Efficient Validation of Process-based Simulation Models

Falko Bause, Jan Kriege, Sebastian Vastag, TU Dortmund, Germany
{falko.bause,jan.kriege,sebastian.vastag}@udo.edu

SNE Simulation Notes Europe SNE 19(2), 2009, 30-38, doi: 10.11128/sne.19.tn.09936

Validation is often time-consuming for simulation models of complex systems especially if failures indicating discrepancies between the system and the corresponding model occur rarely. Some failure types can be detected on the basis of the model's structure employing corresponding efficient techniques. In this paper we present some techniques used in the Collaborative Research Centre 559 ("Modelling of Large Logistics Networks") for the validation of process-based simulation models. These techniques are based on efficient algorithms from the Petri net area, but details are completely hidden from the end user by means of a corresponding toolset. Here we present some internals showing how specific aspects of simulation models can be validated efficiently.

Introduction

Building simulation models of complex systems is a non-trivial task. On the one hand the modeler has to abstract from several details, on the other hand he needs to capture those characteristics of the system which are relevant for the analysis objective. The task even gets more difficult if not fully automated systems being influenced by human decisions and interactions have to be modelled. The main problem is that a simulator is usually a computer program which runs fully automated so that human influence must be captured by rules readable by machines. In the course of the Collaborative Research Centre 559 "Modelling of Large Logistics Networks" (CRC 559; [1, 2]) we made the experience that during the construction of a simulation model several interim versions of the model do not correctly reflect the system behaviour. We found that various discrepancies between the system and the model can be discovered by investigating functional properties [1, 3, 4]. An example is the occurrence of (partial) deadlocks in the simulation model which do not appear in the real system. Especially in models of logistics systems such functional deficits/failures are based on an incorrect modelling for example of human behaviour. A case in point is the well-known concurrent use of a limited number of resources, which are allocated one by one and are only released after having been used (e.g. think of a truck driver who needs a forklift and a free ramp for unloading). The resultant deadlocks are well-known effects in fully automated systems and corresponding simulation models, but normally do not happen in humanly controlled systems.

Certainly, there are several methods for the validation of simulation models (e.g. [5, 6]), but they are often based on the inspection of simulator executions which is time-consuming for large models and in particular if the functional deficits occur rarely. One could think that such rare events can be neglected, since an experienced modeller will detect them in case they really happen, but there are situations where such deficits will go unnoticed. E.g., if faulty simulation models are used in optimisation procedures [7] "optimal" areas might remain undiscovered or, e.g., if (parts of) the simulation models are used as a basis for automated code generation of system control programs the functional deficits are carried over to the real system. In a nutshell, it seems advisable to eliminate such functional deficits from the model.

As mentioned, the usual testing of simulation models is time-consuming concerning the detection of rarely occurring failures, but some failure types can be detected by inspecting the model's structure which can be done efficiently. In this paper we present corresponding techniques which help to validate simulation models with respect to failure types concerning boundedness, liveness and ergodicity of the simulation model. As a base model world we use the process-based model world ProC/B, which has been used within the CRC 559 for the modelling of logistics systems [8]. ProC/B is a modelling language associated with a toolset (cf. Figure 1) which performs validation and simulation of models at the push of a button and hides analysis-specific details from the end user [8, 9]. Here we will look behind the scenes. The detection of functional deficits in ProC/B models is done by mapping those models to Petri nets (PN) keeping essential characteristics [3, 4]. Petri nets [10] are distinguished by very efficient algorithms for checking functional properties and we show how these algorithms can be employed for the validation of ProC/B simulation models.

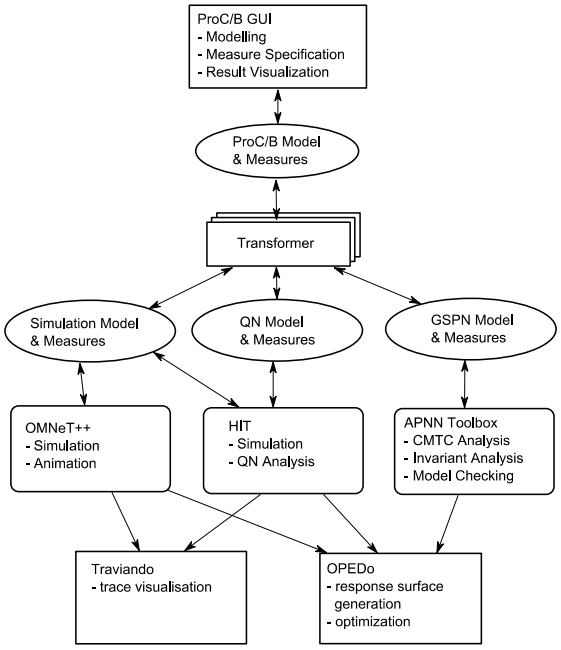

Figure 1. ProC/B Toolset.

This paper is organised as follows: In the next section we briefly introduce the ProC/B model world followed by the main section (Section 2) of this paper where we present an efficient procedure for the validation of ProC/B models. After discussing the general approach we describe the validation in detail with respect to three functional properties: boundedness, liveness and ergodicity. The paper ends with the conclusions in Section 3.

