, human resources can be specified to be the performer of a *Task*. A task can have at most one performer. Performers can be specified on the basis of single persons, roles or organizational units. The specification can be made for each single task, as well as for the *Pool* or *Lane* in which the task resides. Supported workflow resource patterns are *Direct Allocation* and *Role-based Allocation* (cf. [45]).

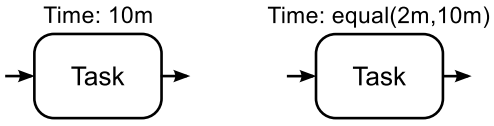
Case Handling If subsequent tasks of a process instance have to be performed by the same human resource, which corresponds to the concept of *case handling* ([59]), tasks can be designated to start or complete such a *case*.

Instantiation Distributions *Start Events* of processes that are not called from other processes must be attributed with instance creation constraints. Such constraints are composed of a time frame, e.g. Mondays from 9 a.m. to 5 p.m., and a distribution function or constant time value, e.g. one hour. Analogous to the working times of human resources, we assume that the configurations stay constant over all simulated weeks.

Modeling Time

Modeling requirement MR3 describes versatile usages of time related artifacts, where we distinguish time durations and time instants. From a user perspective, modeling time must be available in a comprehensible format.

Values of durations, i.e. the execution time of tasks, the delays of events, or the frequency at which new instances are created, are specified as combinations of days, hours, minutes, and seconds in the format recommended by ISO standard 8601 [1]. If a task lasts one day, two hours, three minutes, and four seconds, the user annotates it with the value `1d2h3m4s`. Figure 3.3 gives examples for specifying a duration. In Figure 3.3a a constant value is specified, i.e. each task execution lasts ten minutes. In Figure 3.3b an equally distributed duration is set, i.e., each task execution lasts between two and ten minutes where each value is equally likely to occur. Please note that the smallest time units are seconds. Using distribution functions, the user is enabled to model varying execution times. Obviously, there are further distribution functions. However, for the scope of this work, we restrict them to equal distributions, while remarking that the created prototype can be easily extended by further functions.



(a) (b)

Working Times

At ...	From	To
weekdays	9:00	17:00
Saturdays	8:00	12:00

(c)

Instance Creation

At ...	From	To	Frequency
weekdays	9:00	17:00	1h
Saturdays	8:00	12:00	equal(1h, 2h)

(d)

Figure 3.3.: Time modeling examples: (a) A task with constant execution time, (b) a task with equally distributed execution time, (c) a working time configuration, and (d) and instance creation configuration

The specification of time instants is required for the specification of working times and the time frames during which process instances can be created. As mentioned earlier, these values are specified on the basis of weekdays. For time instants the usual 24-hour time format is used. Examples for working time and instance creation configurations are shown in figures 3.3c and 3.3d. The latter denotes the following facts: At weekdays, i.e. Monday to Friday, between 9 a.m. and 5 p.m. every hour a new instance is created. Further, at Saturdays between 8 a.m. and noon every one to two hours a new instance is spawned.

3.2.2. Formal Model for Simulation Scenarios

Within the following sections, the three essential parts of a simulation scenario - processes, human resources, and non-human resources - will be defined. Subsequently, we give a general definition of simulation scenarios and their validity constraints.

Resources

Since there is a major difference between both concepts, human and non-human resources are defined separately.

Definition 10 (Organizational Model) *An organizational model is a tuple* $OM = (N, E)$*, where*

- N *is a set of nodes that can be partitioned into disjoint sets of organizational units* N_U*, organizational roles* N_R*, and persons* N_P*, and*
- $E \subseteq (N_U \times (N_U \cup N_R)) \cup (N_R \times N_P)$ *is a set of directed edges, indicating that one organizational entity is part of another entity.*

As already stated in section 3.2.1, definition 10 is derived from the structure of organizational charts, which provide a basic graphical notation for modeling organizational structures. By using these charts it is possible to specify one or more roles for a specific person. Figure 3.4c depicts an exemplary organizational chart. For this example, we find: $N_U = \{\text{Retail Corp.}\}, N_R = \{\text{Worker WH}, \text{Worker OM}, \text{Inspector}\}$, and $N_P = \{\text{Alice}, \text{Bob}, \text{Charles}, \text{Eric}, \text{James}, \text{Steve}\}$.

Although these models provide basic information on which and how many resources are available, we have to add a construct for capturing the working times of each person.

Definition 11 (Human Resource Model) *Let* TS *be a set of timestamps. A human resource model is a tuple* $HRM = (OM, \mathsf{WTime})$*, where*

- OM *is an organizational model, and*
- $\mathsf{WTime} : N_P \rightarrow 2^{TSxTS}$ *is a function assigning a set of time intervals to each person in* OM*. The time intervals indicate the working times of the person, i.e., the time the specific person is available for performing tasks.*

In order to support other human aspects, like skills (see section 3.2.1 for examples), definition 11 might be extended by further functions.

Definition 12 (Non-Human Resource) *A non-human resource is a triple* $NHR = (name, quantity, type)$*, where*

- *name is a descriptive name of the resource,*
- *quantity is the available amount of this resource, and*
- *type* $\in \{TOOL, MATERIAL\}$ *is the resource's type, indicating if it is consumed or can be re-used.*

Since there will generally be more than one non-human resource, we introduce the concept of a warehouse.

Definition 13 (Warehouse) *A warehouse is a set of non-human resources, where the name of each resource is unique within the warehouse.*

Processes

Inspired by the definitions of *Process models* by Weske ([64]) and *Core BPMN Processes* by Dijkman ([13]), we define the concept of *Simulation Business Processes*, implying a set of allowed modeling constructs.

Definition 14 (Simulation Business Process) *A simulation business process is a tuple* $P_{SIM} = (\mathcal{O}, \mathcal{F}, \mathsf{TType}, \mathsf{GType}, \mathsf{EType}, \mathsf{EDir}, \mathsf{Excp})$*, where:*

- $\mathcal{O}$ *is a set of objects which can be partitioned into disjoint sets of activities* $\mathcal{A}$*, events* $\mathcal{E}$*, and gateways* $\mathcal{G}$*.*
- $\mathcal{A} = \mathcal{T} \cup \mathcal{S}$*, where* $\mathcal{T}$ *is a set of tasks and* $\mathcal{S}$ *is a set of sub-process invocations.* $\mathcal{T}$ *and* $\mathcal{S}$ *are disjoint, i.e.* $\mathcal{T} \cap \mathcal{S} = \emptyset$*.*
- $\mathcal{E} = \mathcal{E}_S \cup \mathcal{E}_I \cup \mathcal{E}_E$*, where* $\mathcal{E}_S$ *is a set of start events,* $\mathcal{E}_I$ *is a set of intermediate events, and* $\mathcal{E}_E$ *is a set of end events.* $\mathcal{E}_S$*,* $\mathcal{E}_I$*, and* $\mathcal{E}_E$ *are mutually disjoint.*
- $\mathcal{F} \subseteq \mathcal{O} \times \mathcal{O}$ *is the control flow relation, i.e. a set of directed arcs connecting objects. Let* ${}^\bullet\mathcal{F}(o) = \{x | (x, o) \in \mathcal{F}\}$ *and* $\mathcal{F}^\bullet(o) = \{x | (o, x) \in \mathcal{F}\}$ *be the sets of input and output objects of an object* $o \in \mathcal{O}$*. If* $o \in \mathcal{A} \cup \mathcal{E}_I$ *then* $|{}^\bullet\mathcal{F}(o)| = 1$ *and* $|\mathcal{F}^\bullet(o)| = 1$*; if* $o \in \mathcal{E}_S$ *then* $|{}^\bullet\mathcal{F}(o)| = 0$ *and* $|\mathcal{F}^\bullet(o)| = 1$*; if* $o \in \mathcal{E}_E$ *then* $|{}^\bullet\mathcal{F}(o)| = 1$ *and* $|\mathcal{F}^\bullet(o)| = 0$*.*
- $\mathsf{TType} : \mathcal{T} \rightarrow \{\texttt{SIMPLE}, \texttt{LOOP}\}$ *is a function assigning a type to a task, where* `SIMPLE` *implies exactly one execution per task activation and* `LOOP` *implies zero or more executions on a single activation in sequence.*
- $\mathsf{GType} : \mathcal{G} \rightarrow \{\texttt{XORSplit}, \texttt{XORJoin}, \texttt{ANDSplit}, \texttt{ANDJoin}, \texttt{EventSplit}\}$ *is a function assigning a control flow construct to each gateway. If* $\mathsf{GType}(g) \in \{\texttt{ANDSplit}, \texttt{XORSplit}, \texttt{EventSplit}\}$ *then* $|{}^\bullet\mathcal{F}(g)| = 1$ *and* $|\mathcal{F}^\bullet(g)| > 1$*; otherwise* $|{}^\bullet\mathcal{F}(g)| > 1$ *and* $|\mathcal{F}^\bullet(g)| = 1$*.*
- $\mathsf{EType} : \mathcal{E} \rightarrow \{\texttt{Standard}, \texttt{Timer}, \texttt{Message}, \texttt{Link}\}$ *is a total function assigning a type*

to an event, with

$$\mathsf{EType}(e) \in \begin{cases} \{\texttt{Standard}, \texttt{Timer}, \texttt{Message}\} & \text{if } e \in \mathcal{E}_S \\ \{\texttt{Standard}, \texttt{Timer}, \texttt{Message}, \texttt{Link}\} & \text{if } e \in \mathcal{E}_I \\ \{\texttt{Standard}, \texttt{Message}\} & \text{if } e \in \mathcal{E}_E \end{cases}$$

- $\mathsf{EDir} : \mathcal{E}_I \rightarrow \{\texttt{IN}, \texttt{OUT}\}$ *is a function assigning a direction to an intermediate event, where* `IN` *indicates the reception and* `OUT` *the triggering of an event.*
- $\mathsf{Excp} : \mathcal{E}_I \rightsquigarrow \mathcal{T}$ *is a partial function assigning an intermediate event to a task, such that the occurrence of the event will interrupt the execution of the task.*

Definition 14 implies the set of BPMN constructs, outlined in section 3.2.1, which can be used for modeling processes that can be simulated by our engine. Although the definition is tailored to BPMN, it can be applied to other modeling languages as well. For identifying processes and their elements, let $pid(p)$ denote the globally unique identifier of a simulation business process p and let $eid_p(e)$ denote the identifier of element e within process p.

In order to provide sufficient details for a reasonable simulation a *Simulation Business Process Model* must attribute simulation business processes with the additional information described in section 3.2.1. For the following definition, let TS be the set of time stamps and let $\mathcal{D}$ be a set of distribution functions. A function $d \in \mathcal{D}$ takes a time stamp $t_1 \in TS$ and a random value $v \in [0, 1]$ as inputs, and returns a time $t_2 \in TS$, with $t_1 + t_2 \geq t_1$. Please note that this definition of $\mathcal{D}$ is equal to its definition for timed colored Petri nets in section 2.4.

Definition 15 (Simulation Business Process Model) *Let NHR be a set of non-human resources, and HR a set of human resources. A* simulation business process model *is a tuple $PM_{SIM} = (\mathcal{P}_{SIM}, \mathcal{M}, \mathsf{Time}, \mathsf{Prob}, \mathsf{ExcpProb}, \mathsf{Cons}, \mathsf{Prod}, \mathsf{Perf}, \mathsf{Loop}, \mathsf{InstCrea}, \mathsf{Case}, \mathsf{Call})$, where*

- *$\mathcal{P}_{SIM}$ is a set of simulation business processes with elements $P_1, P_2, \ldots, P_n$. Based thereon, we define sets $X_{SIM} = \bigcup_{i=1}^{n} X_i$, where X_i is the respective element in P_i, i.e. $\mathcal{A}_{SIM} = \bigcup_{i=1}^{n} \mathcal{A}_i$ is the set of all activities in $\mathcal{P}_{SIM}$.*
- *$\mathcal{M} : \mathcal{E}_{SIM}^{M^S} \times \mathcal{E}_{SIM}^{M^R}$ is the communication flow relation, where $\mathcal{E}_i^{M^S} = \{e | e \in \mathcal{E}_i \wedge \mathsf{EType}_i(e) = \texttt{Message} \wedge \mathsf{EDir}_i(e) = \texttt{OUT}\}$ is the set of sending message events of process P_i and $\mathcal{E}_i^{M^R} = \{e | e \in \mathcal{E}_i \wedge \mathsf{EType}_i(e) = \texttt{Message} \wedge \mathsf{EDir}_i(e) = \texttt{IN}\}$ is the set of receiving message events of process P_i.*
- *$\mathsf{Time} : \mathcal{A}_{SIM} \cup \mathcal{E}_{SIM} \rightsquigarrow \mathcal{D}$, is a partial function assigning a distribution function to a task or event. For tasks the distribution function returns the duration of the*

task; for events the time until the event is received. Events are only part of the function's domain if they cannot be simulated otherwise.

- $\mathsf{Prob} : \mathcal{F}_{G_X} \rightarrow [0,1]$, *is a function assigning a probability to an edge, where* $\mathcal{F}_{G_X}$ *is the subset of edges exiting exclusive split gateways, i.e.* $\mathcal{F}_{G_X} = \{(g,x)|(g,x) \in \mathcal{F}_{SIM} \wedge g \in \mathcal{G}_{SIM} \wedge \mathsf{GType}_{SIM}(g) = \texttt{XORSplit}\}$. *For each gateway, the probabilities of all outgoing edges must sum up to 1.*
- $\mathsf{ExcpProb} : \mathcal{E}_{I_{SIM}} \rightsquigarrow [0,1]$ *is a function assigning a probability to an attached intermediate event. The probability denotes the likeliness, that the event will occur during the task's execution.*
- $\mathsf{Cons} : \mathcal{A}_{SIM} \rightsquigarrow 2^{NHR \times \mathbb{N}}$ *is a partial function describing the resource consumption of an activity. For an activity* $a \in \mathcal{A}_{SIM}$, $\mathsf{Cons}(a)$ *returns a set of pairs indicating which and how many resources this activity consumes. These resources along with their cardinality are required to start one execution of the respective activity.*
- $\mathsf{Prod} : \mathcal{A}_{SIM} \rightsquigarrow 2^{NHR \times \mathbb{N}}$ *is a partial function describing the resource production of an activity. For an activity* $a \in \mathcal{A}_{SIM}$, $\mathsf{Prod}(a)$ *indicates which and how many resources are newly, or again, available after the activity has completed one execution run.*
- $\mathsf{Perf} : \mathcal{A}_{SIM} \rightsquigarrow HR$ *is a partial function assigning a human performer to an activity. As for* Cons *the availability of the performing human resource is essential for starting the execution.*
- $\mathsf{Loop} : \mathcal{T} \rightsquigarrow \{l | l : [0,1] \rightarrow \mathbb{N}\}$ *is a partial function assigning a distribution function to tasks* $t \in \mathcal{T}$ *with* $\mathsf{TType}(t) = \texttt{LOOP}$ *to speficy the number of task executions per task activation.*
- $\mathsf{InstCrea} : \mathcal{E}_{S_{SIM}} \rightsquigarrow 2^{TS \times TS \times \mathcal{D}}$ *is a function associating start events with distribution functions to specify the instance creation rates depending on distinct time frames.*
- $\mathsf{Case} : \mathcal{T}_{SIM} \rightsquigarrow \{\texttt{START}, \texttt{END}\}$ *is a partial function assigning a case configuration to tasks. Thereby, a task* $t \in \mathcal{T}_{SIM}$ *is specified to start or complete a case.*
- $\mathsf{Call} : \mathcal{S}_{SIM} \rightsquigarrow \mathcal{E}_{S_{SIM}}$ *is a partial function assigning a start event* $e_s \in \mathcal{E}_{S_{SIM}}$ *to a sub-process invocation* $s \in \mathcal{S}_{SIM}$, *denoting that simulating* s *is propagated to the simulation of* e_s.

Simulation Scenarios and Their Validity

We integrate the concepts defined above into the definition of a simulation scenario.

Definition 16 (Simulation Scenario) *A simulation scenario is a tuple* $\mathcal{S} = (\mathcal{PM},$

$\mathcal{HR}, \mathcal{W}, \mathsf{start}, \mathsf{end})$, *where*

- $\mathcal{PM}$ *is a non-empty set of simulation business process models,*
- $\mathcal{HR}$ *is a set of human resource models*
- $\mathcal{W}$ *is a warehouse, and*
- start *and* end *define the start and end of simulated time, with* $\mathsf{start} < \mathsf{end}$.

Let $\mathcal{S} = (\mathcal{PM}, \mathcal{HR}, \mathcal{W}, \mathsf{start}, \mathsf{end})$ be a simulation scenario. For convenience, let $\mathcal{PM}_i$ denote an element of $\mathcal{PM}$, and $\mathcal{HR}_k$ an element of $\mathcal{HR}$. To be a valid input to a simulation engine a scenario must fulfill the following constraints:

- All simulation business processes have exactly one end event and exactly one start event. Further, they are sound, safe, and message exchanges are valid for simulation. See section 3.2.1 for details and examples on these properties.
- For each human performer of activities in any $\mathcal{PM}_i$, there must be a node in the organizational model of any $\mathcal{HR}_k$ representing this human resource. Remember that the set of performers is defined by function Perf of each $\mathcal{PM}_i$.
- For each non-human resource associated with an activity in any $\mathcal{PM}_i$, the warehouse $\mathcal{W}$ must contain a resource of the same name. The used non-human resources are defined by functions Cons and Prod of each $\mathcal{PM}_i$.
- For each sub-process invocation s in any $\mathcal{PM}_i$, exactly on of the following must hold:
  - s is part of the domain of function Call, or
  - s is at least part of the domain of function Time of the respective process $\mathcal{PM}_i$.

  This constraint implies that there is either another process that can be simulated when s is invoked, or s can be simulated like an ordinary task by using the given function values.
- A single organizational entity occurs in exactly one $\mathcal{HR}_k$, i.e. there are no semantic intersections or conflicts between the elements of $\mathcal{HR}$.
- For each receiving event e_r in any $\mathcal{PM}_i$, i.e. start events and receiving intermediate events, exactly one of the following must hold:
  - There exists a throwing event e_t in any $\mathcal{PM}_i$, that corresponds to e_r, i.e. the simulation of e_t leads to the simulation of e_r, or
  - e_r is part of the domain of function Time of the respective process model $\mathcal{PM}_i$, or
  - if e_r is attached to a task, it is part of the domain of function $\mathsf{ExcpProb}$ of the respective process model $\mathcal{PM}_i$.

Applying this constraint ensures that any receiving event is potentially enabled to be received during the simulation.

3.3. Mapping Simulation Scenarios to Petri Nets

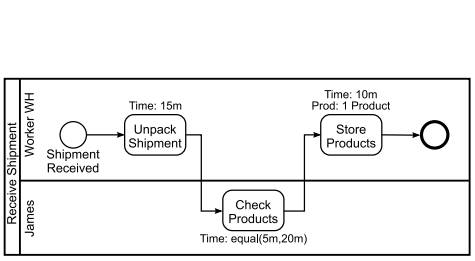
Based on the definitions of processes and resources, we define a mapping of *simulation scenarios* onto the formalism of extended timed colored Petri nets. For illustrating this mapping, we employ the example scenario depicted in figure 3.4. It consists of two processes *Receive Order* and *Receive Shipment* in figures 3.4a and 3.4b, an organizational chart displaying six human resources paired to three roles in figure 3.4c, the initial inventory in figure 3.4d, the instance creation configuration depicted in figure 3.4e, and the working time specification in figure 3.4f.

The two business process models capture the different relations that can exist. On the one hand, there are *hierarchical* relations via sub-process invocations. An example is the call activity *Order at Producer* in figure 3.4b. On the other hand, we find resource relations. As we can see from figure 3.4c, James is part of the role *Inspector*. As one process is referring to the role and the other one to James, both processes are competing for the human resource *James*, assuming there are sufficient process instances. We further observe a resource relation between the tasks *Store Products* and *Pack Shipment*, as both employ to the resource *Product*.

