Towards the Generation of Distributed Test Cases Using Petri Nets

Stefan Heymer and Jens Grabowski

Institute for Telematics, Medical University of Lübeck, Ratzeburger Allee 160, D–23538 Lübeck, Germany
phone +49 451 500 3724, fax +49 451 500 3722
email {heymer,grabowsk}@itm.mu-luebeck.de

Abstract. Test case generation is a means to validate the implementation of a system *a posteriori* with respect to some given requirements imposed on the system. Current methods for the generation of test cases often rely on an interleaving model of the system, which does not fully cover the situation in reactive systems and systems with asynchronous communication. In this paper we present an approach towards the generation of distributed test cases, using Petri nets as the class of models for system, requirements and test case.

1 Motivation

Testing is an important part of system design. In most cases, it is used to check the conformance of an implementation to a system specification. For communication protocols, this type of testing is called *Protocol Conformance Testing* [4]. Conformance testing is based on a set of requirements, the so–called *test purposes*, each of which has to be checked during the procedure of testing. Test purposes are identified, specified and later on implemented (in form of test cases) by hand.

With the increasing importance of distributed systems, there is also an increased demand in distributed testing, and hence a demand for the generation of test cases for distributed testing. Yet, most methods for test generation use interleaving models of computation. These models are not faithful for distributed and communicating systems, as they do not take possibly concurrent executions of actions into account.

Using Petri nets, it is possible to give a finite model for a system, even if this system will show an infinite behaviour. Moreover, with special classes of nets like Timed Petri nets, Algebraic nets or high–level Petri nets, also the incorporation of time and data aspects is possible. There are two general views at the dynamic behaviour of such a net model: the *sequential semantics* of a net looks at the set of its *occurrence sequences*, while the *causal semantics* looks at the set of *partially ordered runs* or *processes* of the net.

In this paper, we provide a foundation for the generation of test cases for distributed systems based on a Petri net model of a system. The net model of the system will be *open* in the sense that the environment of the system will not be part of the model. As the systems under consideration are reactive, i.e. continually communicating with their environment, the test cases we are looking for will be (possibly distributed) models of that environment. We will focus on the flow of control, but we will give some hints at the incorporation of data and time into the process of test generation.

The remainder of this paper is organised in the following way: Section 2 gives some preliminaries, providing the definitions of Petri nets, processes and components. Section

3 will give an account on our approach, first formulating the problem of test generation as the finding the solution of an inequality, then showing how to arrive at such a solution by a process of approximation. We close this paper with some conclusions and directions for further work in Section 4.

2 Preliminaries

To generate test cases, we need to give a formal model for the system under test (SUT). As systems communicate with their environment, we will use a modified form of the components presented in [5]. Components are Petri nets with input and output places acting as an interface between the component and its environment. So, we first give some definition on Petri nets and a special variant of them we will be using, labelled Petri nets. As our approach to test case generation will be based on processes, we will give definitions for occurrence nets and processes. The main point of these preliminaries will be the section on components and labelled components, which will play a central role in our approach.

2.1 Petri nets

We first present the basic notions from Petri net theory. In this paper, we concentrate on the flow of control in systems, and thus use place/transition systems without capacities and arc-inscriptions and their processes [**?**] as basic formalism. We also restrict ourselves to *T–restricted* nets of *finite synchronisation* [**?**].

We start with the definition of nets, markings and system nets.

Definition 1. A triple $N = (P, T; F)$ is a *net*, if P and T are two disjoint sets, the elements of which are called *places* and *transitions*, and $F \subseteq (P \times T) \cup (T \times P)$, called the *flow relation*.

A *marking* M of a net N is a mapping $M : P \to \mathbb{N}$ such that $\sum_{p \in P} M(p) \in \mathbb{N}$.

A *system net* $\Sigma = (N, M_0)$ is a net N equipped with a marking M_0 of N. M_0 is called the *initial marking* of Σ.

As usual, elements of P, T and F are called *places*, *transitions* and *arcs* of the net, respectively. A place or a transition of a net N is called a *node* of N. Sometimes, we will identify the flow relation F of a net with its characteristic function.

Definition 2. Let $N = (P, T; F)$ be a net and $x \in P \cup T$ be a node of N. We define the *preset* $\bullet x$ of x to be

$$\bullet x = \{x' \in P \cup T \mid (x', x) \in F\}$$

and the *postset* $x\bullet$ of x to be

$$x\bullet = \{x' \in P \cup T \mid (x, x') \in F\}.$$

For a set of transitions $T' \subseteq T$ we define $\bullet T' = \bigcup_{t \in T'} \bullet t$ and $T'\bullet = \bigcup_{t \in T'} t\bullet$.

The *minimal* and *maximal* elements of N are defined by $Min\, N = \{x \in P \cup T \mid \bullet x = \emptyset\}$ and $Max\, N = \{x \in P \cup T \mid x\bullet = \emptyset\}$, respectively.

A net is *T–restricted*, if $Min\, N \cup Max\, N \subseteq P$. A net N is of *finite syncronisation*, if for each transition $t \in T$ the preset and postset is finite.

For the dynamics of nets, we have to define the enabling and occurrence of transitions.