1 ProC/B

Process chains are established for the modelling of logistics networks and also have been the core paradigm within the CRC 559 [11, 12]. ProC/B is a formalization of a subset of this paradigm and was developed with the intention to support an automated analysis of corresponding models accentuating performance aspects. The philosophy of ProC/B is to describe system behaviour by process chains and system structure by functional units. Figure 2 and Figure **3** present a typical example of a ProC/B model [3]. The model is hierarchical and represents a freight village. The top level of the model is shown in Figure **2** where the behaviour of two process types (trucks and trains) is described by corresponding process chains. A process chain consists of several activities modelled by so-called process chain elements (PCEs). A PCE might specify amongst others a pure delay of the process or the call of a service. Services

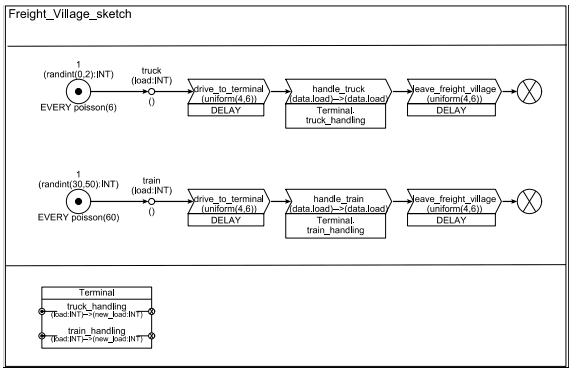

Figure 2. Freight village example.

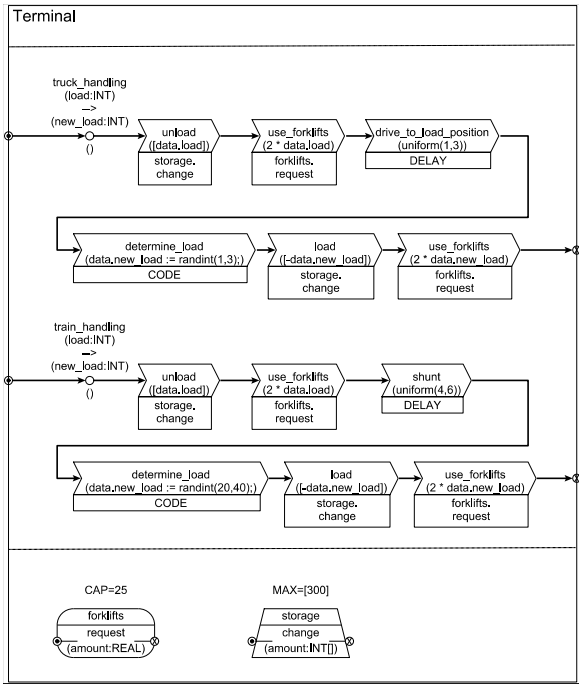

Figure 3. Internals of Function Unit *Terminal***.**

are offered by functional units (FUs) and are described again by process chains whose activities might use services offered by other internal FUs. Figure 3 displays the internals of the FU Terminal whose services are used by trucks and trains (cf. Figure 2). The hierarchical description ends at predefined, so-called standard functional units (cf. Figure 3). ProC/B models might contain two types of standard FUs: servers and counters. Servers (see forklifts in Figure 3) model timing aspects and their behaviour is similar to that of queues in a queuing network. Counters (see storage in Figure 3) model space and a request to a counter is immediately granted if the result respects upper and lower bounds, otherwise the calling process gets blocked until the

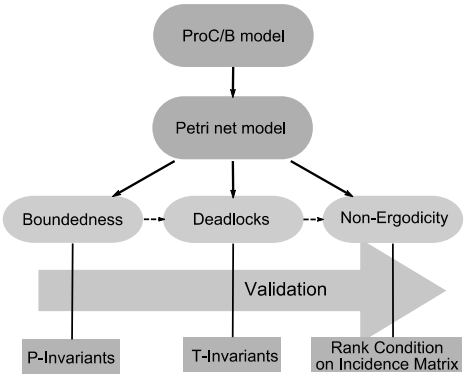

Figure 5. PN techniques for the validation of ProC/B models.

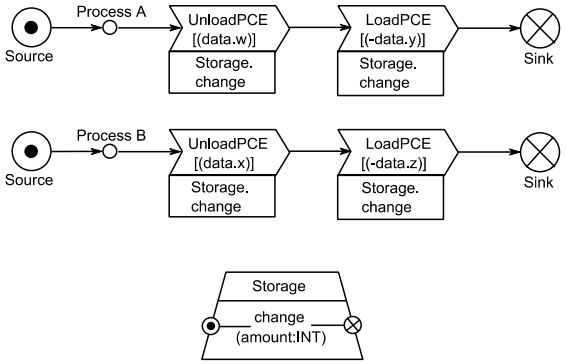

Figure 4. Two processes supplying a storage**.**

change becomes possible. For more details on ProC/B and this specific example we refer the reader to [3] and [8]. As one can imagine ProC/B offers the possibility to describe systems such precisely that an automated analysis is possible.