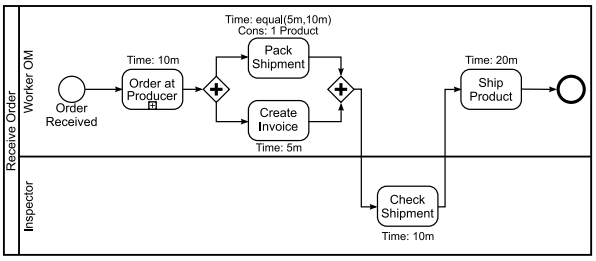
The general mapping algorithm can be divided into the following steps:

1. Map all simulation business process models to one single extended timed Petri net. (cf. section 3.3.2)
2. Add human and non-human resources to the net, while respecting the combination of organizational structure and performers for human resources. (cf. section 3.3.3)
3. Facilitate the net created in steps 1 and 2.

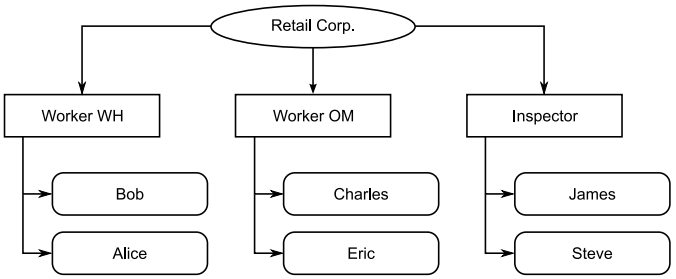
As step 1 is showing, we decide to map all processes to a single Petri net. Thus, all process instances will be represented in this single net. This is a straight-forward and convenient decision. First, the ETCPN constructs, especially colored tokens, are sufficiently expressive to represent resources and distinguish between different process instances, even if they are running in the same net. Second, the duplication of the net for multiple process instance is easily exceeding the available memory. Third, as the processes are sharing resources, such duplication leads to the duplication of some *resource places*, i.e. places containing tokens that represent human resources. For those places computational



(a) Receive Shipment



(b) Receive Order



(c) Organizational Structure

Inventory

Resource name	Amount
Product	20

(d) Initial Inventory

InstCrea(Shipment Received)

At...	From	To	Frequency
... weekdays	9:00	17:30	1h
... Saturdays	8:00	11:30	2h

InstCrea(Order Received)

At...	From	To	Frequency
... weekdays	9:00	18:30	equal(30m, 1h)
... Saturdays	8:00	12:30	2h

(e) Instance Creations

WTime(Retail Corp.)

At...	From	To
... weekdays	9:00	17:30
... Saturdays	8:00	13:00

(f) Working Times

Figure 3.4.: An exemplary simulation scenario

overhead is necessary in order to ensure consistency among their multiple occurrences. Fourth, the resulting single net is explicitly - meaning structurally - showing all relations between the processes of the simulation scenarios.

3.3.1. Mapping Time

The user-friendly time format is unnecessary complex when it comes to simulation. As shown in the conceptual architecture in section 3.1, we employ a time mapper for transforming times between the user's time representation and a format suitable for simulation. Within the simulation engine the only time unit is a second, which is the smallest unit that can be configured within the processes. Therefore, the simulation time is represented as a simple integer value, denoting the virtual, i.e. simulated, time that has already elapsed in seconds. The simulation always starts at a virtual time of 0, representing the start time value configured for the scenario. This is, if the simulated time should start at Monday, October 31, 2011, 8:00 a.m. this time stamp is the *user-friendly representation* of value 0.

Transforming durations is straightforward, since this is the usual conversion of minutes, hours, and days into seconds. If distribution functions are specified, their parameters are simply replaced by the resulting values.

Transforming time instants, however, takes a little more effort. Their representation in virtual time is always computed with respect to the configured start time of the simulation, i.e. if the simulated time should start at 8:00 a.m. the representation of 9:00 a.m. at the same day is 3600, i.e. one hour after start; if the simulated time starts already at 7:00 a.m. the result would be 7200.

Working times and time frames for process instance creation must be converted along with their repetitive occurrence in each simulated week. A naive approach would be to explicitly compute them for the whole simulated time. However, as their frequency of occurring is constant, i.e. weekly, this process can be simplified using the *modulo* operator. Representing a week as seconds is $7(\text{days}) \times 24(\text{hours}) \times 60(\text{minutes}) \times 60\text{seconds} = 604800$ seconds.

Mapping the working time configuration of figure 3.4f, assuming the start of simulated time at Monday, October 31, 2011, 8:00 a.m., results in table 3.1. For the five weekdays, five lines are added to the table; the last line represents Saturdays. Interpreting the table with respect to a point in simulation time t_s, the human resources are available

start	end	mod
3600	32400	604800
90000	118800	640800
176400	205200	604800
262800	291600	640800
349200	378000	604800
432000	446400	640800

Table 3.1.: Transformed working times of figure 3.4f

if we find a pair $(start, end)$, such that $start \leq t_s(\mod 604800) \leq end$. Analogous transformations apply for the time frames in figure 3.4e.

3.3.2. Mapping Simulation Business Process Models

Similar to the mappings presented in [13] and [29], we describe our transformations graphically, opposing a simulation business process model element (SPM construct) with its extended timed colored Petri net representation (ETCPN construct). Throughout the mapping, we use the colors and their respective variables defined in table 3.2. The definitions of *random values*, *integers*, and *strings* are trivial. Process-IDs are kept in a

Color	Variable(s)	Type	Literals	Operation(s)
random value	r	$\mathbb{R}$	$0 \leq r \leq 1$	$+, -$
integer	n	$\mathbb{N}$	$0, 1, 2, \ldots$	$+, -$
string	s, x, na, ro	String	“123”, “abc”	
process-ID	pid	Stack	`[]`	`push(x|[]) →[x|]`
			`[h|]`	`push(x|[h|t]) → [x|[h|t]]`
			`[h|t]`	`pop([x|[h|t]]) → [h|t]`
correlation	c	Set	$\{\}$	$\{\} + s \rightarrow \{s\}$
			$\{s_1, s_2, \ldots\}$	$\{s_1, s_2\} - s_1 \rightarrow \{s_2\}$
working time	w	$\in \mathbb{N} \times \mathbb{N}$	$(0, 1200)$	

Table 3.2.: Colors and variables used within the ETCPN transformation

stack, since by hierarchical process relations a token might carry multiple process-IDs. The stack consists of a *head element*, representing the most recent process ID, and a *tail*,

which is again a, probably empty, stack of IDs. Similar to process-IDs, correlation keys, that are required for simulating message exchanges, are kept in a simple set. Working times are pairs of integers referring to the virtual time. As the only frequency considered in this thesis is a week, we can omit the modulo-value.

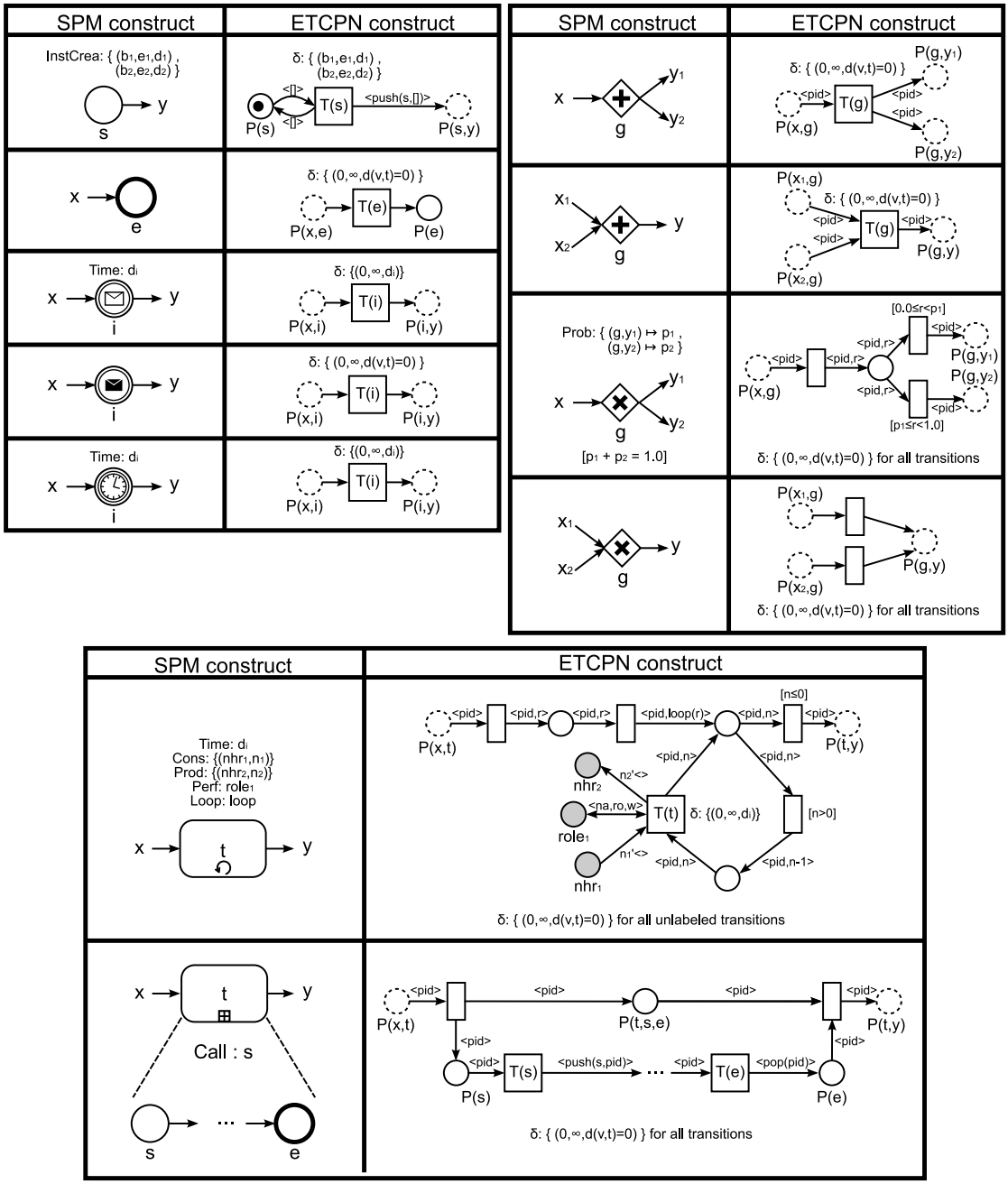
Within the transformations depicted below, the function P() and T() create unique names for places and transitions. Both functions can have arbitrary many parameters.

Figure 3.5 depicts the ETCPN transformations of basic control flow elements. Except for the *XOR-Split* the mapping of gateways is straightforward. Since a gateways does not consume time, the firing delay is 0 throughout the whole simulation. For the *XOR-Split* the configured probabilities need be considered. Therefore, a random value is created that serves as a variable within the guard conditions. Of course, the assigned random value is equally distributed between 0 and 1.

A transition representing a *Start Event* puts a new process-ID onto the stack of process-IDs. Analogously, a transition representing an *End Event* removes the topmost element from the stack. For the start event the instance creation configuration is assigned to the respective transition. The mapping of basic intermediate events is again straightforward. Be aware, that sending events, i.e. EDir = `OUT`, do not consume time and fire immediately. On the other hand, receiving events, i.e. EDir = `IN`, can have delays. Please remember that *Timer Events* allow only constant delays.

A *Task* is represented by a single transition, that is connected to the respective resource places - highlighted by a gray background -, as configured by functions Cons, Prod, and Perf. For non-human resources $n \in \mathbb{N}$ tokens can be consumed or produced for a single transition firing, which is denoted by $n' <>$. If the task is a loop-task, the number of iterations is determined by the given distribution function. As seen for the gateway probabilities, the function is called using a generated random value. The determined number of iterations serves as a parameter within the guard transitions in order to determine the completion of the loop. As we see from the mapping, the resources are not allocated en bloc for all iterations, but each iteration is again competing for the shared resources. This might be subject to discussion and future investigations.

A *Call Activity* for which the Call-function is defined is mapped onto transitions that initiate the call and the return from the called process. The process-IDs are used in order to determine which instance is resumed if a sub-process invocation completes. Otherwise, if the respective *Call Activity* is not part of the domain of function Call, it is handled like an ordinary *Task*.



Notes: x is an input object; y is an output object.
Bindings are specified only if the respective step affects the color values.
Concrete bindings may vary according to other mapping steps.

Figure 3.5.: ETCPN transformations of basic control flow elements

Please note: By the way dependencies on non-human resources are represented, it is redundant if the specific resource is a *tool* or *material*, as "re-using a tool" is equal to "consuming and re-producing a material". Further, a tool might not get returned after each task, but is kept for a sequence of tasks.

In addition to these basic transformations, figure 3.6 shows mappings of more elaborate concepts. As already mentioned before, for simulating message exchanges between different process instances a correlation key is created, when the first message is sent. The

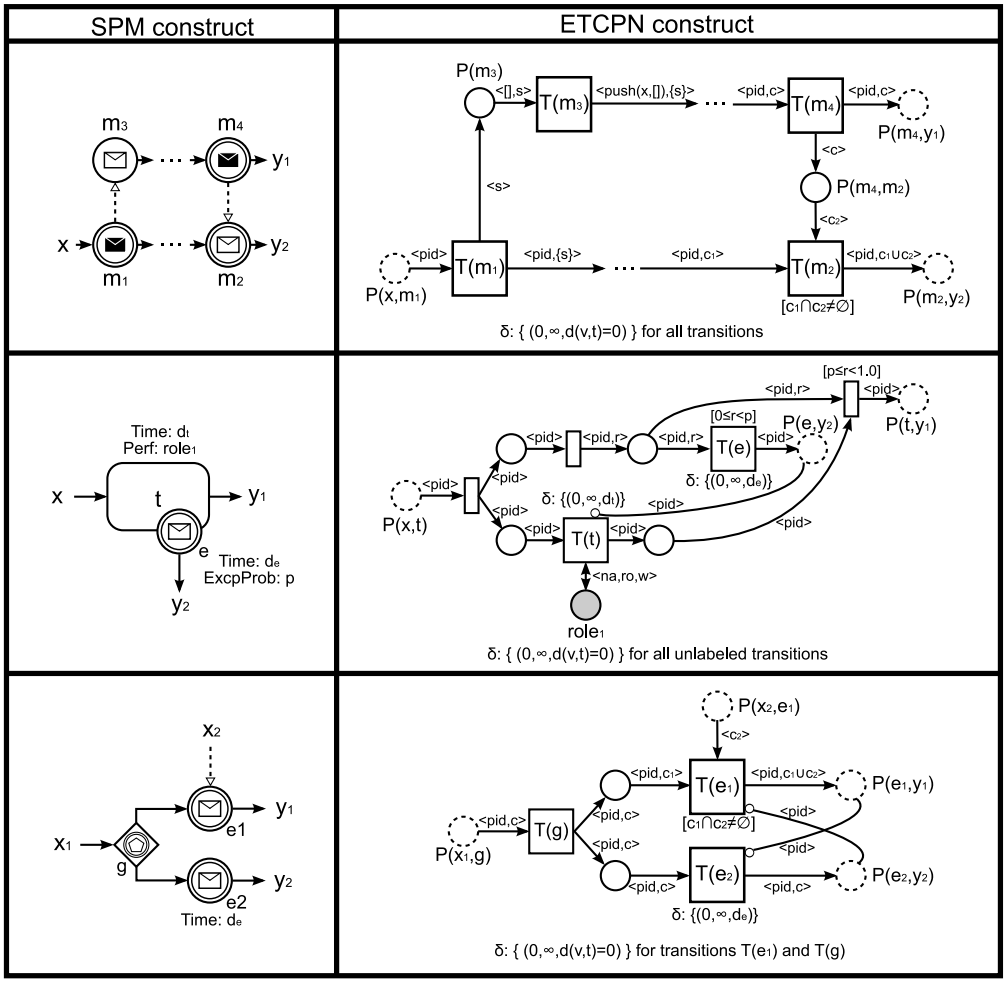


Figure 3.6.: ETCPN transformation of message flows, attached events, and event-based decisions

correlation key is kept until the instances terminate and is used for all messages that are sent. As the transformation shows, if a sending message event is the source of a *Message Flow* arc, it must not have an execution time configured.

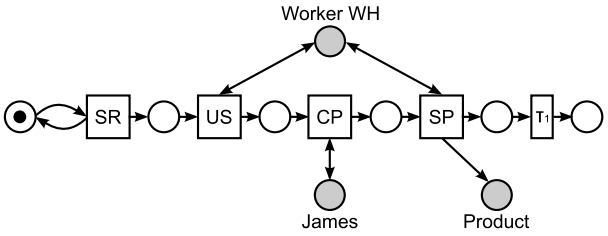
An *Intermediate Event* that is attached to the boundary of a task, requires the usage of *escape arcs*. Initially, the task and a branch that may lead to the interrupting event are concurrently enabled. The occurrence of the event is simulated by a random value that serves as a parameter for the guard conditions. As explained for the ETCPN example in figure 2.3b in section 2.4, a token at Place $P(e, y_2)$ will interrupt the firing of transition $T(t)$. Thereby, the process continues with y_2; otherwise, it continues with y_1 after $T(t)$ has completed firing. In case such an interrupt occurs while $T(t)$ has already started firing, the tokens consumed by $T(t)$ must be put back to their original places. This is particularly import for tokens representing human or non-human resources. For non-human resources it might be subject to discussion if they are always put back, or, in case of consumed materials, might be actually lost. However, for the scope of this thesis, we assume the simple case described before.

Similar mechanisms apply for the transformation of an *Event-based Gateway*. The succeeding *Intermediate Event* nodes are mapped to transitions as explained before. In order to guarantee that only one event occurs for a respective process instance, the output places of the transitions serve as *escape places* for the other ones. Thereby, only one transition can complete firing.

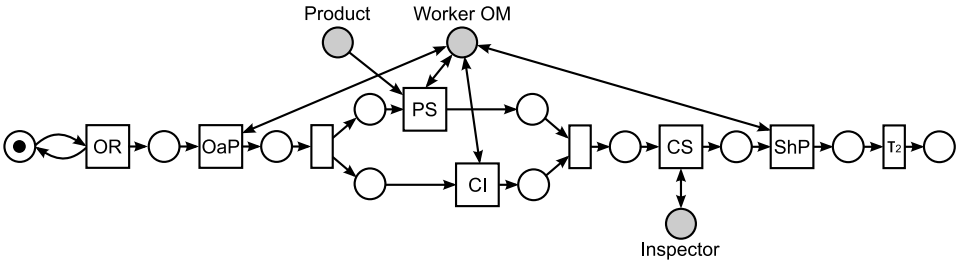
Applying the described transformations to our example scenario in figure 3.4, results in the two Petri nets depicted in figure 3.7. For the sake of simplicity, we only show the structural results and omit the bindings. The transition names have been abbreviated. Examining the two nets, we easily observe that both nets refer to a place *Product*. Such *intersections* of the nets mark places where the nets can be merged, i.e. "glued together". The merge result for our example is depicted in the next section.

3.3.3. Mapping Human and Non-Human Resources

As shown in the ETCPN transformations, each resource that is part of the co-domain of the functions Cons, Prod, or Perf, is mapped to a place labeled with the resource's name. This is satisfactory for non-human resources, where we add the respective number of tokens according to the initial inventory. With respect to our running example, we add 20 equal tokens to the place *Product*. For human resources, however, their hierarchical structures may require changes to the structure of the net.