The modelling and analysis of ProC/B models is accompanied by a toolset which offers a graphical user interface for description and several analysis modules (see Figure 1). Analysis is done by transforming the model specification to the input languages of other tools thus using their analysis capabilities. One such transformation concerns a mapping of ProC/B models to Petri nets. Since an exact mapping would be too complex, only those parts of Proc/B models are captured by the mapping which are primarily relevant for the analysis objectives. E.g., most variables occurring in the ProC/B model are ignored for the transformation, but synchronisation constructs are considered. Therefore, the output of the analysis algorithms might result in so-called non-faults, i.e. faults which hold for the Petri net, but are not occurring in the ProC/B model. Nevertheless such faults or their absence hint at non-validity or validity of the ProC/B model.

2 Validation of ProC/B Models

ProC/B was developed with user-friendliness in mind. It can be used by non-experts to form even complex models of logistics networks and their working processes. Surely with increasing complexity of the model also the possibility of errors in the model increases and appropriate support is needed. The ProC/B Toolkit features several techniques for the validation of models. The methods we present in the following do not intend to check, e.g., whether an accurate representation of input data has been chosen, but try to support a plausibility check for the model internals.

2.1 General Approach

Validation of ProC/B models is here based on a transformation from process chains to Petri nets. Figure 5 shows how validation is performed: An existing ProC/B model is converted to a Petri net model. The new representation is used for validation with respect to Petri net properties. In particular support is offered for checking boundedness, identifying deadlocks and searching for non-ergodic behaviour. The applied techniques are completely hidden from the user, so no knowledge on Petri nets or any functional property is required. The first step in the validation process is to transform the ProC/B model to a Petri net representation. For each language element there is a blueprint of PN parts to be placed instead of the original element. For example, Figure 4 is the original ProC/B model of a typical stock-keeping scenario. The corresponding Petri net is shown in Figure 6.

Details on the transformation can be found in [3, 4, 8]. Petri nets, originally introduced by Carl Adam Petri [13], are a formalism for the description of concurrency and synchronisation aspects in systems [10]. They describe behaviour of systems by the states that can occur, but usually neglect timing aspects, although variants of Petri nets exist which also consider time (see e.g. [14]). A common variant of Petri nets are so-called Place-Transition nets. A Place-Transition net is a five-tuple $P = (P, T, C^+, C^-, m_0)$ which can be interpreted as a graph containing two different types of nodes: places P and transitions T. Connections are defined by two incidence matrices: backward incidence matrix $C^- = \left(c_{ij}^-\right) \in \mathbb{N}_0^{|P|\times|T|}$ and forward incidence matrix $C^+ = \left(c_{ij}^+\right) \in \mathbb{N}_0^{|P|\times|T|}$. If $c_{ij}^- > 0$ an arc with weight c_{ij}^- leads from place i to transition j. Similarly, element c_{ij}^+ gives the weight of an arc from transition j to place i.

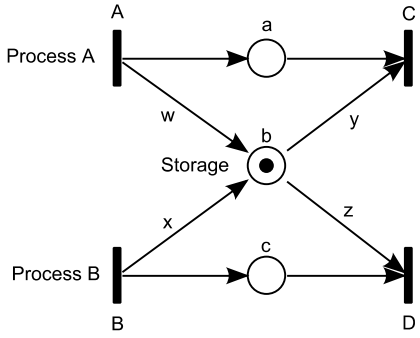

Figure 6. Petri net representing two process chains and one storage**.**

Places are initially marked with the contents of the positive vector $m_0 \in \mathbb{N}^{|P|}$. A graphical representation of Petri nets uses circles for places and bars for transitions. The positive elements of the incidence matrices are shown as directed arrows given a weight. The marking $m(p)$ of a place $p \in P$ can be seen as tokens placed on the place as solid dots.

Enabled transitions change the marking of places that are connected by arcs via "firing". Transition j is enabled in marking m if there are enough tokens available on input places: $m \geq C^- \cdot e_j$ with e_j as the j-th unit vector. Firing transition t will destroy $C^- \cdot e_j$ tokens from all input places and generate $C^+ \cdot e_j$ tokens on the output places.

Figure 6 shows the Petri net derived from the storage system in Figure 4. It contains four transitions A-D and three places a, b and c. Transition A has two outgoing arcs, while place b has two incoming and two outgoing arcs. There are arcs with weights $w - z$, specifying that adjacent transitions produce or consume multiple tokens per firing. In a concrete ProC/B model w-z will be concrete values, but we will keep the notation with variables in the following to consider different modelling failures. In case the model specifies stocked or removed storage units by random variables, average or user specified values will be used in the Petri net representation.

Markings are central in Petri net theory as they express the state of the net. One goal of Petri net analysis is to check whether unwanted markings can be reached. The set of all reachable markings is given by the initial marking m_0 and the firing rule specifies which markings can be reached from a given marking by firing transitions. The total effect firing transitions have on the marking is described by the incidence matrix

$$C = C^+ - C^-$$

For example, the incidence matrix C of the storage system can be written as:

$$C = \begin{bmatrix} 1 & 0 & -1 & 0 \\ w & x & -y & -z \\ 0 & 1 & 0 & -1 \end{bmatrix}$$

The effect of firing transition t_j at marking m is given by the j-th column of matrix C and can simply be calculated as follows. Let e_j denote the j-th unit vector. Then the product $C \cdot e_j$ gives the j-th column, so that the successor m' of a marking m can be calculated by

$$m' = m + C \cdot e_j$$

Since for every reachable marking there exists a firing sequence of transitions, the corresponding unit vectors can be subsumed in a linear combination $\bar{q}$ (Parikh vector [15]).

Thus the state equation of a Petri net with initial marking m_0, firing vector $\bar{q}$ and incidence matrix C can be written as

$$m = m_0 + C \cdot \bar{q}$$

Some very efficient Petri net analysis techniques are based on the investigation of this incidence matrix C. For example, the state equation gives us a necessary condition whether a marking m_f related to an unwanted model state can be reached. E. g., setting $\bar{q} = x, m = m_f$ gives

$$C \cdot x = m_f - m_0$$

and if no positive integer solution for x exists then m_f can not be reached from the initial marking m_0.

Another option for Petri net analysis are reachability graphs. Reachability graphs have markings as nodes and two nodes m and m' are connected with a directed arc labeled t if marking m' is reachable by firing transition $t \in T$ enabled in m. A reachability graph can be constructed by generating the reachability set $RS(PN, m_0)$ starting at initial marking m_0 as the root node and adding reachable markings as leaves for each enabled transition. This step is repeated at the leaves and the resulting tree is later simplified to a graph $RG(PN, m_0)$ by merging equivalent nodes. Usually properties of a Petri net are defined on the basis of the reachability graph/set, so that its generation is a common option for analysis in case the set is finite which means that the Petri net is bounded (cf. Section 2.2). The main problem is that even simple Petri nets can have large reachability graphs/sets and their handling would require a lot of memory and CPU time.

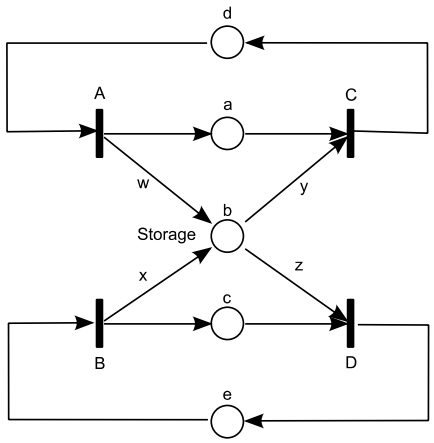

Figure 7. Modified Petri net PN_C.

Petri net theory also offers other analysis methods that can be chosen according to the actual requirements and area of application. The following two properties are checked with a very prominent technique based on the inspection of the structure of the Petri net, namely invariant analysis.

2.2 Boundedness

Boundedness is a property of Petri nets useful to test models of production and storage systems.

In the logistics model world, a bounded system will output the same number of goods as the number of goods entering it (or for manufacturing systems: be at least in a fixed relation). Several types of errors can occur when this property is not satisfied: imagine the model is faulty in the way that goods are not removed when they are accomplished. Concerning Figure 6 this might happen if $w + x > y + z$ which might cause the number of tokens on place b to increase to infinity. Showing that the number of goods is not bounded would indicate that the modeler has forgotten to organise the outgoing transports in an appropriate way.

A Petri net PN is called bounded if the number of tokens on each place is upper bounded by k at every reachable marking, i.e.

$$\exists k \in \mathbb{N}: \forall m \in RS(PN, m_0): \forall p \in P: m(p) \leq k$$

With a limited number of tokens at each place the set of reachable markings is also bounded:

$$\exists k \in \mathbb{N}: |RS(PN, m_0)| < k$$

Since almost all logistics systems (and thus ProC/B models) are open systems, those systems are not bounded in principal. Nevertheless checking for boundedness in an appropriately modified model helps to find modelling errors. As part of our validation approach the Petri net is modified to a closed net by short-circuiting the transitions representing the source and the sink of a process chain. The result of this modification on the net in Figure 6 is shown in Figure 7. Since we now have a closed Petri net checking for boundedness makes sense.

A sufficient condition showing the boundedness of the Petri net can be deduced from the state equation. Multiplying the equation with $v \in \mathbb{N}^{|P|}$ we get

$$v^T \cdot m = v^T \cdot m_0 + v^T \cdot C \cdot \bar{q}$$

and choosing a vector v with $v^T \cdot C = 0$ establishes a condition on all reachable markings:

$$v^T \cdot m = v^T \cdot m_0.$$

Such a vector $v \neq 0$ is called a place invariant.