(a) ETCPN structure for process *Receive Shipment* (figure 3.4a)



(b) ETCPN structure for process *Receive Order* (figure 3.4b)

Figure 3.7.: Mapping Results for the processes in figure 3.4

We seek to *maximally partition* the set of human resources into disjoint sets, such that each of these sets is represented by a single resource place within the Petri net. The finest-grained case would be to have each single human being represented by a single place. The least partitioned case is to have all human resources within a single place. Obviously, we seek to partition the set in order to reduce the size of structural conflicts within the Petri net.

The partitioning of human resources is influenced by two aspects. On the one hand, there are the configuration functions mentioned above. On the other hand, the grouping of resources according to the organizational structure is a factor. We will explain their interplay using our running example, based on algorithm 2.

From the process mapping we get four resource places having the labels *Worker WH*, *Worker OM*, *Inspector*, and *James*, forming the set hrPlaces. Parsing the organizational chart we receive the set persons $= \{\text{Alice}, \text{Bob}, \text{Charles}, \text{Eric}, \text{James}, \text{Steve}\}$. Initially, each of these persons is mapped onto a colored token holding the person's name, its roles, and its working times. The generated tokens are added to the places they belong to. Please note that a token might be added to multiple places and that a token representing a human resource is distinct from tokens representing other, probably also human, resources. For our example, we receive the distribution of tokens depicted in

table 3.3.

Algorithm 2: Mapping of Human Resources

Input: persons is a set of persons; hrPlaces is a set of places for human resources

```
hrTokens ⟵ {} ;
for p ∈ persons do
    t ⟵ tokenFrom (p) ;
    addToken (t, placesForPerson (p)) ;
end
for p1 ∈ hrPlaces do
    for p2 ∈ hrPlaces do
        if p1 ≠ p2 ∧ tokensAt (p1) ∩ tokensAt (p2) ≠ ∅ then
            p1 ⟵ merge (p1, p2) ;
            hrPlaces −{ p2 } ;
        end
    end
end
```

In the second step of algorithm 2, places whose token sets are intersecting are merged together, i.e. their token sets are united and the connections and bindings to and from transitions are adapted. As we can see from table 3.3, places *James* and *Inspector* have intersecting token sets. Therefore, the places are merged. With respect to the merge of place *Product* at the end of the previous section, we receive the final net structure depicted in figure 3.8. Please note, that the transitions representing the *Parallel Gateways* of the original model, have been collapsed into their preceding (OaP) and succeeding (CS) transitions.

Place	Tokens (persons)
Worker WH	{ Bob, Alice }
Worker OM	{ Charles, Eric }
Inspector	{ James, Steve }
James	{ James }

Table 3.3.: Token distribution after the first step of algorithm 2

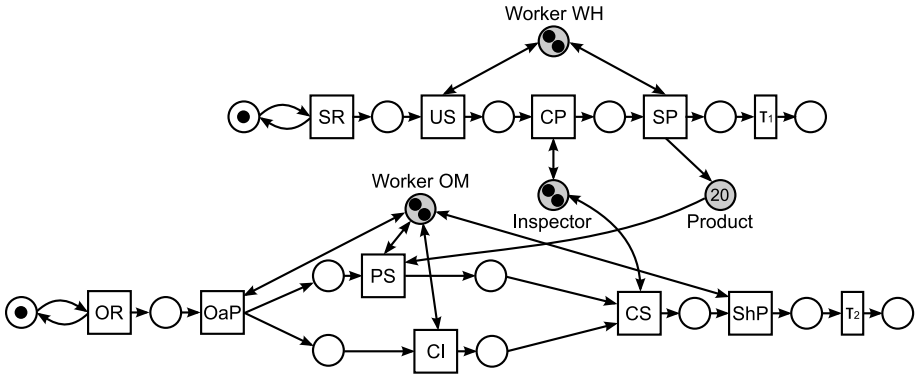


Figure 3.8.: Final net structure for the example in figure 3.4

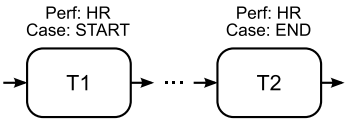
3.3.4. Remarks and Additional Mappings

Definition 15 of a *Simulation Business Process Model* refers to the concept of *case handling* introduced by van der Aalst et al. in [59]. By assigning a *case* to a human resource, the resource is designated to perform all tasks belonging to this specific case.

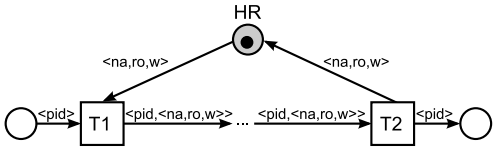
Within this work, we assume a very simple case handling mechanism. If a case is configured, a human resource will subsequently perform all activities for this case without having the opportunity to participate in other process instances in the meantime. This may, at first glance, partly contradict the assumption that human resources participate in multiple processes at a time. However, it is a rather realistic approach with respect to, e.g., manufacturing processes where consecutive steps are usually performed without interruption. Similar examples can be found in versatile domains. Nevertheless, we consider this approach as subject to discussion and future investigations.

From the perspective of extended timed colored Petri nets, our interpretation of the *case handling paradigm*, leads to tokens carrying other tokens. As an example consider the process in figure 3.9a. Task *T1* is designated to start a case. Thereby, the token representing the human resource in figure 3.9b is not immediately returned after the firing of transition *T1*, but instead bound to the token holding the process ID. After firing transition *T2*, for which the case is designated to end, the human resource token is returned to its original place. If a task between *T1* and *T2* is interrupted by an attached intermediate event the bound token is not returned to its original place, but transferred to the path following the event. If the resource should be returned, the first task following the event, must be set to end the case.

As mentioned in the overview of the mapping algorithm in section 3.3, the generated net



(a) The initial process fragment



(b) The resulting ETCPN fragment

Figure 3.9.: Case handling example

is facilitated in the end. This facilitation targets the removal of redundant τ-transitions in order to reduce the size of the net, as we have observed for the *parallel split* and *join* for our running example.

We assume the mapping to be *process-aware*, denoting that within the resulting Petri net the information which transition belongs to which original process is still available. For a transition t and a simulation business process p, let $\mathsf{process\_ref}(t) = pid(p)$ be the *process reference* of t. Regarding the mapping result of our running example in figure 3.8, we observe: For $t \in \{\mathrm{SR}, \mathrm{US}, \mathrm{CP}, \mathrm{SP}, \tau_1\}$, $\mathsf{process\_ref}(t) =$ Receive Shipment; for $t \in \{\mathrm{OR}, \mathrm{OaP}, \mathrm{PS}, \mathrm{CI}, \mathrm{CS}, \mathrm{ShP}, \tau_2\}$, $\mathsf{process\_ref}(t) =$ Receive Order.

4. Simulation Engine and Model Partitioning

As a result of the transformations described in the previous chapter, we receive an extended timed colored Petri net explicitly showing all relations and dependencies that are incorporated in a simulation scenario. This net forms the basis for our simulation. As outlined in the introduction to this thesis, we consider the distribution of our simulation among a set of simulation engines. By using multiple engines for a single simulation run, we expect a gain on the simulation performance. In order to achieve this gain, we seek to exploit the parallelism inherent to the processes of our scenarios. Therefore, we partition the Petri net into a set of spatial regions (cf. requirement TR1). Each of the simulation engines is assigned to exactly one of these partitions.

Before we explain and investigate *model parallelism* in section 4.2, section 4.1 describes a simulation engine that is able to participate in such a distributed setting. Based on the partitioning guidelines outlined in section 4.2, we propose two approaches for partitioning the net in sections 4.3 and 4.4.

4.1. A Logical Process for Conservative Distributed DES

Section 2.5 has already outlined the vital parts of distributed discrete event simulation (DDES). As extended timed colored Petri nets are our formalism of choice, we need to develop a logical process (LP) whose engine knows their execution semantics (cf. requirement FR3). Further, a respective LP must contribute to the technical requirements TR2 and TR3, i.e., facilities for communication and synchronization between multiple engines must be provided.

As stated by literature (e.g. [9]), an appropriate combination of synchronization and model partitioning is crucial for a speed-up of the simulation, i.e., the synchronization protocol affects the notion of a reasonable model partitioning. For the scope of this

thesis, we consider a conservative synchronization protocol for assuring causality among the logical processes. On the one hand, the implementation of this protocol is straightforward. On the other hand, we emphasize our focus on model partitioning, since the investigations on partitioning algorithms might be re-used for other synchronization protocols.

Before we outline the concepts of conservative distributed DES, we define the concept of *subnets* of an extended timed colored Petri net. For a function f with domain D, we use $f|S$ to denote the restriction of f to a subset $S \subseteq D$.

Definition 17 (Subnet) *Let* $ETCPN = (P, T, F, Esc, \mathcal{T}, \lambda, \Sigma, \gamma, \mu, \epsilon, TS, \mathcal{D}, \delta)$ *be an extended timed colored Petri net. An extended timed colored Petri net* $ETCPN' = (P', T', F', Esc', \mathcal{T}', \lambda', \Sigma', \gamma', \mu', \epsilon', TS', \mathcal{D}', \delta')$ *is a subnet of* $ETCPN$*, if, and only if, the following properties are satisfied:*

- $P' \subseteq P$,
- $T' \subseteq T$,
- $F' = \{(n_1, n_2) | (n_1, n_2) \in F \wedge (n_1, n_2) \in P' \times T' \cup T' \times P'\}$,
- $Esc' \subseteq Esc$,
- $\mathcal{T}' \subseteq \mathcal{T}$,
- $\lambda' = \lambda|\mathcal{T}'$,
- $\Sigma' \subseteq \Sigma$,
- $\gamma' = \gamma|P'$,
- $\mu' = \mu|T'$,
- $\epsilon' = \epsilon|F'$,
- $TS' \subseteq TS$,
- $\mathcal{D}' \subseteq \mathcal{D}$, *and*
- $\delta' = \delta|T'$.

An *input border node* $n \in P' \cup T'$ is a node for which the set $\{(x, n) | (x, n) \in F \wedge (x, n) \notin F'\}$ is not empty. Analogously, an *output border node* $n \in P' \cup T'$ is a node for which the set $\{(n, x) | (n, x) \in F \wedge (n, x) \notin F'\}$ is not empty. Generally speaking, a subnet is a part of a Petri net. We say that a subnet S of a net N with a set of transitions T' is the *induced subnet* w.r.t. T' if S contains all places that are connect to transitions $t \in T'$ in N. For induced subnets, output and input border nodes are always places, if we assume that each transition has at least one input place and one output place. This assumption holds for the mapping proposed in the previous chapter.

For the following explanations we consider the timed Petri net depicted in figure 4.1a. Choosing an arbitrary way of partitioning this net, we might receive the three *spatial*

regions shown in figure 4.1b, where each transition, along with the places it is connected to, is put into a single region. In this example, a partition is the *induced subnet* for the transition it contains. Please note: W.r.t. this example a specific place of the original net is an output border place in multiple partitions, e.g. place p_3; in contrast, a place serves as input border place for at most one partition. For the remainder, we only consider partitionings where these properties hold, for reasons given in section 4.2.

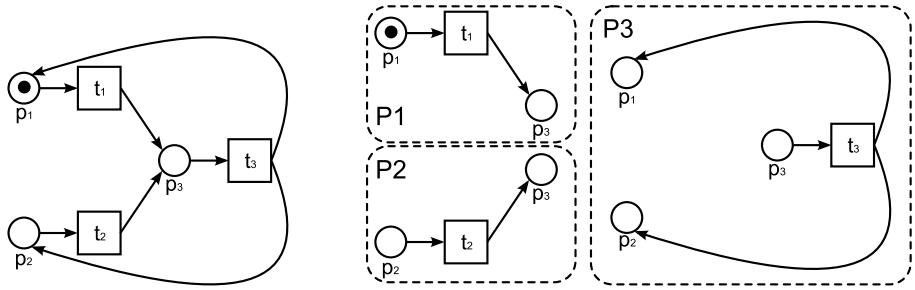


Figure 4.1.: (a) A simple timed Petri net, and (b) an arbitrary partitioning of this net

As outlined in section 2.5, the partitions are simulated by *logical processes*. Each logical process embodies a *Petri net region* (PNR), i.e. one of the partitions, a *simulation engine* (SE), and a *communication interface* (CI).

SE holds the clock, representing the *local virtual time* (LVT) of SE, an *event list* (EVL), which is a temporally ordered list of future events that are scheduled for simulation, and a log which is the history of event occurrences. An *event* $e = \langle et, t \rangle$ consists of a parameterized event type et and a time stamp t which is the local time at which e occurs. With respect to the concepts represented by our chosen class of Petri nets, we distinguish the following event types.

Start Firing ($\mathsf{SF}(tr, tokens)$**)** An event $\langle \mathsf{SF}(tr, tokens), t \rangle$ denotes the start of the firing of a transition tr at LVT t, thereby removing the configured *tokens* from its input places. The occurrence of such event leads to the scheduling of an event $\langle \mathsf{EF}(tr, tokens), t + fd(tr) \rangle$. Remember that $fd(tr)$ denotes the firing delay of tr. Further, as tokens are removed from the transition's input places, it must be checked if other transitions have been disabled and, therefore, their respective SF-events have to be descheduled.

End Firing ($\mathsf{EF}(tr, intokens)$**)** An event $\langle \mathsf{EF}(tr, intokens), t \rangle$ denotes the completion of the firing of a transition tr at LVT t, thereby producing the respective tokens at

the output places of tr. For the computation of the output tokens, the set *intokens* is considered, representing the tokens that have been consumed by the firing of tr. To each produced token the current LVT, i.e. t, is assigned. As new tokens are available after the event is processed, it must be checked if transitions are newly enabled, i.e., if respective SF-events need to be scheduled. Further, as tokens might be produced at *escape places*, we need to check for potential deschedulings of SF- and EF-events. If an EF-event is devalidated, the respective transitions firing is aborted immediately, and the respective resource tokens consumed by this firing are put back to their original places.

Work Begin (WB(hrt)) An event $\langle \mathsf{WB}(hrt), t \rangle$ denotes the begin of the working time of the human resource represented by the token hrt, thereby making the token available for enabling transitions. By the firing of an WB-event the next WB-event for hrt is scheduled. Further, it is checked if token hrt newly enables transitions. If so, the respective SF-events are scheduled.

As the event types show, we decided for a two-step transition firing. The events require checks for newly enabled and disabled transitions. It is, however, not necessary to evaluate all transitions of the respective PNR. Instead, a transition tr can only disable transitions that are in *structural conflict* to tr, or by producing a token at an escape place. On the other hand, tr can only enable *succeeding* transitions, i.e. transitions for which the output places of tr serve as input places. For a deeper understanding of the enabling test, we refer the interested reader to [14].

The events in EVL are ordered according to a two-level priority scheme. First, they are naturally ordered by their occurrence times. Second, if two SF-events have the same occurrence time, we consider the time stamps attached to the tokens. In this case, the event carrying tokens with earlier time stamps is preferred. This is a natural first-come, first-served prioritization of process instances. However, this prioritization scheme has different implications. First, all processes are equally important, i.e., their is no mechanism for considering *external*, i.e. user-defined, priorities. Second, human resources are not choosing tasks from a list of possible tasks, but are assigned to the first occurring task. Third, there is no concept for interrupting a transition firing due to the end of the human resource's working time, i.e., human resources are assumed to work overtime.

The communication interface is responsible for receiving messages from and sending them to other logical processes. Therefore, CI holds an *input queue* (IQ) for each LP from which messages will be received during simulation. Obviously, the number of input

queues is determined by the partitioning of the Petri net. Analogously, an *output buffer* (OB) is assigned to each output border place of PNR. The queues and buffers serve as interfaces to the communication channels connecting the logical processes.

There are two types of messages that are sent between the LPs. A *token-message* $m = \langle tok, p, t \rangle$ carries a token tok to the destination place p; t is the message's time stamp representing the LVT of the sending LP. A *null-message* $m = \langle 0, p, t \rangle$ sent from LP_1 to LP_2 is a promise that LP_1 will sent no token-messages for the destination place p until virtual time t. The need for token-messages is obvious. Null-messages provide a mechanism for *deadlock avoidance* and performance improvement within the conservative synchronization protocol, as will be explained later. Within the input queues, the messages are sorted according to their time stamp.

Figure 4.2 depicts a conceptual view on the logical process handling partition P3 in figure 4.1b. We add some example events and messages for a better illustration of IQs, OB, EVL, and the log.

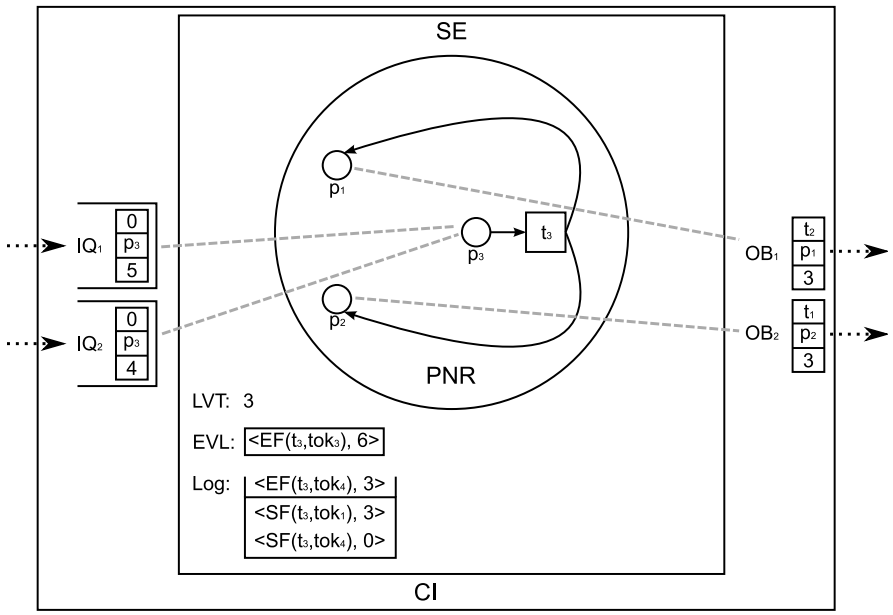


Figure 4.2.: Conceptual view on the LP for partition P3 in figure 4.1b

Based on the LP's architecture, algorithm 3 describes the procedure for the embodied simulation engine. It is a refinement of the basic DES algorithm in section 2.5. In the beginning (step 1), the initial events are scheduled. These events are SF-events for the transitions that are enabled in the initial marking of the handled Petri net region and

Algorithm 3: Conservative DDES algorithm

```
LVT ⟵ 0, EVL ⟵ {} ;
1  scheduleInitialEvents (EVL) ;
   while LVT ≤ endTime do
2     while ¬ isEmpty (EVL) ∧ time (head (EVL)) = LVT do
         fire (dequeue (EVL)) ;
      end
3     updateInputQueues (false) ;
4     IQ_min ⟵ queueWithMinimumTimestamp (), minTime ⟵time (IQ_min);
      evlTime ⟵ time (head (EVL));
5     if (minTime ≤ evlTime ∨ isEmpty (EVL)) ∧ tokenMessage (IQ_min) then
         m_token ⟵ dequeue (IQ_min) ;
         advance (LVT, time (m_token)) ;
         add (place (m_token), token (m_token)) ;
         scheduleNewEvents (EVL) ;
         descheduleDevalidatedEvents (EVL) ;
      end
6     else if minTime > evlTime ∧¬ isEmpty (EVL) then
         fire (dequeue (EVL)) ;
      else
7        sendLookahead () ;
8        updateInputQueues (true) ;
      end
   end
9  sendLookahead;
10 analyzeLog;
```

WB-events for all human resource tokens. After the initialization, the simulation loop is started. First (step 2), all events that are scheduled for occurring at the current LVT are executed. Afterwards (step 3), the input queues are updated in a non-blocking manner - indicated by the parameter `false` -, i.e. all messages that are available at the input channels are inserted into the respective input queue. In step 4, IQ_{min} represents the input queue holding the message with the minimum time stamp w.r.t. all IQs. If this message is a token message and its time stamp is smaller or equal to the time stamp of the first event in EVL, the message is processed (step 5). As outlined before, the reception of a token may lead to newly enabled transitions or, if the receiving place is an

escape place, to the inhibition of already enabled or event firing transitions. Therefore, the respective events need to be scheduled or descheduled. Otherwise, if the occurrence time of the event is smaller than the time stamp of the message, the event is fired along with its potential scheduling and descheduling of other events (step 6). In case the event puts a token to an output border place, a token message is put into the respective output buffer and sent out. If neither a token message is available to be processed nor an event can be fired, i.e., EVL is empty or it is still possible to receive messages with a time stamp smaller than the occurrence time of the first event in EVL, the simulation engine initiates a *blocked waiting* (step 8). This waiting is terminated by the reception of the next message on any input channel. Before the engine falls into the blocked waiting, it sends out *lookahead*-information, i.e. null-messages, for all output border places (step 7).