Place invariants covering places p_i with $v_i \neq 0$ fix the ratio of tokens on places no matter which marking is reached. A Petri net is said to be covered with a positive place invariant v if

$$\exists v \in \mathbb{N}^{|P|}: v^T \cdot C = 0 \quad \text{and} \quad v > 0.$$

The existence of such positive invariants gives us a sufficient condition for boundedness: a Petri net is bounded when it is covered by a positive place invariant implying that the weighted number of tokens is constant at all markings.

We are going to check PN_C (cf. Figure 7) for boundedness. The new incidence matrix C_C of the modified net of Figure 7 is:

$$C_C = \begin{bmatrix} 1 & 0 & -1 & 0 \\ w & x & -y & -z \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$$

For general values of w-z, place invariants of PN_C are $v_1 = (1,0,0,1,0)$, $v_2 = (0,0,1,0,1)$ because $v_1 \cdot C_C = v_2 \cdot C_C = 0$. Invariants v_1 and v_2 do not cover place b as there is no linear combination of invariants (which is itself an invariant) covering the second element. The uncovered place b is associated with the ProC/B storage in Figure 4 and the ProC/B toolset will mark model elements that are not covered with invariants as potentially unbounded. The user has the option to resolve this warning, e.g. by setting a limit for the maximum storage capacity.

This step might be unnecessary when each process loads the same number of goods it unloads, i.e. if $w = y$ and $x = z$. Under these conditions there are

two more invariants, $v_3 = (y, -1, x, 0, 0)$ and $v_4 = (-y, 1, 0, 0, x)$. They can be combined to $2y\ v_1 + v_3 + 2v_4 = (y, 1, x, 2y, 2x)$ covering all places in PN_C. In this case the ProC/B toolset would not output a corresponding warning, indicating that the situation seems to be basically modeled correctly.

2.3 Deadlocks

Deadlocks might be caused by processes being dependent on each other and concurrently waiting for each other. This is a good example of errors easily solvable by humans and being problematic in computer simulations [16]. Consider the loading/unloading process model of Figure 4 and the derived closed Place-Transition net PN_C of Figure 7. PN_C deadlocks if $w + x < y + z$ since eventually a marking m_d will be reached with

$$m_d < C^- \cdot e_j, \forall j$$

Thus also transitions C and D representing loading processes have to stop. In the real system these load processes might represent trucks or trains which have to pursue a tight timetable and surely there will be some responsible person, e.g. the driver, solving this "deadlock situation". Even though the original ProC/B model represents an open system and thus will not deadlock, the occurrence of a deadlock in the closed Petri net indicates a model aspect which should be checked by the modeller.

Deadlocks are related to the liveness property. A transition t is live at marking m if

$$\exists m_r \in RS(PN, m): m_r \geq C^- \cdot e_t,$$

i.e. that one can always reach a marking, starting from m, so that transition t is enabled. A Petri net is live if all transitions $t \in T$ are live at all reachable markings of the Petri net. Obviously, in a live Petri net no deadlocks can occur.

Invariant analysis gives a necessary condition for liveness. After checking the closed loop net PN_C for boundedness we know that its reachability set is finite. The corresponding finite reachability graph thus consists of one or several strongly connected components and in each such component all transitions must occur as labels if the Petri net is considered to be live. Since we can reach any node/marking in a strongly connected component from any other marking of this strongly connected component, each marking m can be repeatedly reached. In terms of the state equation this means

$$\exists \bar{q} \in \mathbb{N}^{|P|}: m = m + C \cdot \bar{q}$$

which only holds if $C \cdot \bar{q} = 0$. Such a vector $\bar{q}$ is called a transition invariant. If a bounded Petri net is live then one can show that it is covered with at least one transition invariant $\bar{q} > 0$ [14, 15]. This implies that if one does not find a positive solution $\bar{q} > 0$ for $C \cdot \bar{q} = 0$ the Petri net is not live.

A valid transition invariant for the net PN_C in Figure 7 is $v_1 = (1,0,1,0)$ assuming $w = y$. It covers transitions A and C belonging to process A. Both transitions are live when the loaded quantity equals the unloaded. Of course, a similar invariant $v_2 = (0,1,0,1)$ under condition $x = z$ exists for transitions B and D. With $w = y$ and $x = z$ $v_1 + v_2$ is also an invariant covering all transitions and a closer inspection shows that PN_C is live. Apart from the symmetric case $w = y$ and $x = z$, invariant $v_3 = (x - z, y - w, x - z, y - w)$ exists assuming goods are exchanged between both process chains and $w \neq y$ and $x \neq z$. $v_3 > 0$ e.g. holds if $x > z$ and $y > w$. So in summary, the closed Petri net of Figure 7 is only covered by positive transition invariants if the quantities for unloading and loading match. The ProC/B toolset will mark corresponding uncovered ProC/B model elements thus indicating those model parts which should be inspected more carefully by the user.

Coverage by positive transition invariants is a necessary condition for liveness in bounded Petri nets. There are also Petri net techniques available giving characterising conditions for liveness or the existence of deadlocks. We only want to mention two here: the investigation of special classes of Petri nets (see [14, 17]) and the partial exploration of the reachability set/graph (cf. [18]).

Net classes are specified by imposing restrictions on the interconnection of places and transitions. For several net classes checking for deadlocks or liveness can be done very efficiently on the basis of the net's structure. As an example we consider the net of Figure 8, which belongs to the class of state machines.

The net class of state machines contains Place-Transition nets with transitions having only one incoming and one outgoing arc. Let $\cdot t$ be the set of input places and $t \cdot$ the set of output places of transition $t \in T$. A Place-Transition net is called a state machine if and only if

$$\forall t \in T: |\cdot t| = |t \cdot| = 1$$

Figure 8 shows an example of a state machine PN_{SM}. The criteria for liveness in state machines is that the net considered as a graph is strongly connected and

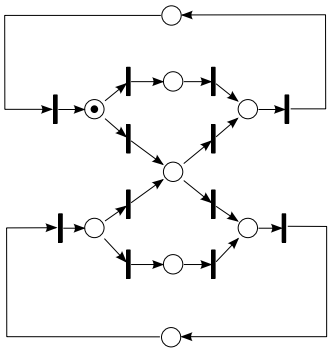

Figure 8. State Machine PN_{SM}.

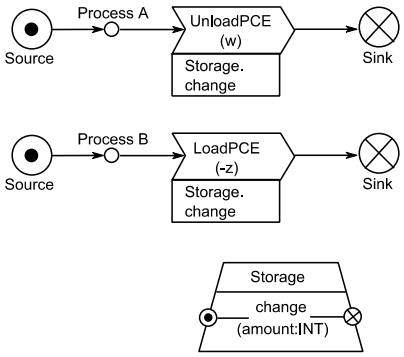

Figure 9. Simple non-ergodic process chain model.

$m_0 \neq 0$. One characteristic of state machines is that firing does not change the number of tokens in the whole net, i.e.

$$\exists k \in \mathbb{N}: \forall m \in RS(PN_{SM}, m_0): |m| = k$$

and thus obviously a live state machine is also bounded. A token on a place will always enable at least one transition, so the net is live at m_0 with just a single token that can move around freely. Therefore the initial marking with one token as shown in Figure 8 gives a live Petri net PN_{SM}. Similar conditions based on the structure of the net are known for other net classes as well (cf. [14]).

The stubborn set method [18] is a method which only needs to partially explore the reachability graph of the Petri net in order to verify for specific properties, e.g. the absence or existence of deadlocks. The main idea is that only a small part of the state space is explored, and the exploration is made such that all deadlock states of the whole reachability graph are also part of the smaller subset. Looking for deadlocks is then only necessary in the smaller subset.

2.4 Non-Ergodicity

Non-ergodicity can be observed in models of logistics networks in situations with an interdependence between two or more processes. This interdependence is typically caused by a synchronisation between the processes or by stock-keeping scenarios.

Figure 9 shows a very simple process chain model consisting of two process chains and a storage. The upper process chain unloads goods to the storage, while the lower process chain loads goods from the storage. In this scenario process A can be interpreted as a server for process B and vice versa, since process A delivers goods that are loaded by process B and process B frees storage space needed by process A to unload. Assume the case $z = w$ and that arrivals of process A occur at rate λ and arrivals for process B at rate μ. Then μ is the service rate for process A and λ the service rate for process B. From queueing theory it is known that for process A (with arrival rate λ and service rate μ) $\lambda < \mu$ has to hold for a steady-state distribution to exist. At the same time the condition $\mu < \lambda$ has to hold for process B, which already demonstrates that this type of situation is problematic.

A similar situation occurs here if the average number of delivered and loaded units differ, i.e. if $z \neq w$.

Non-ergodicity implies that the steady-state distribution does not exist and thus non-terminating simulations are useless for those models. In general non-ergodicity is not a surprising effect when dealing with overload situation to determine the model's peak performance. In these cases an appropriate choice of model parameters yields an ergodic model, but for logistics networks typical situations exist (cf. Figure 9) where non-ergodicity is an intrinsic characteristic of the model and cannot be avoided by selecting different parameters for e.g. the interarrival times. In most of these cases non-ergodicity implies an incorrect modelling of the system resulting from the negligence of characteristics of the system.

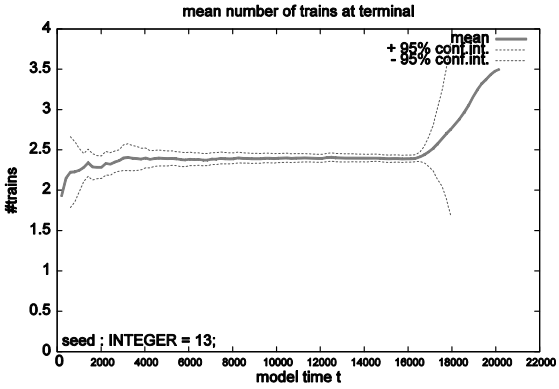

Figure 10. Simulation result of a non-ergodic model.