Sending the *lookahead*, $la(p)$, for a place p, is sending a null-message with a time stamp that is ahead of the LP's current LVT. Thereby, the recipient of the message is informed that no messages for place p will be sent by the respective LP that have a time stamp smaller than $la(p)$. The lookahead depends on the local virtual time, the marking of the Petri net region, and the currently scheduled events.

Consider the snapshot of the logical process in figure 4.2. Disregarding the messages in the output buffers, the LP has to fall into a blocked waiting, since the next event in EVL has a time stamp of 6 whereas there are only two null-messages having time stamps 4 and 5 in the input queues. In this situation, it is possible that token-messages with time stamps smaller than 6 will be received. Therefore, the lookahead for places p_1 and p_2 is computed. Due to the net structure, we find $la(p_1) = la(p_2)$. We assume transition t_3 to have a constant firing delay of 3 time units. The earliest possible time instant for which t_3 gets enabled is 4, i.e., if a token message carrying this time stamp will be inserted into IQ_2. A consecutive firing of t_3 would produce tokens at p_1 and p_2 at a local virtual time of 7. However, there is an EF-event for t_3 scheduled to occur at LVT 6, which is naturally before 7. Therefore, $la(p_1) = la(p_2) = 6$.

After LVT has reached the end time of the simulation, the lookaheads are sent again (step 9), in order to notify the other LPs that there will be no more messages that need to be considered for the current simulation run. Finally, the log is analyzed for creating meaningful results for the user (step 10).

As already mentioned, null-messages, and thereby the lookahead-computation, provide a tool for avoiding deadlocks between multiple LPs. Such deadlocks occur, if there are cyclic waiting dependencies between the logical processes. Using null-messages the LPs can incrementally improve their lookahead and thereby become finally unblocked.

However, to enable such incremental improvement the partitioning algorithm must put a special focus on τ-transitions, if cyclic dependencies are present. We address this problem in the next sections. For detailed information on avoiding deadlocks using null-messages, see, e.g., [34, 16].

4.2. Model Parallelism and Partitioning Rules

A reasonable model partitioning is always a trade-off between multiple issues. On the one hand, its main goal is the exploitation of the inherent model parallelism. On the other hand, we find the ratio between communication and local computation for the logical processes to be a key factor. Sending messages is relatively expensive w.r.t. to the consumed time, compared to local computation. A third issue are the multiple architectures and settings on which the simulation is executed. Obviously, when the simulation is physically distributed on multiple machines the reduction of communication overhead becomes even more important. Summarizing this trade-off, there exists no general-purpose solution to the problem of model partitioning.

However, for the partitioning of timed Petri nets, guidelines have been proposed, describing steps towards a reasonable partitioning, i.e., a partitioning that achieves a better performance than all, or at least most, of the other partitionings and the single-engine execution.

From the perspective of timed Petri nets, the inherent model parallelism is given by the structure of the net (*structural parallelism*), on the one side, and its marking and time constraints (*semantic parallelism*), on the other side. Generally speaking we can state that, two transitions t_1 and t_2 of a Petri net are *inherently parallel*, if there exists a reachable marking, such that both are enabled within this marking, i.e. t_1 and t_2 are not mutually exclusive.

Consider the example net in figure 4.3. From the structure of the net, we derive that transitions t_3 and t_4 will be concurrently enabled after the firing of transition τ_1. This is the only structural parallelism in figure 4.3. Approaches for finding structural parallelism already at the base of the process models are, e.g., the *refined process structure tree* ([61]), or behavioral profiles ([63]). We further identify semantic parallelism. After a firing of t_1, transition t_2 is newly enabled, while t_1 is enabled again. This is, t_1 and t_2 are semantically parallel. As we see for this small example, fully exploiting the model parallelism will usually lead to many small partitions for which a high communication

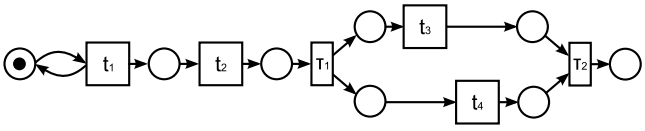


Figure 4.3.: A Petri net with structural and semantic parallelism

overhead is predictable during simulation.

In the literature discussing distributed discrete event simulation of Petri nets, we find versatile approaches to the partitioning problem. In [51], Thomas and Zahorjan create a partition for each place and each transition, targeting the maximum exploitation of model parallelism. However, the amount of messages that will be sent in order to simulate the original net is tremendous and will thereby prevent a performance improvement. In [2], Ammar and Deng allow partitioning by arbitrary *arc cutting*. Although this allows for forming bigger partitions, thoughtless arc cutting might create high message overhead, e.g. for conflict resolution. In [37], the partitioning of Nicol and Roy claims that structurally conflicting transitions must go to a single partition along with all their input places. Thereby, the need for distributed conflict resolution is eliminated.

Analyzing the versatile approaches, Chiola and Ferscha developed rules for Petri net partitioning in [10]. These rules use the *minimum partitioning* of the Petri net as a starting point. The idea of the minimum partitioning is to have each transition along with all its conflicting transitions in a single partition, thereby making all structural conflicts local to the logical processes. Further, all places that are connected to the respective set of transitions are added to the partition. This approach is analogous to the claim of Nicol an Roy mentioned above. A Petri net can be uniquely cut into a set of minimum partitions ([10]). Obviously, the simplest case of a minimum partition contains only a single transition and its input and output places. Please note: The partitioning in figure 4.1b is minimum. Algorithm 4 along with the procedure `FillPartition` sketches the creation of a minimum partitioning.

The minimum partitioning of our running example from the previous chapter (cf. figures 3.4 and 3.8) is shown in figure 4.4. Of course, the minimum partitioning could be used for the distributed simulation since the partitions are valid inputs to the logical processes described in the last section. Further, the model parallelism is exploited to a high extent. However, the partitions are generally small, leading to high message traffic while spending only a vanishingly small amount of time for local computation.

Chiola and Ferscha propose the following rules for merging partitions into bigger ones:

Algorithm 4: Minimum Partitioning of a Petri net

Input : $N \longleftarrow (P, T, F)$ is a Petri net

Output: A set of Petri nets, which represent the partitions of N

handled $\longleftarrow \{\}$;
partitions $\longleftarrow \{\}$;
for $t \in T$ **do**
  if $t \notin$ handled **then**
    $P_i \longleftarrow \{\}, T_i \longleftarrow \{t\}, F_i \longleftarrow \{\}$;
    $N_i \longleftarrow (P_i, T_i, F_i)$; `// create a new partition`
    `FillPartition` (N_i, t, N) ;
    handled $\cup T_i$;
    partitions $\cup N_i$;
  end
end
return partitions

Procedure `FillPartition`(N_i, t, N)

Input : $N_i \longleftarrow (P_i, T_i, F_i)$ and $N \longleftarrow (P, T, F)$ are Petri nets, with $P_i \subseteq P, T_i \subseteq T$, and $F_i \subseteq F$

Input : t is a transition with $t \in T_i$

$P_i \cup {}^{\bullet[N]}t \cup t^{\bullet[N]}$;
`// Collect transitions that are in structural conflict with` t
$T_{conf} \longleftarrow \left(\bigcup_{p \in {}^{\bullet[N]}t} p^{\bullet[N]}\right) - T_i$;
for $t_{conf} \in T_{conf}$ **do**
  $T_i \cup t_{conf}$;
  `FillPartition` (N_i, t_{conf}, N) ;
end

Rule 1 Mutually exclusive transitions go into one LP, since they bear no potential parallelism.

Rule 2 Partitions leading to high message traffic intensity are merged to save message transfer time.

Rule 3 For transitions having a single input place, the input place can serve as input border place for the partition, since the enabling test is trivial for these transitions.

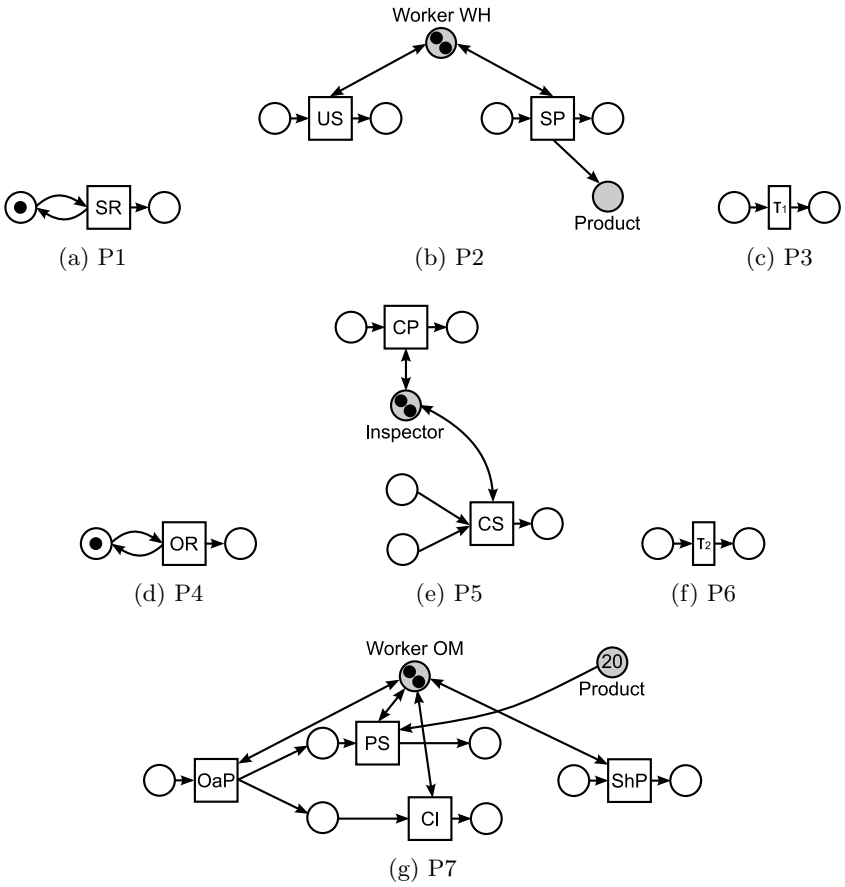


Figure 4.4.: Minimum partitioning of our running example in figure 3.8

Rule 4 Endogenous simulation speed should be balanced, i.e. the probability of blocked waiting is reduced by balanced virtual time increments in all logical processes.

The rules have been slightly adapted to fit the wording and the use case of this thesis. The benefits of rules 2 and 3 are obvious. Rule 4 requires a lot of investigations in order to be automatically applied during partitioning. Rule 1 basically states that we should avoid to put model parallelism into a single partition.

Projecting rule 1 onto the distribution of our business process simulation, we make the following observation. Since we receive a single Petri net from our transformations, all process instances will be running within the same net. Thereby, transitions that are mutual exclusive w.r.t. a single process instance, e.g. US and SP in our running example, get simultaneously enabled if we consider multiple process instances within the same net. Therefore, if we assume a sufficient amount of available resources within the

net, we will hardly find any transitions that are really mutual exclusive.

Before proposing another partitioning rule we define the concept of the *partition graph* for a set of of partitions.

Definition 18 (Partition Graph) *The* partition graph *for a set* Part *of Petri net partitions is a tuple* $PG = (\mathsf{Part}, A)$*, where* $A \subseteq \mathsf{Part} \times \mathsf{Part}$ *is a multi-set of directed arcs. For two partitions* $P_1, P_2 \in \mathsf{Part}$ *and each place* p_b*, such that* p_b *serves as an output border place in* P_1 *and as an input border place in* P_2*,* $(P_1, P_2) \in A$*.*

The *arc-degree*, $AD(p)$, for a partition p is the number of arcs connected to p in PG. In addition, we define the *partition-degree*, $PD(p)$, as the number of partitions p is connected to in PG. Figure 4.5 illustrates the partition graph for the minimum partitioning of figure 4.4. For example, $AD(P5) = 5$ with $PD(P5) = 2$, and $AD(P1) = PD(P1) = 1$. As we can see for $P7$ and $P5$, there can be multiple arcs having the same direction between two nodes. We can further restrict AD to incoming (AD_{IN}) and outgoing (AD_{OUT}) arcs. Analogously, we can restrict PD to PD_{IN} and PD_{OUT}. The arc degree and partition degree can be used as a metric for inferring on the communication overhead.

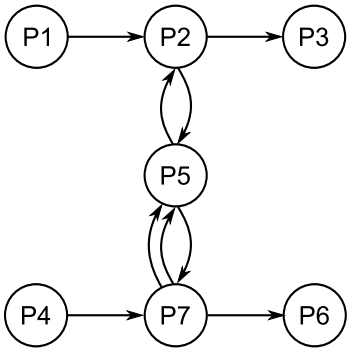


Figure 4.5.: Partition Graph for figure 4.4

In addition, the partition graph visualizes cyclic dependencies between the partitions and, thereby, between the logical processes that would run the simulation. As mentioned before, such dependencies might lead to a deadlock using a conservative synchronization protocol. Therefore, we state the following rule:

Rule 5 For partitions that form a cycle within the partition graph, it has to be ensured that they can constantly improve on their lookahead, i.e. a potential deadlock due to cyclic wait dependencies is avoided.

A necessary condition for such partitions is that for all firing sequences $t_0, t_1, \ldots t_j$, where

t_0 removes a token from an input border place and t_j produces a token at an output border place, at least one of the $t_i, 0 \leq i \leq j$, has a non-zero firing delay.

Within the following two sections, we seek to provide concrete algorithms for constructing reasonable model partitionings. Despite knowing that there is no general-purpose solution, the algorithms are assumed to provide a starting point for the automated model partitioning for distributed business process simulation. The algorithms are not set to return a fixed number of partitions. If this should be necessary, e.g. by computational resource constraints, some of the created partitions might be further merged, where rules 1-5 apply analogously.

4.3. Bottom-Up Partitioning

As outlined above, sending and receiving messages has a key influence on the performance of the distributed simulation. First, it takes much more time than local computation. Second, due to the conservative protocol, a logical process blocks its computation until it has received all token-messages or the respective null-messages, that have to be processed before processing the next local event. One factor on the likeliness of blocked waiting is the number of input queues of the logical process, i.e. $PD_{IN}(p)$ for the partition p controlled by the LP. Obviously, $PD_{IN}(p)$ is affected by $AD_{IN}(p)$.

Therefore, our first partitioning approach tries to keep AN_{IN} and PD_{IN} small. This is, we seek to have only one input border place for most of our partitions, which is similar to Rule 3 in the previous section. We further seek to avoid small regions which would lead to a bad ratio of message sendings and local computation (Rule 2).

Algorithm 6 describes the approach, using the following functions:

filter(*set***,** *block***)** Filters the collection *set* according to the constraints specified by the given *block*. It returns a collection containing all the elements for which the given *block* evaluates to true. *set* is not changed by this function.

merge(p_1**,** p_2**)** Merges partition p_2 into partition p_1, removes p_2 from the set of partitions, and updates the partition graph. The function returns the new p_1.

forceMerge(set_1**,** set_2**)** Merges every member of set_1 to a partition of set_2, where $set_1 \subseteq set_2$. If two elements of set_2 are equally good for merging an element of set_1, a non-deterministic choice is made.

succPartition, precPartition(p, PG) Returns the set of succeeding (preceding) partitions for partition p in the partition graph $PG = (P, A)$. A succeeding (preceding) partition r for p is a partition for which $(p, r) \in A$ ($(r, p) \in A$). If the set has only one element, the single element is assumed to be returned.

transitions(p) Returns the set of transitions in partition p.

Algorithm 6: Bottom-Up Partitioning

Input : PN is an extended timed colored Petri net

Output: partitions is a set of partitions of PN

```
   partitions ⟵ minimumPartitioning (PN) ;
   pg ⟵ partitionGraphFrom (partitions) ;
1  sequences ⟵ filter (partitions, {p → AD_OUT (p, pg) = AD_IN (p, pg) = 1}) ;
2  maximize (sequences) ;
3  foreach s ∈ sequences do
       p ⟵ succPartition (s, pg) ;
       if AD_IN (p) = 1 then
           s = merge (s, p) ;
           sequences −{s} ;
   end
   // remove small cycles
4  foreach p ∈ partitions do
       if PD_IN (p) = PD_OUT (p) = 1 then
           if succPartition (p, pg) = precPartition (p, pg) then
               p = merge (p, succPartition (p)) ;
   end
5  groupInstanceCreation (partitions) ;
   small ⟵ filter (partitions, {p → |transitions (p)| ≤ 2}) ;
6  forceMerge (small,partitions) ;
   shortSequences ⟵ filter (sequences, {s → | transitions (p)| ≤ 5}) ;
7  forceMerge (shortSequences,partitions) ;
```

The algorithm starts with with the minimum partitioning. In a first step, we filter the set of minimum partitions such that only the sequences remain, i.e. partitions having an input arc degree and output arc degree of 1. In step 2, we try to merge these small sequences into longer sequences. For each of the resulting *maximized sequences* s we check its *succeeding partition* in PG (step 3). Let p denote the succeeding partition of s. If $AD_{IN}(p) = 1$, we merge p to s. Thereby, the size of the input border of s remains

constant while its output border may grow. The size of the output border, however, has no direct impact on the performance of the respective logical process.

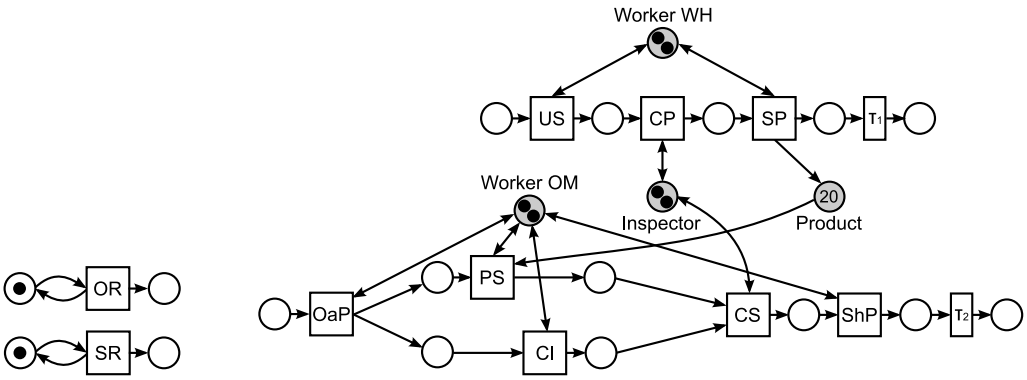
Projecting these three steps onto the minimum partitioning of figure 4.4, we observe no change in the partitions, simply because there are no sequences we could maximize.

Until this point in the algorithm, we assume the created partitions to be rather small, which is obviously supported by our running example. This is a reasonable assumption, since there are, in general, no large sequences, i.e. sequences of more than 15 transitions, that do not have any dependencies to human or non-human resources. Therefore, further steps are required to create partitions of reasonable size, such that an appropriate ratio between local computation and message exchange is achieved.

Continuing the partitioning with step 4, we look for the simplest cyclic dependency we can observe within the partition graph. If there is a partition p having $AD_{IN}(p) \geq 1$, $AD_{OUT}(p) \geq 1$, and $PD(p) = 1$, p is merged into the single partition it is related to, in order to remove the cyclic dependency from PG. Thereby, we partly address Rule 5. Again, this step does not affect our running example, i.e. we still have the minimum partitions.

Step 5 is the only step in the algorithm where we deliberately merge partitions that are actually unrelated. The partitions containing transitions, for which an instance creation distribution is defined are grouped into a single partition. As we have, so far, maximized only sequences, it is assured that the created partition has no input border place. Therefore, the respective logical process will never fall into a blocked waiting, i.e., all process instances are created without waiting. There are, of course, other opportunities for handling these *instance creating partitions*. On the one hand, for maximally exploiting the model parallelism, we could assign each of the partitions to a single logical process. This is reasonable if there are enough computational resources available. On the other hand, integrating the instance creation into partitions that have a non-empty input border bears the potential of slowing down the simulation, because the instance creation might be delayed by blocked waiting times. The consolidation of instance creating partitions is the first step affecting our running example, resulting in the partition in figure 4.6a.

The concluding steps 6 and 7 of algorithm 6 seek to eliminate small partitions. The problem with having small partitions has already been outlined above. In step 6 we force all partitions containing only two or less transitions to be merged into bigger partitions. Of course, we explicitly exclude the instance creating partitions. For our example, this



(a) P1 + P4 (b) P2 + P3 + P5 + P6 + P7

Figure 4.6.: Bottom-Up partitioning result for the example of figure 3.8

leads to the integration of the partitions P2, P3, P5, and P6 into P7 (see figure 4.6b). This step may have an unpredictable impact on the resulting communication interfaces. However, we decide to merge these small partitions in favor of reducing message sendings. For bigger examples, step 7 would further force short sequences, i.e. sequences consisting of only five or less transitions, to be merged into other partitions. Please note: This integration of sequences will not increase the size of the communication interface of the logical process, but may reduce it. These concluding lines add a potential non-determinism to the algorithm that may lead to different partitionings in different runs, since the function `forceMerge` may choose arbitrarily between multiple equally good opportunities for merging a partition. However, as the opportunities are considered to be equally good, the effect of this arbitrary choice can be neglected.

Summarizing this approach, we step-wise construct partitions of reasonable size with a small value for PD_{IN} by adding and combining small partitions. Therefore, we call this approach a *bottom-up partitioning*.

4.4. Top-Down Partitioning

As discussed earlier, the determination of mutual exclusive net parts within the generated net is not trivial. However, with respect to a single process instance, all parts are mutual exclusive except for those that are explicitly branched by a transition. These branches arise if we have modeled concurrency within the original process models.

Based on this idea of mutual exclusiveness w.r.t. a single instance, we propose a second

partitioning algorithm. Its basic approach is to partition the generated net on the basis of the original processes. Thereby, parts belonging to the same process go into the same partition.

Algorithm 7 describes the approach. In addition to the definitions in the previous section, we define the following functions:

procRef(p) Returns the set of process references for partition p. Let T be the set of partitions in p. Then, $\mathsf{procRef}(p) = \bigcup_{t_i \in T} process\_ref(t_i)$.

enrichOrMerge(set_1, set_2) This function is similar to `forceMerge` in the previous section. For each partition $p_1 \in set_1$ the function looks for enriching the partition by moving sequential parts of partitions $p_2, p_3, \ldots \in set_2$ to p_1. If such enrichment is not possible, it forces p_1 to be merged to another partition.

Algorithm 7: Top-Down Partitioning

Input : PN is an extended timed colored Petri net

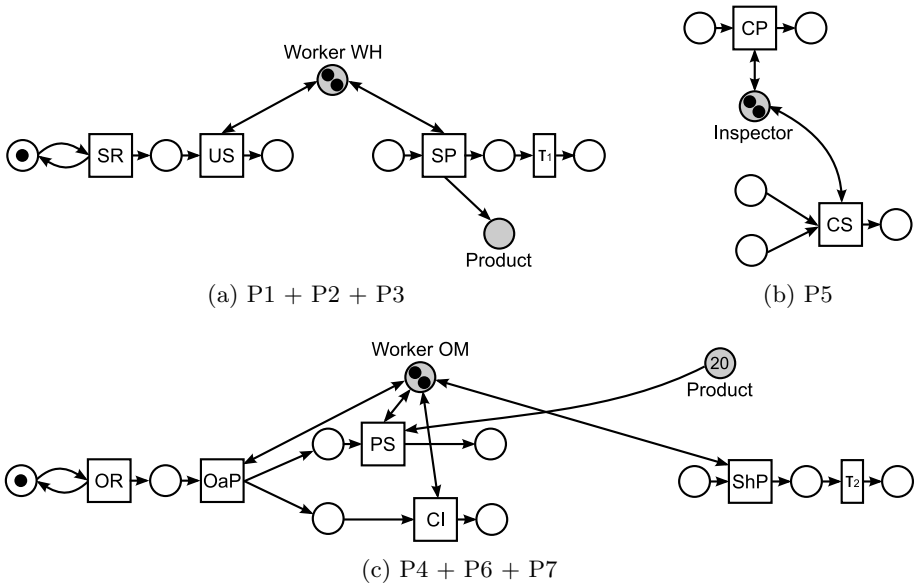
Output: partitions is a set of partitions of PN

```
partitions ⟵ minimumPartitioning (PN) ;
foreach p1 ∈ partitions do
    if |procRef (p1)| = 1 then
1       process ⟵filter (partitions, {p2 → p2 ≠ p1 ∧ procRef(p2) = procRef(p1)}) ;
        foreach p2 ∈ process do merge (p1, p2);
    end
end
pg ⟵ partitionGraphFrom(partitions) ;
2 smallAndCommunicative ⟵ filter(partitions, {p → |transitions(p)| < AD(p)}) ;
3 enrichOrMerge (smallAndCommunicative,partitions) ;
zeroLA ⟵ filter(partitions, {p → hasZeroLookaheadSequence(p)}) ;
4 enrichOrMerge (zeroLA,partitions) ;
```

Once more, we start with the minimum partitioning of the net in order to have conflicting transitions in a single partition. In step 1, partitions that have a single process reference are merged with other partitions having the same process reference. The, thereby, created *process partitions* contain those parts of the processes that are not competing for resources with other processes. Due to this independence, these partitions can be considered to be completely concurrent to each other.

Applying the first step to our running example, we receive the partitions depicted in

figure 4.7. As the figure illustrates, we receive an imbalance on the partition size. While the process partitions are rather big, the other partitions holding the resource conflicts are probably small w.r.t. the number of transitions, e.g. partition $P5$ in figure 4.7b.



(a) P1 + P2 + P3

(b) P5

(c) P4 + P6 + P7

Figure 4.7.: Partitions after step 1 of algorithm 7

Another phenomenon concerning these *resource partitions* is their comparably big communication interface. For example, partition $P5$ has only two transitions but an arc degree of 5. Thinking about the simulation of this partition, we can state that every transition firing will lead to the sending of a message, i.e., we observe a bad ratio between local computation and message sending. In step 2, we filter such partitions from the set of partitions for addressing this problem in step 3, following partitioning rule 2.

As outlined for the function `enrichOrMerge`, we seek to enhance these partitions or, if such enhancement is not possible, fully merge them into another partition. For enriching a partition p, we try to find transition sequences within the process partitions, that could be moved to p. For our example partition $P5$, there are no such sequences. Therefore, we seek to merge $P5$ to one of the remaining two partitions. Regarding the partition graph in figure 4.8, we have three arcs between $P5$ and the partition $P4 + P6 + P7$, and only two arcs between $P5$ and $P1 + P2 + P3$. Since the former opportunity reduces the number of border places, and thereby the size of the communication interfaces, we create a partition $P4 + P5 + P6 + P7$.

In step 4 of algorithm 7, we address partitioning rule 5. We filter partitions, for which

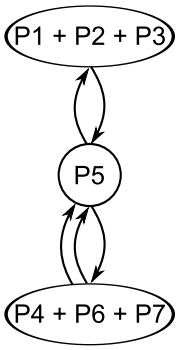
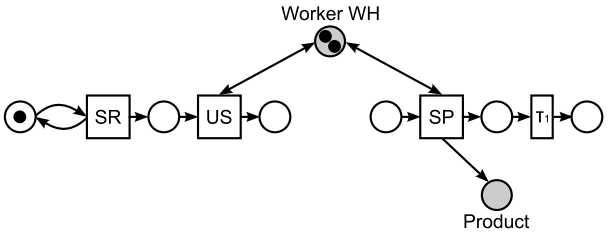


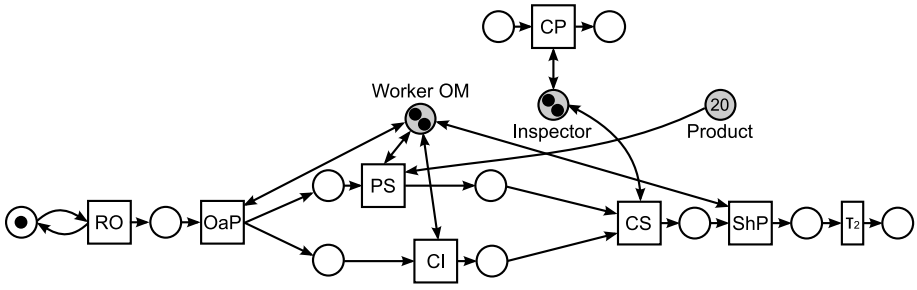
Figure 4.8.: Partition graph after step 1

firing sequences can be identified that do not improve the lookahead. We handle the resulting set of partitions in the same way as the small partitions in step 3. For our running example this step does not affect the partitioning. As a result, we receive the two partitions shown in figure 4.9.

Obviously, the results and the algorithm are completely different to the bottom-up partitioning in the previous section. In this second approach, we more rely on the original processes and try to balance the size of partitions afterwards. We, therefore, call this approach a *top-down partitioning*.



(a) P1 + P2 + P3



(b) P4 + P5 + P6 + P7

Figure 4.9.: Top-down partitioning result for the example of figure 3.8

There is a possible variation of the top-down approach, such that, analogous to the bottom-up partitioning, the instantiating transitions are grouped into a single partition. For our running example, this would create a third partition which is equal to the partition in figure 4.6a.

A second variation targets *dangling* parts. Imagine, transition CP in figure 4.9b would not be connected to the place *Inspector*. As this would be a very small part that is not connected to the rest of the partition, it is moved to another partition, where it can be connected to the net structure. By moving such danglers, the ratio between message sendings and local computation is intended to be improved.

Further, we could improve the implementation of `enrichOrMerge`, such that it does not only consider pure sequences of transitions, but also considers parts that would reduce the communication interface of the partition under consideration. This is, however, subject to future work.

5. Prototypical Realization

Before we evaluate our approaches in the next chapter, this chapter describes the concrete architecture of our prototype. It is based on the conceptual blueprint described in section 3.1.

The simulation engine and the partitioning strategies are integrated into an already existing modeling framework. This Java-based framework is used for prototypical implementations at the inubit AG[1]. Further, the framework provides a simple web-server and web-based modeling and configuration interfaces.

A major task of our prototype is the distribution of the simulation onto multiple computational resources, which might be different threads running on the same machine or even different machines. On the one hand, although concurrent programming in Java[2] is supported, it requires rather high effort. On the other hand, there are languages that have been proven to be well-suited for working with multiple threads of execution, like Erlang[3] ([4]). Fortunately, there is a language that combines the concepts of Erlang with the Java Virtual Machine (JVM). *Scala*[4] ([39]) is a language integrating object-oriented and functional programming, running on the JVM. Scala, following Erlang, provides a light-weight process abstraction, called *actors* ([23]). Actors communicate asynchronously using message-passing. The performance of Scala, also in the context of concurrent programming, is documented by several commercial projects[5]. Scala abstracts from the concrete distribution of threads to computational units and also provides means for physically distributed execution.

The seamless integration with Java and the high-performance framework for concurrent programming make Scala an ideal solution in our setting. We further benefit from the functional programming style, especially, when handling collections of objects. Figure 5.1

[1] `http://www.inubit.com`
[2] `http://java.com`
[3] `http://www.erlang.org`
[4] `http://www.scala-lang.org`
[5] For a list of companies using scala, please see: `http://www.scala-lang.org/node/1658`

shows the separation of concerns between Java and Scala in our prototype. The modeling and simulation configuration aspect is handled via Java; the nets, their partitioning, and the simulation are handled in Scala.

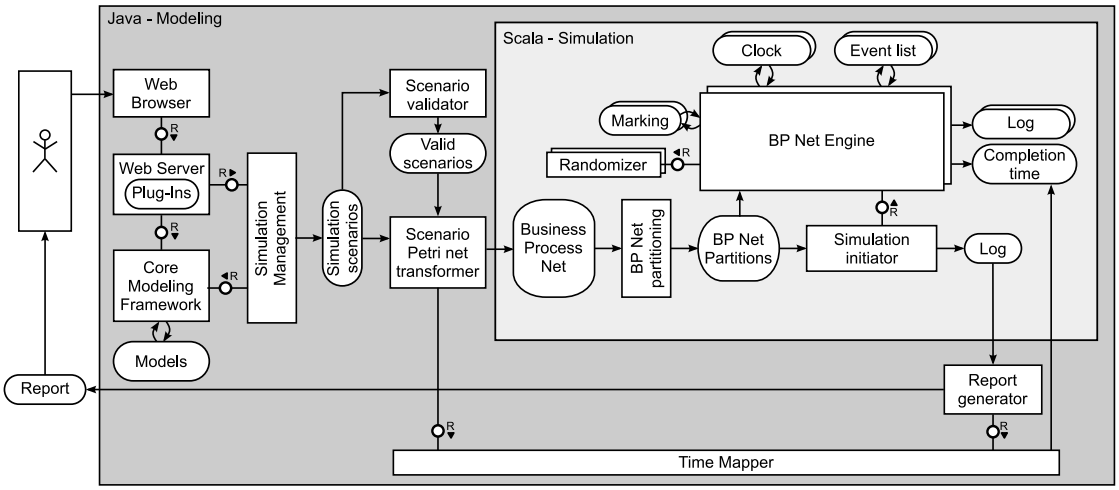


Figure 5.1.: Integration into the existing modeling framework

As we can see from the figure, the prototype does not employ extended timed colored Petri nets, but a concept called *business process net* (BP net) that is similar to ETCPN. As there is no general implementation of ETCPN required, BP nets have been introduced in order to facilitate, e.g., the enabling test of transitions and the mapping of simulation scenarios. However, BP nets are still formally based on extended timed colored Petri nets. Figure 5.2 depicts the classes that form the BP net implementation, as a UML class diagram ([21]).

Following the definition of Petri nets, a BP net consists of places and transitions. The flow relation is represented by the *inputs* and *outputs* fields. An instance of the class *DelayedTransition* holds a function representing its firing delay. If such delay is not necessary, this function constantly returns 0, which is the default configuration. Except for *OutputBorderPlace*, the different implementations of the *Place*-interface represent the different color types of the places. As there is no class for representing arcs, the class *EscapePlace* is introduced as a special *ProcessPlace* to be handled differently during the enabling test. The class *OutputBorderPlace*, is a wrapper that connects an arbitrary place with an output buffer. Similarly, the different actions and guard conditions of transitions are incorporated in the different refinements of the class *DelayedTransition*. The general case is the class *BPTransition* that represents tasks, gateways and most of the events. The class provides means for specifying guards on the random value and

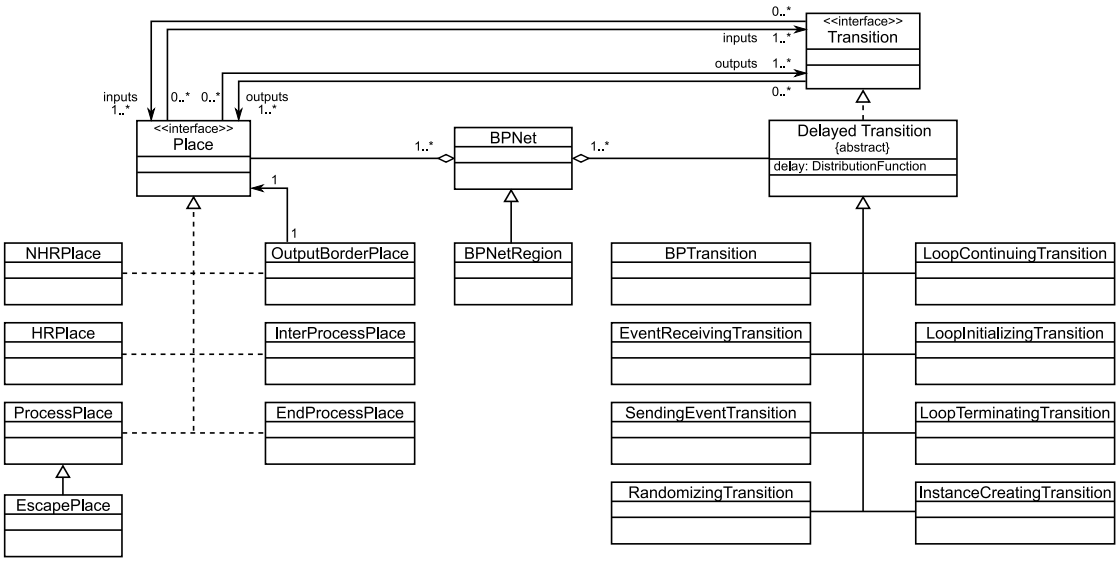


Figure 5.2.: UML class diagram: BP nets

human resources. All other transition classes have telling names that indicate their purpose.

BPNetRegion is a class representing partitions of a *BPNet*. A vital mechanism of colored Petri nets are *arc expressions* and *bindings*. In the prototype they are primarily handled by the different transitions and place implementations. Nevertheless, for non-human resource places a binding can specify the number of tokens that is consumed by a respective transition firing.

The concept of actors provided by Scala perfectly fits the use case of logical processes. Therefore, as we can see in figure 5.3, a logical process is a special actor. Since there are multiple opportunities for synchronization, the class *LogicalProcess* is further sub-classed. As we only implement one synchronization protocol, *CoservativeLP* is the only sub-class in our case. However, the architecture can be easily extended by further protocols. The *SimulationInitiator* creates the respective number of LPs and assigns their *BPNetRegion*. If an LP terminates it reports its termination to the initiator.

By changing the superclass of *LogicalProcess* from *scala.actors.Actor* to *scala.actors.remote.RemoteActor*, we can easily switch to a physically distributed simulation using multiple machines.

For partitioning an instance of class *BPNet*, the four partitioning strategies can be used. The class *SingleRegionPartitioning* refers to the case where the whole net goes into a

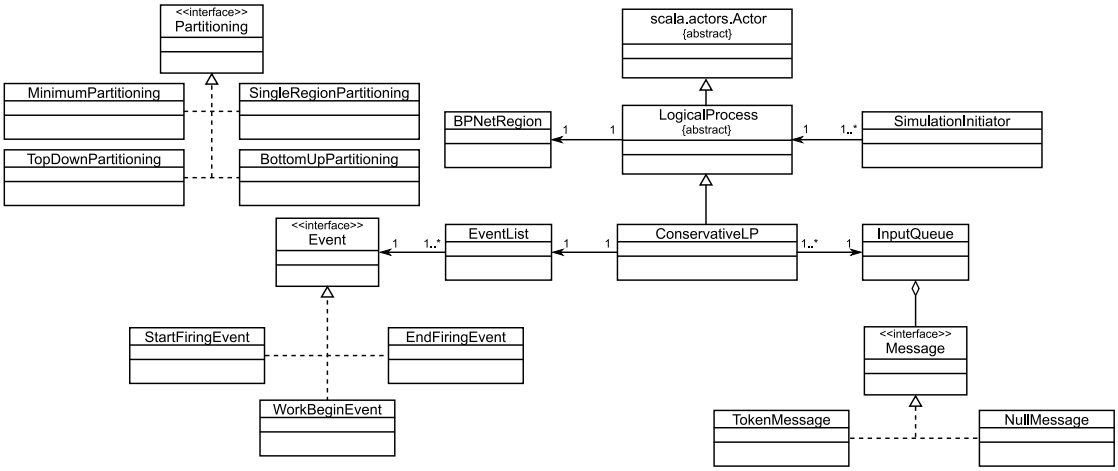


Figure 5.3.: UML class diagram: Logical process and partitioning

single partition, i.e., actually no real partitioning takes place. *MinimumPartitioning*, *TopDownPartitioning*, and *BottomUpPartitioning* provide implementations of the algorithms presented in the previous chapter. By implementing the *Partitioning*-interface, new partitioning can easily be added to our prototype.