Figure 10 shows a simulation result of a non-ergodic model of a freight village taken from [3]. The figure indicates that non-ergodicity is difficult to detect by simulation, since the result seems stable for a long period of time and the simulation might even have been stopped before the non-ergodic behaviour became visible from the results. Hence, it would require very long simulation runs and a large amount of CPU time to detect non-ergodic behaviour by simulation, if it is detected at all.

For Petri nets an efficient technique for the detection of potentially non-ergodic models is available [19] which is based on rank computations for the incidence matrix of the Petri net. This technique can also be applied to process chain models as described in [3].

Let m denote the number of transitions of the Petri net and $N(s) \in \mathbb{R}^m$ be a vector counting the number of transitions firing in the time interval $[0, s]$. For an ergodic Petri net the mean firing flow vector

$$N := \lim_{s\to\infty} E[N(s)]/s$$

exists and the expected input flow of tokens at a place equals the expected output flow, which can be expressed using the incidence matrix C resulting in $C \times N = 0$. The kernel of matrix C is defined as

$$kernel(C) := \{x \in \mathbb{R}^m | \, C \times x = 0\}$$

and thus N is in the kernel of C.

In general the computation of N is difficult, but for some transitions the corresponding values of N can be determined easily. This holds for source transitions and sets of transitions that partially exhibit an Equal-Conflict (PEC set), i.e. for transitions with the property that at any marking either all or none of those transitions are enabled.

Figure 11 shows the Petri net representation of the process chain model from Figure 9. For the two source transitions the components of vector N can be determined easily and a computation of the basis of the kernel of the incidence matrix shows that the firing rates of those transitions are dependent. The basis of $kernel(C)$ is given by (z, z, z, w, w, w) for the Petri net from Figure 11, where the first three entries correspond to the transitions of process A and the last three entries to the transitions of process B, implying a dependence between the source transitions (here $a \in \mathbb{R}$ has to exist with $\lambda = az$ and $\mu = aw$). Thus, the Petri net of Figure 11 is sensitive towards small changes of the firing rates of the two source transitions.

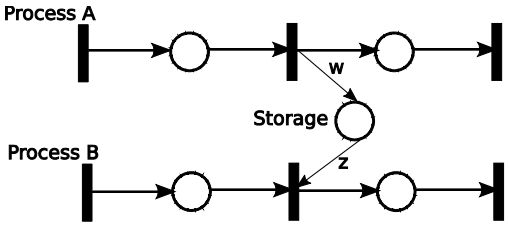

Figure 11. Petri net representation of the process chain model from Figure 9**.**

This kind of sensitivity is called *e-sensitivity* in [19] and indicates potential non-ergodic nets. Furthermore a formal criterion for detecting such nets is given. Let $k_1, \ldots, k_r$ be a basis of the kernel and $\tilde{T}$ a PEC set. Then a Petri net is e-sensitive if

$$rank\left(\left(Proj(k_1, \tilde{T}) \ldots Proj(k_r, \tilde{T})\right)\right) < |\tilde{T}|$$

holds, where $Proj$ denotes the projection of vector k_i onto $\tilde{T}$.

For potentially non-ergodic nets the approach identifies a set of transitions implying e-sensitivity. Since each of the transitions corresponds to an element of the process chain model, they can be used to identify the critical part in the process chain model.

Applying this approach to the model of the freight village introduced in Section 1 identifies the model as being potentially non-ergodic too. As already mentioned this hints at an incorrect modelling of the system, e.g. we ignored existing time tables and delivery schedules for the trucks and trains in this model.

Ergodic models can be found when restrictions on concrete values are modified. Figure 12 shows an ergodic system if e.g. $\lambda(w - y) < \mu(z - x)$ with $w > y$, $z > x$. The loading process chain element of process B uses the alter-or-skip service [20]: it will pick up all available, but not more then z goods from the storage. This allows process B to continue even if the storage is empty. For details on the theoretical background the interested reader is referred to [19]. [3] explains how the approach can be automatically applied to ProC/B models.

3 Conclusions

In this article we gave insight into some possibilities for the validation of process-based models as being offered by the ProC/B toolset. Validation is based on the automated transformation of ProC/B models into similar behaving Petri nets and usage of corresponding Petri net analysis techniques. Due to the complexity of realistic ProC/B models (and simulation