From a user perspective, the prototype provides web interfaces for configuring the scenarios and displaying the results. Figure 5.4 shows the scenario configuration screen, where process models, human resource models, and non-human resources can be added to or removed from the scenario.

Figure 5.5 shows the modeling interface, where a task configuration dialog is displayed. We can see the elements for specifying the execution time and non-human resource consumption and production. Further, there is a field for configuring the case participation of that task. The *Pools* and *Lanes* of the process model are linked to elements of a human resource model (cf. requirement MR2).

Figure 5.6 depicts an excerpt of the result visualization (cf. functional requirement FR4). Some tasks are colored to denote that a kind of bottleneck has been detected. In this example, for the framed task almost all instances have endured a delay, i.e., they had to wait before the task execution could start.

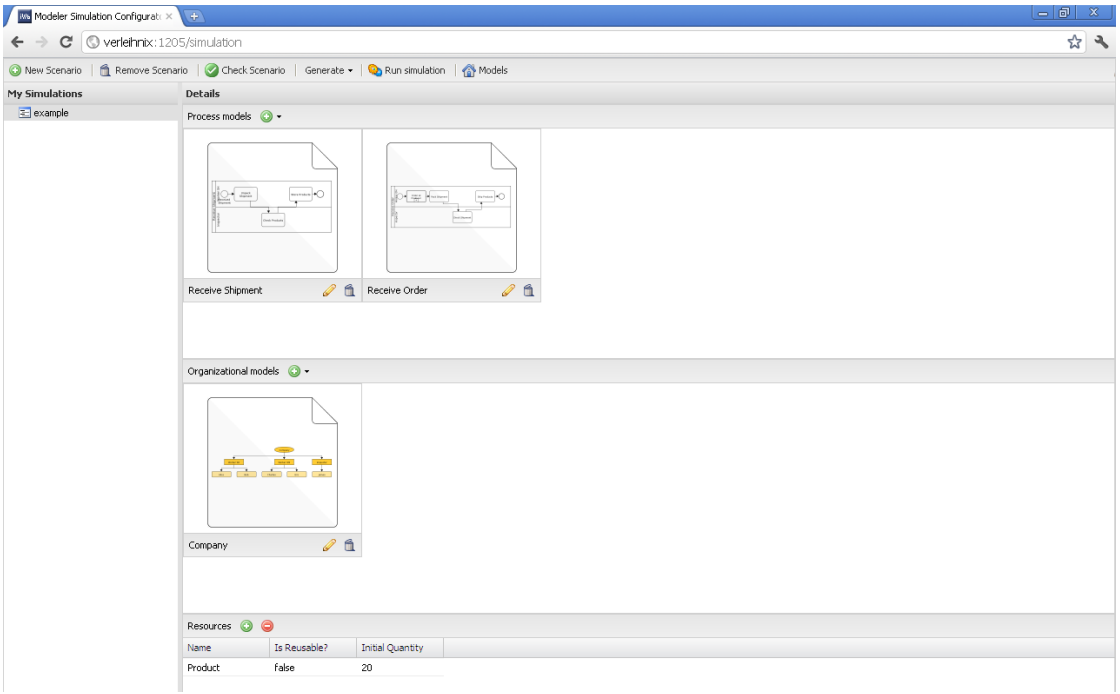


Figure 5.4.: Screenshot: Scenario configuration screen

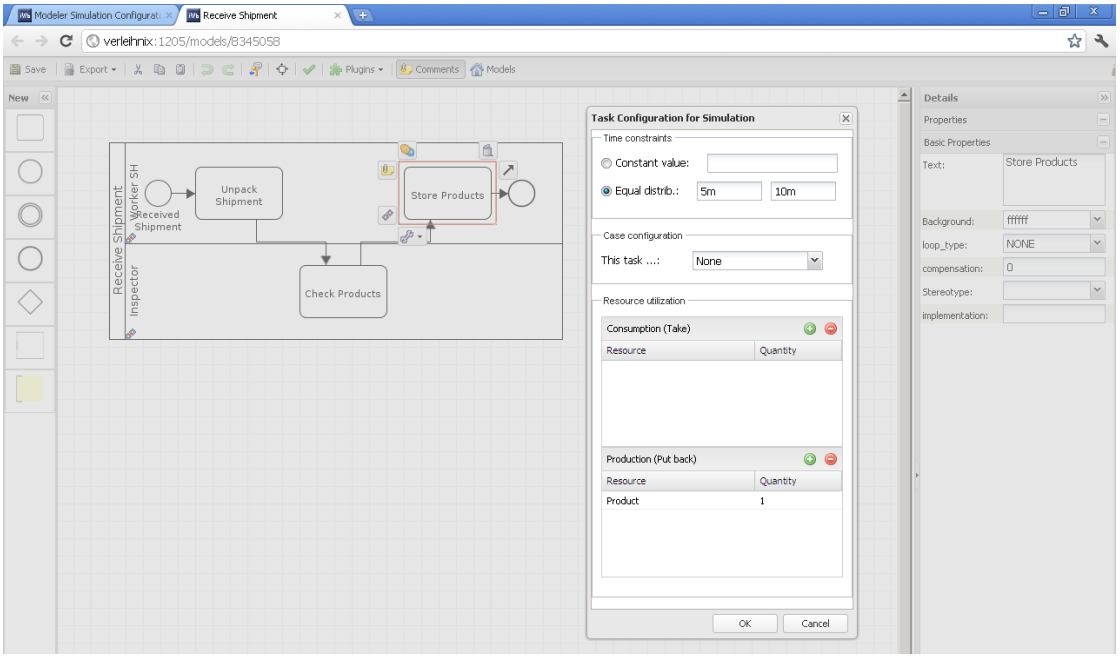


Figure 5.5.: Screenshot: The modeling interface with a task configuration dialog

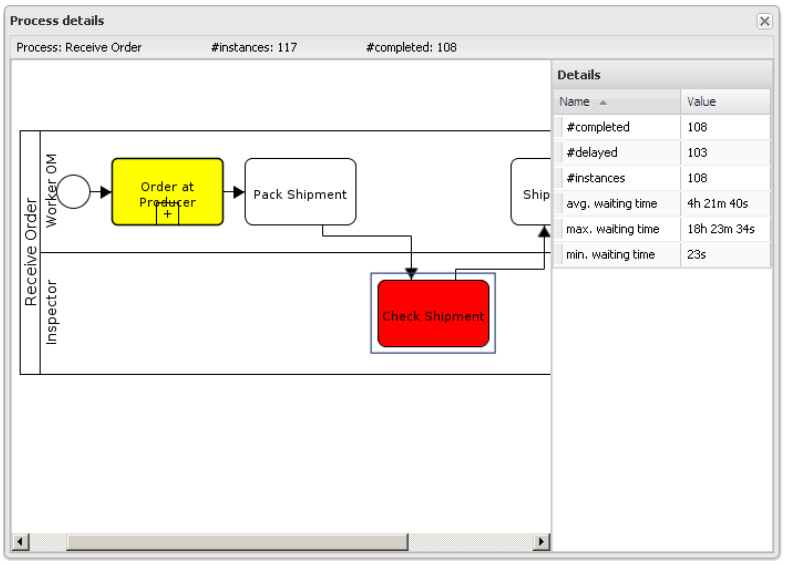


Figure 5.6.: Screenshot: An excerpt of the simulation results

6. Evaluation

This chapter presents experimental results for the previously developed algorithms. Section 6.1 describes two use cases for our simulation. Using these examples and the implemented prototype, statistics on the approaches are presented in section 6.2. Section 6.3 analyzes the gathered results and draws conclusions.

6.1. Example Scenarios

Within this section, we describe two examples we regard as suitable for applying our simulation approach. First, section 6.1.1 presents an example scenario considering the intra-enterprise processes of a software company. Second, section 6.1.2 depicts a simple supply chain example involving multiple enterprises.

6.1.1. Example 1: TurboSoft Inc.

The first example completely follows the motivation of this thesis. It describes a fictive software company called *TurboSoft Inc.* TurboSoft runs 16 processes. For focusing on the evaluation of the example in this chapter, we moved the process models and organizational chart to appendix A. The processes are based on the processes of a real German software company. Thus, the models are depicted in German language.

The *Lanes* within the process models indicate the performing roles, that can be found within the organizational chart. As we can see, the role *PS* is the most used resource within the processes. Further, the processes do not explicitly use any non-human resource.

Regarding the sub-process relation, we can identify four groups of processes. First, the biggest group contains processes *Project realization*, *Business modeling*, *Capture Requirements*, *Design*, *Implementation*, *Deployment*, *Test Software*, and *Project Completion* (figures A.12, A.1, A.13, A.4, A.7, A.2, A.16, and A.3). Second, another group

consists of the processes *Incident Management*, *Problem Management*, *Handling Software Errors*, *Document Software Error* and *Licensing* (figures A.8, A.10, A.6, A.5, and A.9). Please note: Although the problem and incident management process are initially unrelated, both processes, in the end, rely on process *Document Software Error*. Third, processes *Resolve Software Error* (figure A.14) and *Test Functionality* (figure A.15) form a group. Last, the process *Procurement* (figure A.11) is independent of all other processes. This holds even when we consider the human resource relations, as this process is the only one where the role *Team Assistenz* performs tasks. If we consider further resource relations, the second and the third group are related by using the human resource *SD*. The complete organizational chart is depicted in figure A.17.

The only processes for which instance creation intervals and functions are specified are *Project Realization*, *Problem Management*, *Incident Management*, *Procurement*, and *Resolve Software Error*. Instances of all other processes are created by the invocation of the respective sub-processes.

Please note: The processes *Problem Management* and *Incident Management* contain cyclic structures that may lead to the repetitive execution of multiple tasks.

6.1.2. Example 2: Product Supply Chain

The example described within this section has been adapted from the work of Ferscha and Chiola in [15]. Their original process is modeled using timed Petri nets and has been transformed into BPMN process models. The scenario depicts a simple supply chain involving one producer `P`, three wholesalers `W1 - W3` and nine retailers `A - J`. Three retailers interact with one wholesaler; all wholesalers interact with the single producer. The flow of orders and shipments between the companies is sketched in figure 6.1.

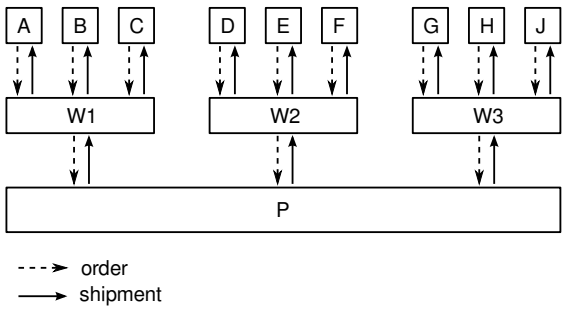


Figure 6.1.: Flow of orders and shipments within the supply chain

Each of the eleven enterprises follows a similar process and has its own human and non-human resources participating in this process. The process of wholesaler `W1` is shown in figure 6.2. On the one hand, incoming orders of the retailers are processed by the incoming order management department. If the wholesaler needs to order things from the producer, a respective order is emitted by the department for outgoing orders. Meanwhile, a worker in the department for outgoing shipments, collects the ordered products and prepares them for deliverance. Before the shipment gets delivered to one of the three producers, an inspector checks it.

On the other hand, incoming shipments from the producer are unpacked by a worker of the department for incoming shipments and afterwards inspected by an inspector. What can be implicitly taken from this explanation, but is not explicitly shown in the model, is a dependency between incoming shipments, emitted orders, and outgoing shipments, i.e. products that are not in the warehouse need to be ordered and the shipment has to be delayed until the respective products have been received.

Obviously, this example describes a setting where the interaction of enterprises is sim-

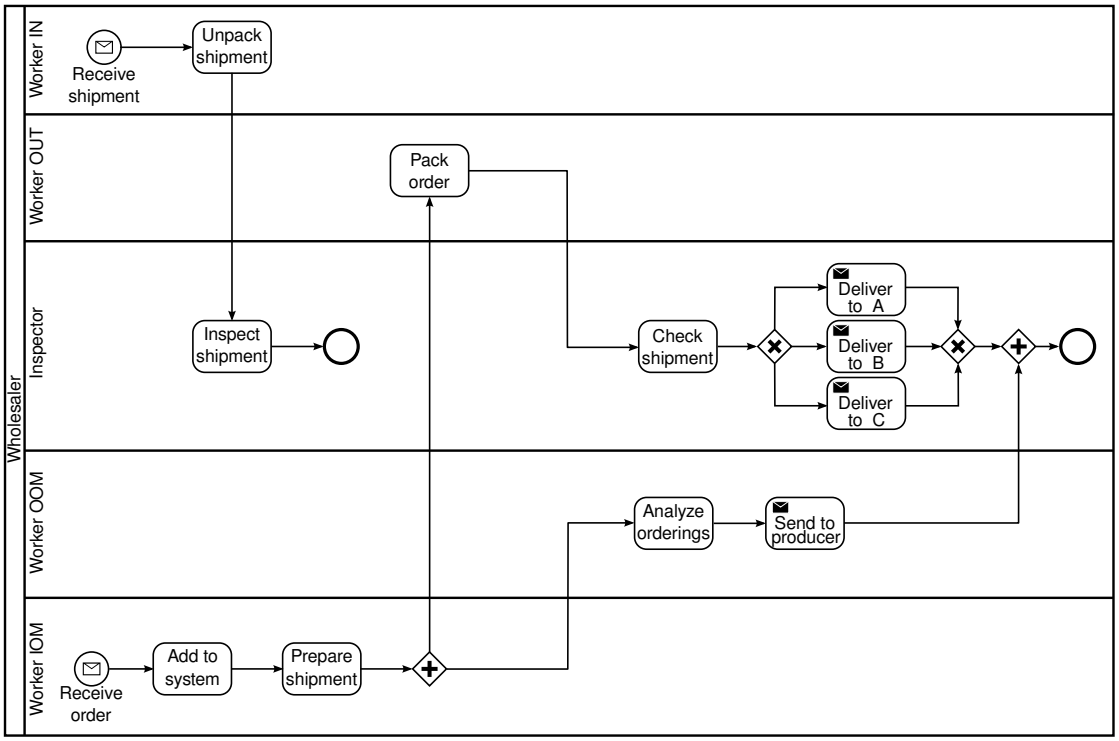


Figure 6.2.: The wholesaler process

ulated, which is, actually, out of scope of this thesis. It is, however, taken into account to represent an example of decoupled processes. Such decoupled processes could also be observed within a single business, if the degree of resource sharing is low.

6.2. Statistics

The examples are evaluated on an Intel XEON processor, having four physical CPU cores with 2.93 GHz and 64-bit architecture. By enabling hyper-threading, there are virtually eight CPUs available. The machine employs 12 GB of main memory and uses openSUSE 11.2 as an operating system. The prototype runs with Java version 6 (1.6.0_24) and Scala version 2.9.1.

For assuring the comparability of the results, the conservative logical process is also used for running the single-engine simulation. Naturally, there will be no message sending, since the logical process will not have any input queues or output buffers.

Analyzing the setting of example 1, we find a high degree of resource relation. Especially the human resource PS is used in several processes. In an initial configuration approximately 1000 process instances are created in a simulation time frame of four weeks.

Running the simulation, we receive the results depicted in figure 6.3a. For the minimum and the bottom-up partitioning the simulation does not terminate, since the cyclic dependencies prevent the important incremental lookahead improvement. However, the simulation using the top-down partitioning approach terminates, but increases the time for simulation by a factor of three. Figure 6.3d depicts the partition graph for this example. The number next to an arc denotes the multiplicity of that arc.

In a second step, the number of instances is step-wise increased. We still simulate 4 weeks, but the instance creation parameters have been adapted. Thereby, there are more concurrent process instances. The simulation results are depicted in figures 6.3b and 6.3c. Surprisingly, for the case of figure 6.3b, the top-down partitioning performs slightly better than the single-engine approach. However, this improvement is not constantly growing, i.e., for the case of figure 6.3c, both approaches perform almost equally good.

For the evaluation of the bottom-up approach for example 1, we remove the cyclic dependencies from the processes *Problem Management* (figure A.10) and *Incident Management* (figure A.8). Further, the process *Handling Software Errors* is removed from

Simulated time: 4 weeks; ≈ 1000 instances

Partitioning	avg. time	avg. wait time	max. wait time
None	4.595	-	-
Minimum	∞	-	-
Bottom-Up	∞	-	-
Top-Down	13.967	6.003	11.974

(a)

Simulated time: 4 weeks; ≈ 2000 instances

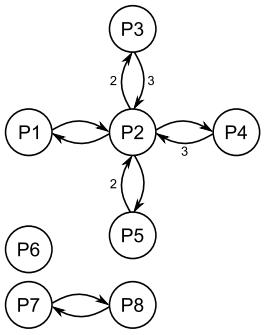
Partitioning	avg. time	avg. wait time	max. wait time
None	18.005	-	-
Top-Down	17.158	5.250	14.706

(b)

Simulated time: 4 weeks; ≈ 4000 instances

Partitioning	avg. time	avg. wait time	max. wait time
None	37.713	-	-
Top-Down	37.269	11.588	36.805

(c)



(d)

Figure 6.3.: (a)-(c) Simulation results for Example 1, and (d) the partition graph for the top-down partitioning

the scenario. Again the number of concurrent process instances is increased step-wise. The results are shown in figure 6.4. For the initial setting the single-engine approach still shows the best performance. For the bottom-up approach the grouping of instance creating transitions (*gi*) is slightly favorable. The top-down partitioning performs better than the bottom-up algorithms but is still slower than the single-engine simulation.

Simulated time: 4 weeks; ≈ 1000 instances

Partitioning	avg. time	avg. wait time	max. wait time
None	4.572	-	-
Minimum	∞	-	-
Bottom-Up	8.903	2.642	6.128
Bottom-Up (gi)	8.553	1.948	6.532
Top-Down	6.334	1.738	4.750

(a)

Simulated time: 4 weeks; ≈ 2000 instances

Partitioning	avg. time	avg. wait time	max. wait time
None	10.083	-	-
Bottom-Up	10.177	4.082	9.696
Top-Down	14.244	3.800	13.346

(b)

Simulated time: 4 weeks; ≈ 4000 instances

Partitioning	avg. time	avg. wait time	max. wait time
None	15.083	-	-
Bottom-Up	25.403	6.844	22.758
Top-Down	15.481	5.230	14.884

(c)

Figure 6.4.: Results for Example 1 - Facilitated version

However, increasing the number of parallel instances reveals the same trend as for the original example. The simulation using the top-down partitioning and the single-engine approach perform equally good. The bottom-up approach performs much weaker.

The partition graphs and the visualization of the simulation execution times of the

facilitated example are shown in figure 6.5.

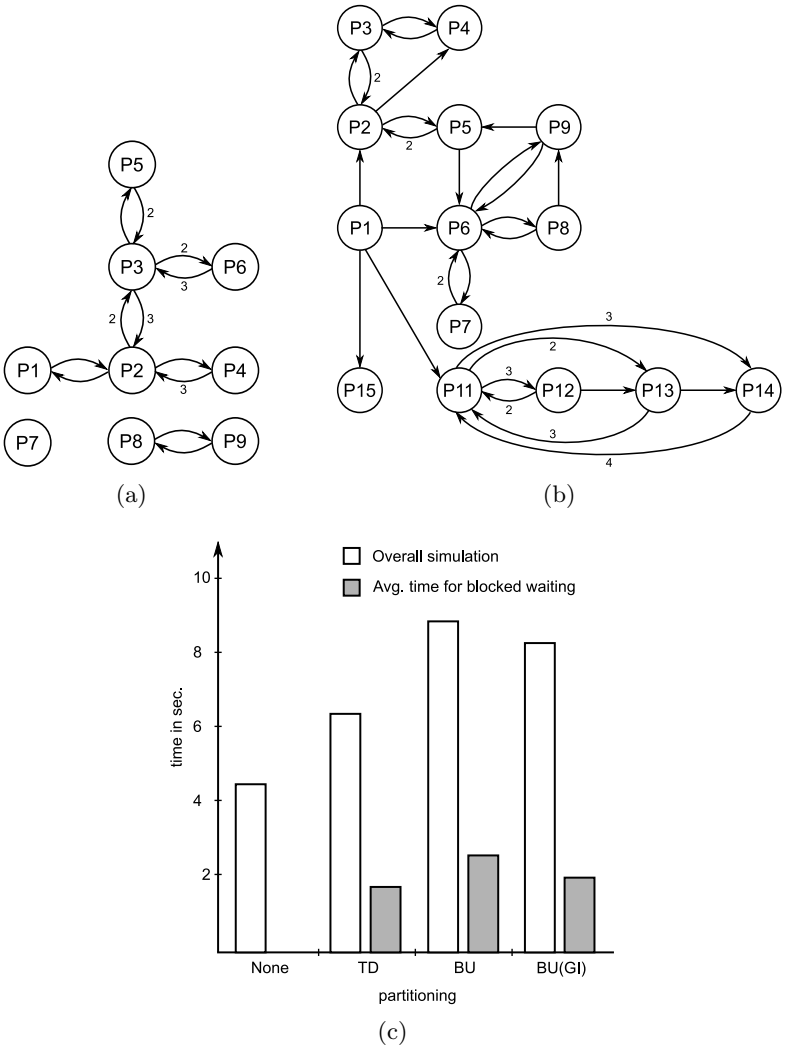
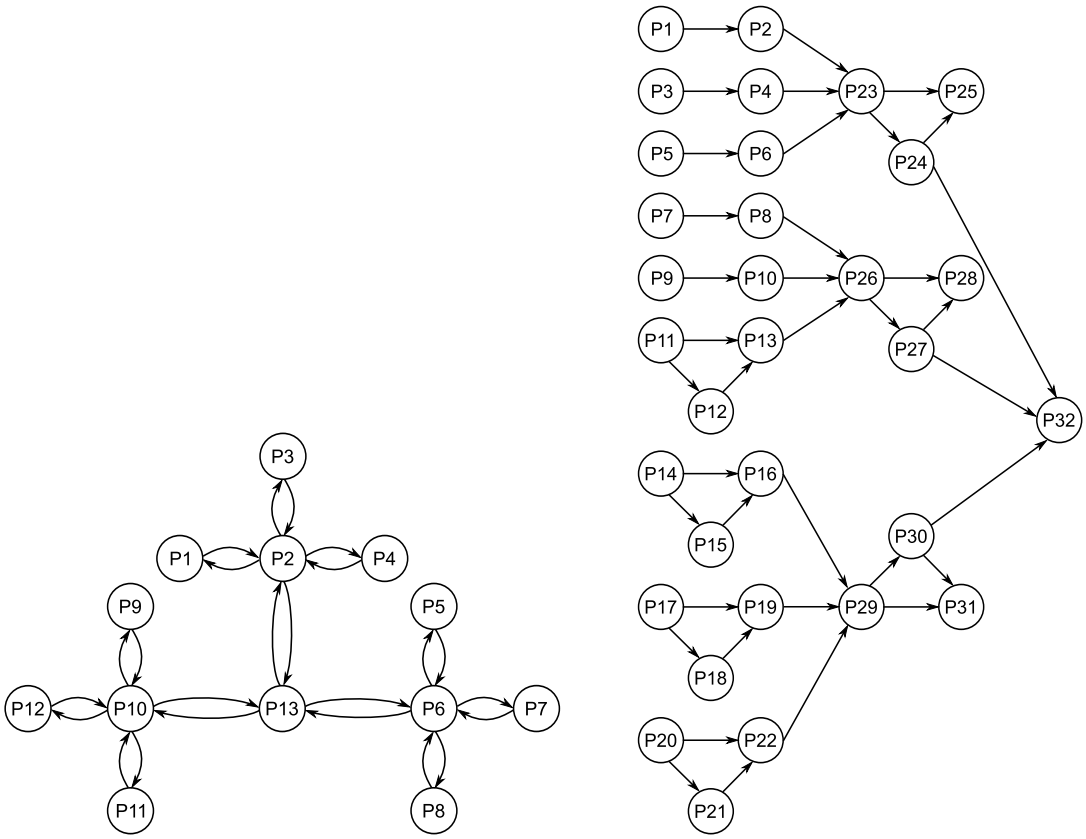


Figure 6.5.: Results for the facilitated example 1: (a) the partition graph for the top-down partitioning, (b) the partition graph for the bottom-up partitioning, and (c) the simulation execution times.

Analyzing the setting of example 2, we observe a low degree of resource relation between the different companies. The example is configured such that each of the nine retailers emits orders every 20 minutes of simulated time. Each process instances lasts about 45 minutes of simulated time. Thus, there are many concurrent instances.

The partition graphs for the partitionings of example 2 are depicted in figure 6.6. For the

top-down approach (figure 6.6a), each company is represented by a single partition. For the bottom-up approach (figure 6.6b) we observe the non-determinism for the mapping of retailers, which are sometimes divided into two partitions, e.g. P1 and P2, and in other cases into three partitions, e.g. P11, P12, and P13.



(a) top-down partitioning (TD)

(b) bottom-up partitioning (BU)

Figure 6.6.: Partitioning results for example 2

The simulation results are shown in figure 6.7. For all partitionings we achieve a better performance than for the single-engine simulation. Interestingly, the minimum partitioning is better than the single-engine case, and also even slightly better than the top-down partitioning. The best performance is achieved by using the bottom-up partitioning without grouping the instance creation.

Simulated time: 4 weeks; ≈ 23000 instances

Partitioning	avg. time	avg. wait time	max. wait time
None	29.360	-	-
Minimum	11.491	4.968	10.716
Bottom-Up	9.776	0.632	3.163
Bottom-Up (gi)	11.647	3.268	5.487
Top-Down	12.133	6.105	8.824

(a)

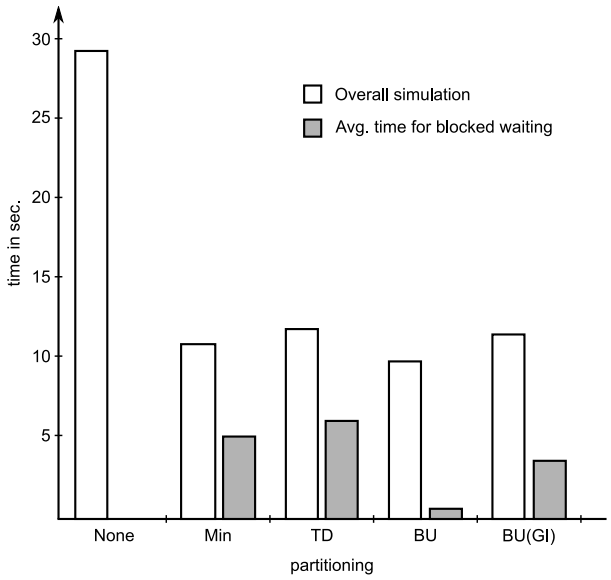


Figure 6.7.: Results for Example 2

6.3. Result Analysis

The evaluation statistics provide versatile results. The evaluation of the second example suggests the benefit and potential of distributing the simulation. For processes that are loosely coupled and a high degree of concurrent instances there is a huge gain in the performance. This setting favors the bottom-up partitioning as it creates smaller partitions than the top-down approach. Further, the minimum partitioning shows good performance, since there are so many concurrent instances.

However, the original use-case of this thesis are scenarios similar to the first example. For

this example, we observe that besides the structural issues of the processes the number of parallel process instances affects the performance of the distributed simulation. The results for the original version of the example show that cyclic structures are problematic and are not sufficiently covered by the proposed algorithms. Further, even in the simplified case, the bottom-up partitioning fails to improve the performance of the simulation. We assume the high degree of resource relation to be the inhibitory factor. This is also represented in the partition graphs where we observe a rather high arc degree, which is also due to the resource relations.

Taking these results as an initial basis, it seems questionable if the automated distribution of business process simulation is generally beneficial. At least, there is no simple straight-forward solution, that performs better than the single-engine approach in most of the settings. In addition, the performance results for the single-engine simulations of our examples are quiet good. Further, the single-engine approach saves time that is spent for partitioning in the distributed case.

The field of distributed simulation is, however, too large to be completely addressed by this thesis. As the next chapter shows, there are several opportunities for future investigations that contribute to the issue of this thesis.

7. Summary and Outlook

This thesis has addressed two aspects of business process simulation. The simulation of business processes is a vital part of business process management and provides a powerful tool for the analysis and improvement of business processes.

For simulating the interplay of multiple processes along with their participating resources, this thesis has proposed a formal model for simulation scenarios. In order to get simulated, the three components of the simulation model, i.e. process models, human resources, and non-human resources, are integrated into a single model that serves as an input for the simulation engine. As a formalism, extended timed colored Petri nets (ETCPN) have been chosen. This thesis has described a mapping of simulation scenarios onto ETCPN. The mapping extends mappings of BPMN process models onto timed colored Petri nets.

For achieving a performance improvement, the distribution of the simulation has been investigated. To enable such distribution, this thesis has proposed two concrete algorithms for partitioning the generated Petri nets. The algorithms have been evaluated using a prototypical implementation that has been created in company with this thesis.

The evaluation results have different implications. First, for processes with a low degree of resource relation, distribution is an appropriate mechanism for accelerating the simulation. Second, if the degree of resource relation is rather high, the presented partitioning strategies tend to impair the simulation performance. Third, cyclic structures within the process models must be handled with high care during partitioning.

Considering only the results of this thesis, it seems questionable if the distribution of business process simulation is able to achieve the performance improvement observed in other domains. However, as this thesis is only able to show an excerpt, there are several opportunities for further investigations that contribute to answering this question.

As the focus of this thesis is put on partitioning, the prototype implements the simplest case of a synchronization protocol. There are, nevertheless, many further approaches for assuring a causally correct distributed simulation. Especially hybrid and adaptive

synchronization protocols should be evaluated. Accompanying the investigations on synchronization, the partitioning algorithms must be further refined. As the evaluation results show, cyclic structures are problematic for the proposed algorithms. In addition, the partitioning strategies should be made aware of timing and instance creation constraints. Another result from the evaluation is the varying performance of the proposed algorithms for different examples. Affecting measures are, e.g., the number of created process instances and the structural and behavioral relations between the processes. Therefore, the properties of simulation scenarios should be analyzed, such that recommendations on partitioning strategies are possible.

This thesis has only considered the distributed simulation using multiple threads on a single machine. A physical distribution onto multiple machines is even more challenging, since the costs for message sending increase. This may influence all aforementioned issues.

Regarding simulation modeling, the working time is the only attribute of human resources supported within this thesis. However, human behavior is a critical factor for appropriate simulation results. Therefore, the model and the mapping should be extended by human skills and the general human work behavior.

Bibliography

[1] *ISO International Standard 8601:2004 : Data elements and interchange formats - Information interchange - Representation of dates and times*, 2004.

[2] Hany H. Ammar and Su Deng. Time warp simulation of stochastic petri nets. In *PNPM*, pages 186–195, 1991.

[3] Ravi Anupindi, Sunil Chopra, Sudhakar D. Deshmukh, Jan A. Van Mieghem, and Eitan Zemel. *Managing Business Process Flows: Principles of Operations Management*. Prentice Hall International, 2nd edition, December 2007.

[4] Joe Armstrong, Robert Virding, Claes Wikström, and Mike Williams. *Concurrent Programming in ERLANG*. Prentice Hall, 1996.

[5] Tim Baines, Stephen Mason, Peer-Olaf Siebers, and John Ladbrook. Humans: the missing link in manufacturing simulation? *Simulation Modelling Practice and Theory*, 12(7-8):515–526, 2004.

[6] R. E. Bryant. A switch-level model and simulator for mos digital systems. *IEEE Trans. Comput.*, 33:160–177, February 1984.

[7] K. M. Chandy and J. Misra. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Trans. Softw. Eng.*, 5:440–452, September 1979.

[8] K. Mani Chandy, Jayadev Misra, and Laura M. Haas. Distributed deadlock detection. *ACM Trans. Comput. Syst.*, 1:144–156, May 1983.

[9] Giovanni Chiola and Alois Ferscha. Distributed Simulation of Petri Nets. *IEEE Parallel Distrib. Technol.*, 1:33–50, August 1993.

[10] Giovanni Chiola and Alois Ferscha. Distributed Simulation of Timed Petri Nets: Exploiting the Net Structure to Obtain Efficiency. In *Proceedings of the 14th International Conference on Application and Theory of Petri Nets*, pages 146–165, London, UK, 1993. Springer-Verlag.

[11] Gero Decker and Mathias Weske. Local enforceability in interaction petri nets. In *Proceedings of the 5th international conference on Business process management*, BPM'07, pages 305–319, Berlin, Heidelberg, 2007. Springer-Verlag.

[12] P. M. Dickens and Jr. P. F. Reynolds. Srads with local rollback. Technical report, University of Virginia, Charlottesville, VA, USA, 1990.

[13] Remco M. Dijkman, Marlon Dumas, and Chun Ouyang. Semantics and Analysis of Business Process Models in BPMN. *Inf. Softw. Technol.*, 50:1281–1294, November 2008.

[14] Sami Evangelista and Jean francois Pradat-peyre. An efficient algorithm for the enabling test of colored petri nets. In *University of Arhus*, pages 137–156, 2004.

[15] A. Ferscha and G. Chiola. Self-adaptive logical processes: the probabilistic distributed simulation protocol. In *In Proc. of the 27 th Annual Simulation Symposium*, pages 78–88. IEEE Computer Society Press, 1994.

[16] Alois Ferscha. Parallel and distributed simulation of discrete event systems. In *Handbook of Parallel and Distributed Computing*, pages 1003–1041, 1995.

[17] Alois Ferscha. Optimistic distributed execution of business process models. In *Proceedings of the Thirty-First Annual Hawaii International Conference on System Sciences-Volume 7 - Volume 7*, HICSS '98, pages 723–, Washington, DC, USA, 1998. IEEE Computer Society.

[18] Alois Ferscha and Satish K. Tripathi. Parallel and Distributed Simulation of Discrete Event Systems. Technical report, University of Maryland at College Park, College Park, MD, USA, 1994.

[19] Richard M. Fujimoto. *Parallel and Distributed Simulation Systems*, volume 1 of *Wiley Series On Parallel And Distributed Computing*. John Wiley & Sons, January 2000.

[20] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley, Boston, MA, 1995.

[21] Object Management Group. Omg unified modeling language (omg uml), superstructure, version 2.3. Technical report, Object Management Group (OMG), May 2010.

[22] Object Management Group. Business process model and notation (BPMN) version 2.0. Technical report, Object Management Group (OMG), January 2011.

[23] Philipp Haller and Martin Odersky. Scala actors: Unifying thread-based and event-based programming. *Theor. Comput. Sci.*, 410:202–220, February 2009.

[24] Vlatka Hlupic and Stewart Robinson. Business Process Modelling and Analysis Using Discrete-Event Simulation. In *Proceedings of the 30th conference on Winter simulation*, WSC '98, pages 1363–1370, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[25] David Jefferson and Henry Sowizral. *Fast concurrent simulation using the time warp mechanism, part 1 local control.* Washington : United States Air Force, 1982.

[26] Kurt Jensen. An introduction to the theoretical aspects of coloured petri nets. In *A Decade of Concurrency, Reflections and Perspectives, REX School/Symposium*, pages 230–272, London, UK, 1994. Springer-Verlag.

[27] G. Keller, M. Nüttgens, and A. W. Scheer. Semantische Prozeßmodellierung auf der Grundlage Ereignisgesteuerter Prozeßketten (EPK). Technical Report 89, Universität des Saarlandes, Germany, Saarbrücken, Germany, January 1992.

[28] Andreas Knpfel, Bernhard Grne, and Peter Tabeling. *Fundamental Modeling Concepts: Effective Communications of IT-Systems.* John Wiley & Sons, 2006.

[29] Stefan Krumnow, Matthias Weidlich, and Rüdiger Molle. Architecture blueprint for a business process simulation engine. In Stefan Klink, Agnes Koschmider, Marco von Mevius, and Andreas Oberweis, editors, *EMISA*, volume 172 of *LNI*, pages 9–23. GI, 2010.

[30] M. H. Jansen-Vullers and M. Netjes. Business Process Simulation - A Tool Survey. In *In Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN*, 2006.

[31] Matteo Magnani and Danilo Montesi. Bpmn: how much does it cost? an incremental approach. In *Proceedings of the 5th international conference on Business process management*, BPM'07, pages 80–87, Berlin, Heidelberg, 2007. Springer-Verlag.

[32] M. Ajmone Marsan. Stochastic petri nets: An elementary introduction. In *In Advances in Petri Nets*, pages 1–29. Springer, 1989.

[33] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes. *Information and Computation*, 100:1–40, September 1992.

[34] Jayadev Misra. Distributed Discrete-Event Simulation. *ACM Comput. Surv.*, 18:39–65, March 1986.

[35] Michael Zur Muehlen. Resource Modeling in Workflow Applications. In *Proceedings of the 1999 Workflow Management Conference (WFM99*, pages 137–153, 1999.

[36] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[37] David M. Nicol and Subhas Roy. Parallel simulation of timed petri-nets. In *Proceedings of the 23rd conference on Winter simulation*, WSC '91, pages 574–583, Washington, DC, USA, 1991. IEEE Computer Society.

[38] Alexandre Nketsa and Nabil Ben Khalifa. Timed petri nets and prediction to improve the chandy-misra conservative-distributed simulation. *Appl. Math. Comput.*, 120:235–254, May 2001.

[39] Martin Odersky, Stphane Micheloud, Nikolay Mihaylov, Michel Schinz, Erik Stenman, Matthias Zenger, and et al. An overview of the scala programming language. Technical report, 2004.

[40] David Parmenter. *Key Performance Indicators: Developing, Implementing, and Using Winning KPIs*. Wiley, January 2007.

[41] C. A. Petri. *Communication with Automata (in German: Kommunikation mit Automaten)*. PhD thesis, Institut für instrumentelle Mathematik, Bonn, 1962.

[42] Frank Puhlmann. Why do we actually need the Pi-Calculus for Business Process Management. In *9th International Conference on Business Information Systems (BIS 2006)*, 2006.

[43] Frank Puhlmann and Mathias Weske. Using the pi-calculus for formalizing workflow patterns. In Wil van der Aalst, Boualem Benatallah, Fabio Casati, and Francisco Curbera, editors, *Business Process Management*, volume 3649 of *Lecture Notes in Computer Science*, pages 153–168. Springer Berlin / Heidelberg, 2005.

[44] Jorgen Randers, editor. *Elements of the System Dynamics Method*. MIT Press, Cambridge, MA, USA, 1980.