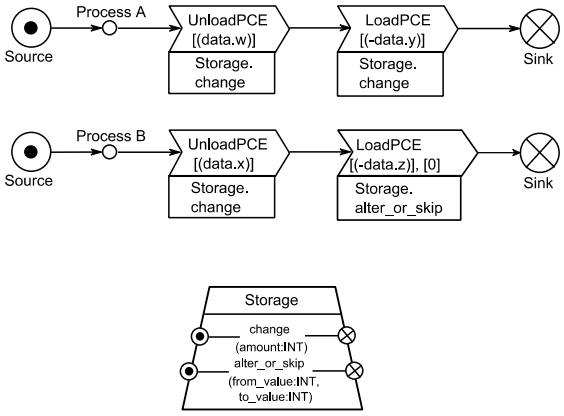

Figure 12. Simple ergodic process chain model**.**

models in general) the transformation does not account for all details of the ProC/B model. This might result also in non-faults, i.e. faults occurring in the Petri net, but being nonexistent in the ProC/B model, so that the output of the Petri net analysis has to be considered as an indication of possible errors in the ProC/B model. The essential advantage of the presented Petri net techniques is their efficiency. The analysis investigates the structure of the Petri net and renders the generation of the state space unnecessary, so that a modeler is able to validate specific model aspects during the construction phase within a short time. As Figure 4 suggests, one might first apply invariant techniques, since this step might result in a change of the model also having impact on the model's ergodicity. Once those tests are passed, ergodicity might be checked, since subsequent corrections usually do not change the invariants.

Acknowledgements

The work described in this paper was supported by the Deutsche Forschungsgemeinschaft as part of the Collaborative Research Center "Modelling of Large Logistics Networks" (559).

References

[1] P. Buchholz, U. Clausen. *Große Netze der Logistik – Die Ergebnisse des Sonderforschungsbereichs 559*. Springer, 2009.

[2] Collaborative Research Center. *Modelling of Large Logistics Networks* (559). http://www.sfb559.uni-dortmund.de.

[3] F. Bause, J. Kriege. *Detecting Non-Ergodic Simulation Models of Logistics Networks*. Proc. 2nd Int. Conference on Performance Evaluation Methodologies and Tools (ValueTools 2007), Nantes, 2007.

[4] P. Kemper, C. Tepper. *A Petri net approach to debug simulation models of logistic networks*. In Proc. 5th Mathmod Conference, Vienna, February 8-10, 2006.

[5] A.M. Law. *Simulation Modelling & Analysis. McGraw-Hill*, 4th ed., 2006.

[6] M. Rabe, S. Spieckermann, S. Wenzel. *Verifikation und Validierung für die Simulation in Produktion und Logistik – Vorgehensmodelle und Techniken.* Springer, 2008.

[7] M.C. Fu, C.-H. Chen, L. Shi. *Some topics for simulation optimisation*. Proc. 2008 Winter Simulation Conference, Miami, 2008, pp. 27—38.

[8] F. Bause, H. Beilner, M. Fischer, P. Kemper, M. Völker. *The ProC/B Toolset for Modelling and Analysis of Process Chains*. In: T. Field, P.G. Harrison, J. Bradley U. Harder: TOOLS 2002, 2324, 51–70. Springer, 2002.

[9] F. Bause, P. Buchholz, C. Tepper: *The ProC/B-approach: From Informal Descriptions to Formal Models.* ISoLA – 1st Int. Symposium on Leveraging Applications of Formal Method, 30th October – 2nd November, Paphos (Cyprus), 2004.

[10] J.L. Peterson: *Petri Net Theory and the Modelling of Systems*. Prentice Hall, 1981.

[11] A. Kuhn: *Prozessketten in der Logistik - Entwicklungstrends und Umsetzungsstrategien*. Verlag Praxiswissen, 1995.

[12] A. Kuhn. *Prozesskettenmanagement – Erfolgsbeispiele aus der Praxis*. Verlag Praxiswissen, 1999.

[13] C.A. Petri: *Kommunikation mit Automaten*. Bonn: Schriften des Rheinisch-Westfälischen Institutes für instrumentelle Mathematik Universität Bonn, 1962.

[14] F. Bause, P.S. Kritzinger. *Stochastic Petri Nets - An Introduction to the Theory*. Vieweg&Sohn, 2002. available at http://ls4-www.informatik.uni-dortmund.de/QM/MA/fb/spnbook2.html.

[15] P.H. Starke. *Analyse von Petri-Netz-Modellen*. B.G. Teubner, 1990.

[16] F. Bause, H. Beilner. *Intrinsic Problems in Simulation of Logistic Networks*. Simulation in Industry, 11th Europ. Simulation Symposium and Exhibition (ESS99), Erlangen, October 26-28, 1999, SCS Publishing House, pp. 193-198.

[17] E. Best, P.S. Thiagarajan. *Some classes of live and safe Petri nets. Concurrency and nets: advances in Petri nets*, Springer, 1987, pp. 71 – 94.

[18] A. Valmari. *Stubborn Sets for Reduced State Space Generation.* Proc. 10th Int. Conf. on Application and Theory of Petri Nets, Vol. 2, Bonn, 1989, pp. 1-22.

[19] F. Bause. *On Non-Ergodic Infinite-State Stochastic Petri Nets*. Proc. of PNPM, Urbana, 2003.

[20] F. Bause, P. Buchholz, J. Kriege, S. Vastag. *A Simulation Environment for Hierarchical Process Chains based on ONMeT++*. SIMULATION, The Society for Modelling and Simulation International, first published on June 29 as doi:10.1177/0037549709104236, 2009.

Corresponding author: Sebastian Vastag
Informatik IV, TU Dortmund, D-44221 Dortmund
sebastian.vastag@udo.edu