[45] N. Russell, Arthur Hofstede, D. Edmond, and Wil V. Aalst. Workflow Resource Patterns. Found at http://www.workflowpatterns.com, 2004.

[46] Nick Russell, Arthur H. M. Ter Hofstede, and Nataliya Mulyar. Workflow controlflow patterns: A revised view. Technical report, BPMcenter.org, 2006.

[47] Thomas J. Schriber and Daniel T. Brunner. Inside Discrete-Event Simulation Soft-

ware: How It Works And Why It Matters. In *Proceedings of the 30th conference on Winter simulation*, WSC '98, pages 77–86, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[48] Robert E. Shannon. *Systems simulation: The Art and Science.* Prentice Hall, June 1975.

[49] Robert E. Shannon. Introduction to the art and science of simulation. In *Proceedings of the 30th conference on Winter simulation*, WSC '98, pages 7–14, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.

[50] Jeff S. Steinman. Breathing time warp. In *Proceedings of the seventh workshop on Parallel and distributed simulation*, PADS '93, pages 109–118, New York, NY, USA, 1993. ACM.

[51] Gregory S. Thomas and John Zahorjan. Parallel simulation of performance petri nets: extending the domain of parallel simulation. In *Proceedings of the 23rd conference on Winter simulation*, WSC '91, pages 564–573, Washington, DC, USA, 1991. IEEE Computer Society.

[52] Kerim Tumay. Business Process Simulation. In *Proceedings of the 27th conference on Winter simulation*, WSC '95, pages 55–60, Washington, DC, USA, 1995. IEEE Computer Society.

[53] W. M. P. van der Aalst. *Timed coloured Petri nets and their application to logistics.* PhD thesis, Eindhoven University of Technology, 1992.

[54] W. M. P. van der Aalst and A. H. M. ter Hofstede. Yawl: yet another workflow language. *Information Systems*, 30(4):245–275, 2005.

[55] Wil van der Aalst. Making Work Flow: On the Application of Petri Nets to Business Process Management. In Javier Esparza and Charles Lakos, editors, *Application and Theory of Petri Nets 2002*, volume 2360 of *Lecture Notes in Computer Science*, pages 1–22. Springer Berlin / Heidelberg, 2002.

[56] Wil M. P. van der Aalst. Making work flow: On the application of petri nets to business process management. In Javier Esparza and Charles Lakos, editors, *ICATPN*, volume 2360 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2002.

[57] Wil M. P. van der Aalst. Business process simulation revisited. In *Enterprise and Organizational Modeling and Simulation - 6th International Workshop , EO-*

MAS 2010, held at CAiSE 2010, Hammamet, Tunisia, June 7-8, 2010. Selected Papers, volume 63 of *Lecture Notes in Business Information Processing*, pages 1–14. Springer, 2010.

[58] Wil M. P. van der Aalst, Arthur H. M. Ter Hofstede, and Mathias Weske. Business process management: A survey. In *Proceedings of the 2003 International Conference on Business Process Management*, BPM'03, pages 1–12. Springer-Verlag, 2003.

[59] Wil M.P. van der Aalst, Mathias Weske, and Dolf Grnbauer. Case handling: A new paradigm for business process support. *Data and Knowledge Engineering*, 53:2005, 2005.

[60] W.M.P. van der Aalst, J. Nakatumba, A. Rozinat, and N. Russell. Business process simulation: How to get it right. In *International Handbook on Business Process Management*. Springer-Verlag, 2010.

[61] Jussi Vanhatalo, Hagen Völzer, and Jana Koehler. The refined process structure tree. In *Proceedings of the 6th International Conference on Business Process Management*, BPM '08, pages 100–115, Berlin, Heidelberg, 2008. Springer-Verlag.

[62] Jianrui Wang and Richard A. Wysk. A pi-calculus formalism for discrete event simulation. In *Proceedings of the 40th Conference on Winter Simulation*, WSC '08, pages 703–711. Winter Simulation Conference, 2008.

[63] Mathias Weidlich, Jan Mendling, and Mathias Weske. Computation of behavioral profiles of process models. Technical report, Hasso Plattner Insititute at the University of Potsdam, 2009.

[64] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer, 2007.

[65] Jr. K. Preston White and Ricki G. Ingalls. Introduction to simulation. In *Winter Simulation Conference*, WSC '09, pages 12–23. Winter Simulation Conference, 2009.

[66] Behrouz Zarei. A partial taxonomic review of parallel discrete-event simulation research. From: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.119.4047&rep=rep1&type=pdf`.

[67] Wlodzimierz M. Zuberek. M-timed petri nets, priorities, preemptions, and performance evaluation of petri nets. In Grzegorz Rozenberg, editor, *Applications and Theory in Petri Nets*, volume 222 of *Lecture Notes in Computer Science*, pages 478–498. Springer, 1985.

A. TurboSoft Inc. Processes

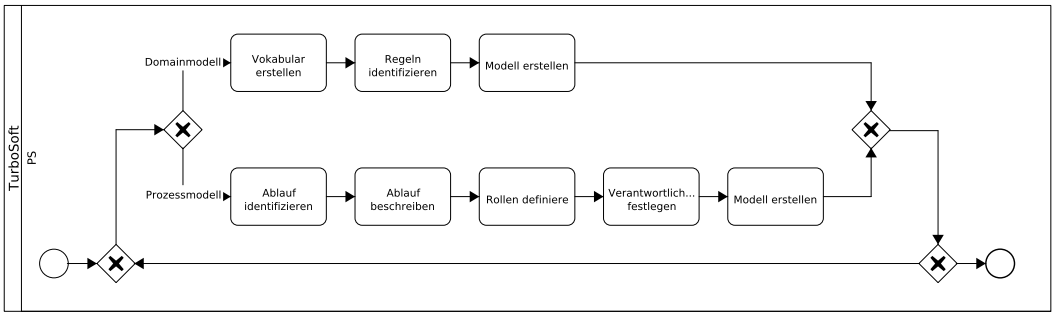


Figure A.1.: Process: Business Modeling

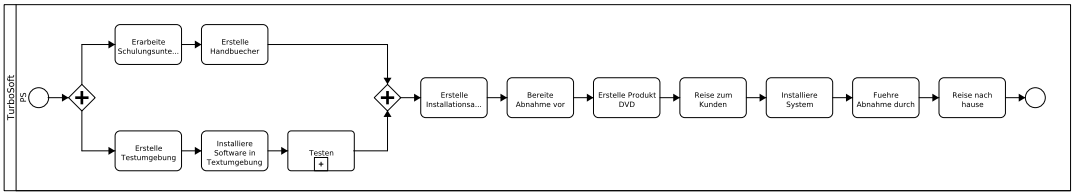


Figure A.2.: Process: Deployment

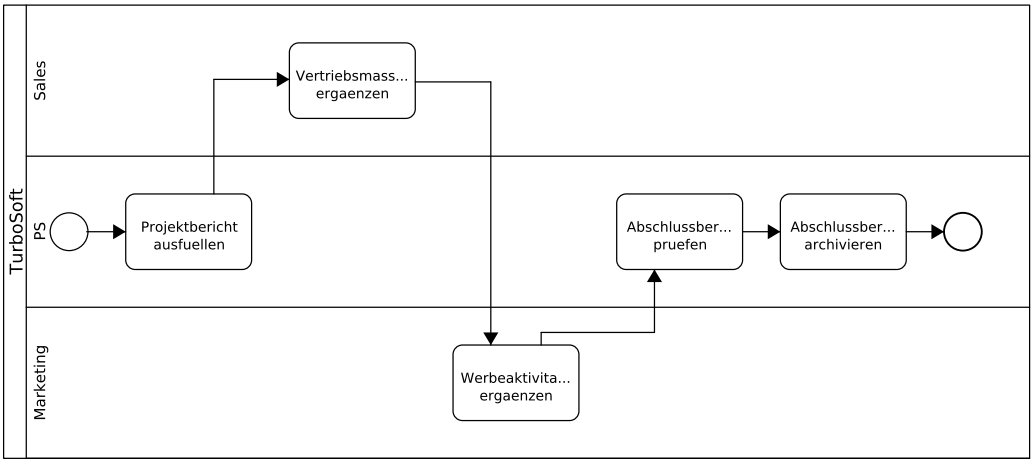


Figure A.3.: Process: Project completion

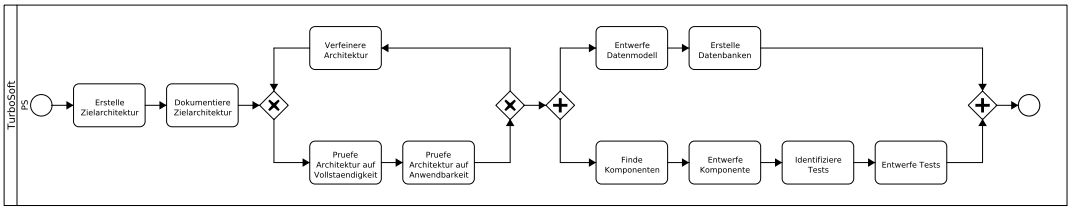


Figure A.4.: Process: Design

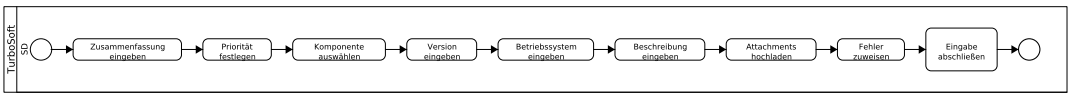


Figure A.5.: Process: Document software error

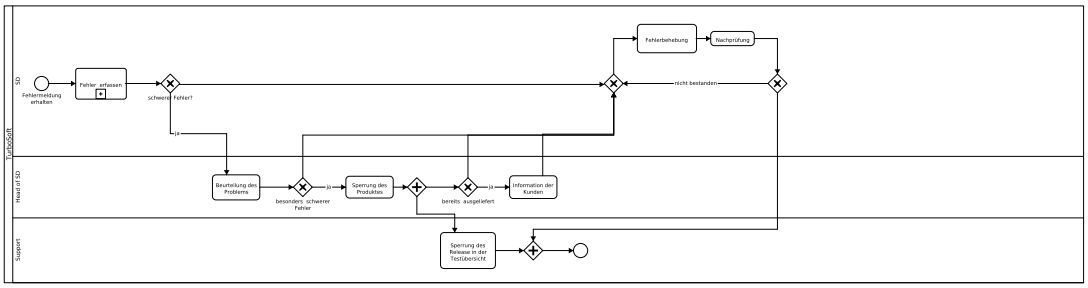


Figure A.6.: Process: Handling software errors

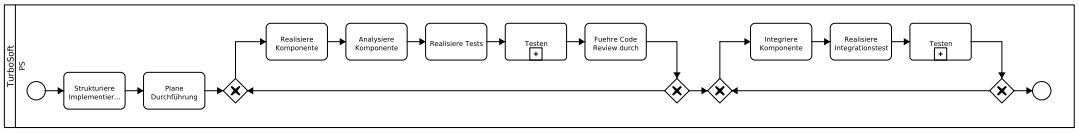


Figure A.7.: Process: Implementation

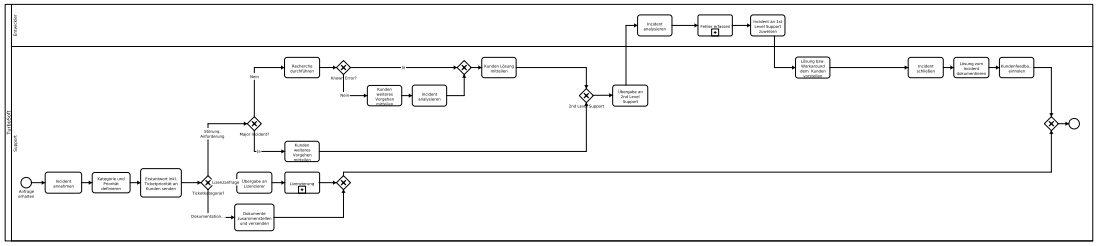


Figure A.8.: Process: Incident management

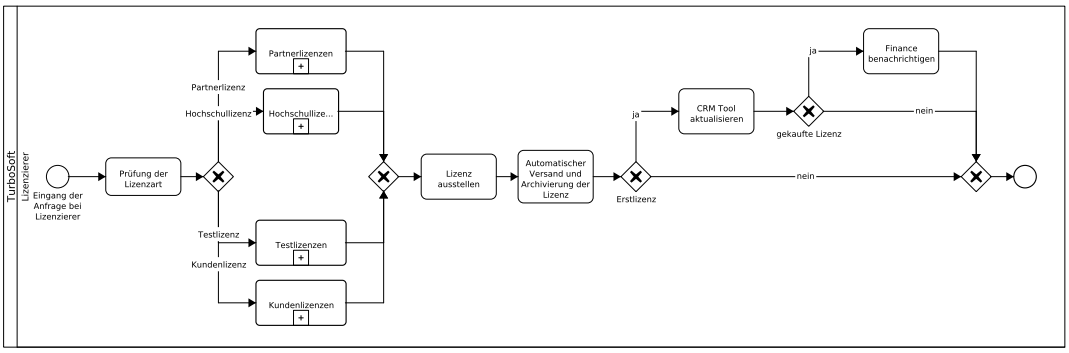


Figure A.9.: Process: Licensing

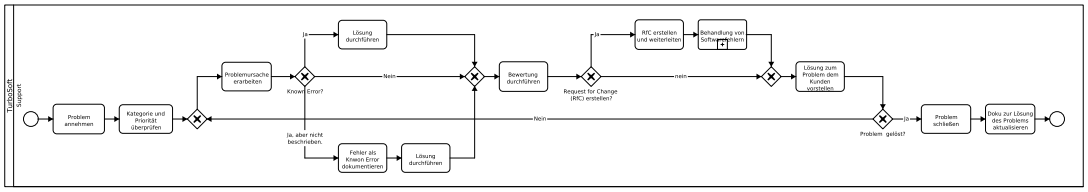


Figure A.10.: Process: Problem management

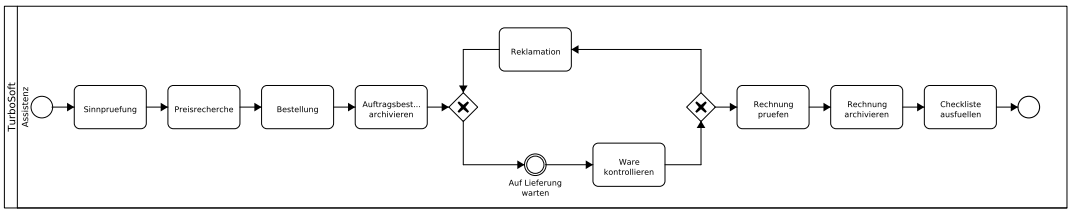


Figure A.11.: Process: Procurement

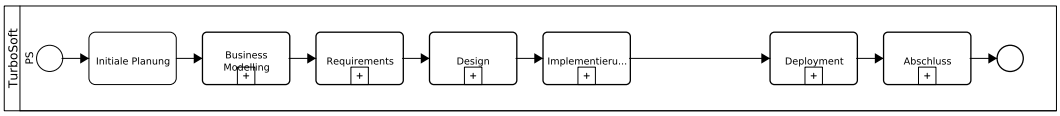


Figure A.12.: Process: Project Realization

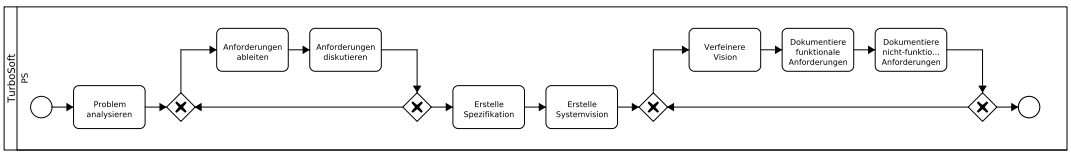


Figure A.13.: Process: Capture Requirement

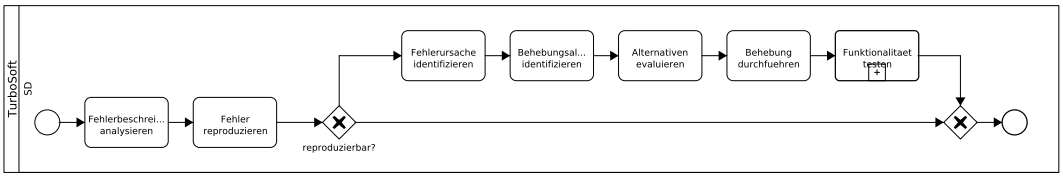


Figure A.14.: Process: Resolve software error

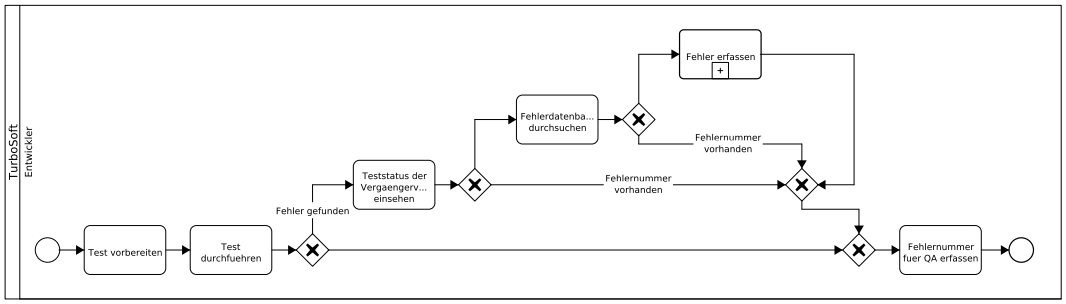


Figure A.15.: Process: Test functionality

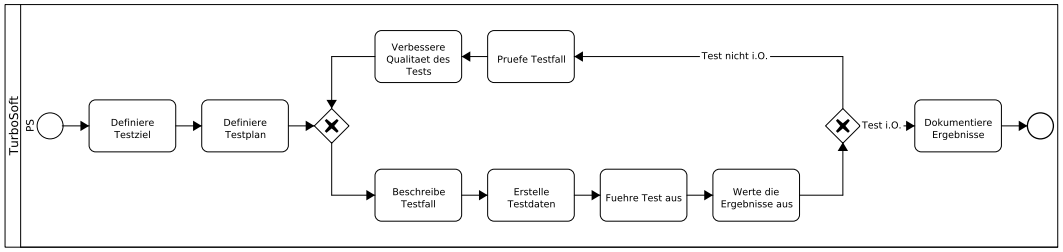


Figure A.16.: Process: Test software

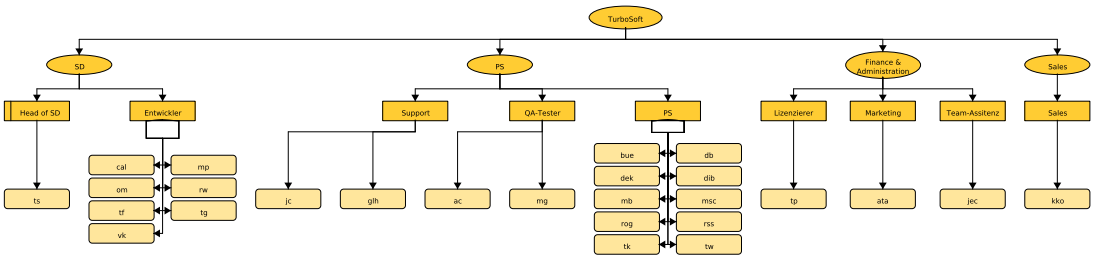


Figure A.17.: Organizational Structure: TurboSoft Inc.

List of Algorithms

1. Basic DES algorithm 20
2. Mapping of Human Resources 44
3. Conservative DDES algorithm 52
4. Minimum Partitioning of a Petri net 56
5. FillPartition(N_i, t, N) 56
6. Bottom-Up Partitioning 60
7. Top-Down Partitioning 63

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Potsdam, November, 2011