Definition 3. A transition t is enabled under a marking M, if

$$\forall p \in P. F(p,t) \leq M(p).$$

If t is enabled under the marking M, it can *occur*, its occurrence leading to a new marking M' (denoted $M\,[t\rangle\, M'$), which is defined as for each place p as

$$M'(p) = M(p) - F(p,t) + F(t,p).$$

Enabling and occurrence of transitions can be generalised to sets of transitions. A set of transitions $T' \subseteq T$ is enabled under the marking M, if

$$\forall p \in P. \sum_{t \in T'} F(p,t) \leq M(p).$$

The occurrence of a *step* $T' \subseteq T$ enabled under a marking M then leads to the marking M' (denoted $M\,[T'\rangle\, M'$) defined for each place $p \in P$ by

$$M'(p) = M(p) - \sum_{t \in T'} F(p,t) + \sum_{t \in T'} F(t,p).$$

Having presented these basic notions, we know go on to give a definition of processes of Petri nets.

2.2 Occurrence Nets and Processes

A *process* of a system net is an *occurrence net* with a labelling function establishing the correspondence between nodes of the occurrence net and nodes of the system net. First, we give a definition for occurrence nets.

Definition 4. A net $K = (B, E; F)$ is an *occurrence net*, if the following conditions hold:

- F is acyclic, i.e. the (irreflexive) transitive closure of F is a partial order, which will be denoted $<_K$,
- for each $b \in B$ holds $|\bullet b| \leq 1$,
- for each $b \in B$ the set $\{b' \in B \mid b' <_K b\}$ is finite (N is *finitely preceeded*), and
- $^\circ K$ is finite.

The elements of B and E are called *conditions* and *events*, respectively.

Branching processes are "unfoldings" of net systems containing information about both concurrency and conflicts, and were introduced by Engelfriet in **??**. Before defining the notion of a process, we first give a definition for homomorphisms on nets.

Definition 5. Let $N_1 = (P_1, T_1; F_1)$ and $N_2 = (P_2, T_2; F_2)$ be two nets. A *homomorphism* from N_1 to N_2 is a mapping $h : P_1 \cup T_1 \to P_2 \cup T_2$ such that

- $h(P_1) \subseteq h(P_2)$ and $h(T_1) \subseteq h(T_2)$, and
- for every $t \in T_1$, the restriction of h to $\bullet t$ is a bijection between $\bullet t$ in N_1 and $\bullet h(t)$ in N_2, and similarly for $t\bullet$ and $h(t)\bullet$.

Thus, a homomorphism is a mapping that preserves the nature of nodes and the environment of transitions.

Definition 6. A *branching process* of a net system $\Sigma = (N, M_0)$ is a pair $\beta = (N', \rho)$ where $N' = (B, E; F)$ is an occurrence net, and ρ is a homomorphism from N' to N such that

1. the restriction of ρ to $Min\, N'$ is a bijection between $Min\, N'$ and $\{p \in P \mid M(p) > 0\}$,
2. for every $e_1, e_2 \in E$, if $\bullet e_1 = \bullet e_2$ and $\rho(e_1) = \rho(e_2)$ then $e_1 = e_2$.

It has been shown in **??** that a net system has an unique maximal branching process up to isomorphism. We call it the *unfolding* of the system. Let $\beta' = (N', \rho')$ and $\beta = (N, \rho)$ be two branching processes of a net system. Then β' is a *prefix* of β if N' is a subnet of N satisfying the following:

- If a condition belongs to N', then its input event in N also belongs to N', and
- if an event belongs to N', then its input and output conditions in N also belong to N', and furthermore
- ρ' is the restriction of ρ to N'.

2.3 Components

Yet to come...

3 Our Approach

After having given all necessary definitions, we are now able to present our approach. We will first sketch the idea behind it, and will then give an step by step account on the approximation process, leading from a system under test and a test purpose to a set of test processes.

3.1 The Idea

In the following, we give a short account of our approach to the generation of test cases from Petri nets. As was already mentioned in Section 2, we use open components as the models of systems under test. This is the case as we are interested in the interaction of the system with its environment.

For a SUT $\mathcal{S}$, we also give a so–called test purpose $\mathcal{P}$ in the form of a closed component, showing some behaviour we expect the SUT to show. This test purpose does not have to

contain a complete description of the behaviour the system has to show, but may detail only some "areas of interest". It is given as a component containing a Petri net process.

So what we are looking for is another component $\mathcal{T}$ for a test system, which composed with the SUT $\mathcal{S}$ should result in a closed component $\mathcal{T} \circ \mathcal{S}$. This closed component has to show at least the behaviour of the test purpose. But what does this mean?

The behaviour of a system net can be given by its set of branching processes. This description of behaviour is non–interleaving, it reflects concurrency found in the system under test $\mathcal{S}$ and possibly in the test system $\mathcal{T}$. That is, we give the semantics of components as its set of labelled open processes analogously to [3]. Moreover, for test generation we consider isomorphism classes of such processes, that is, classes of processes that are isomorphic with respect to net structure and labelling.

What we want to find is a set of processes that contain the behaviour required by the test purpose $\mathcal{P}$. To reach this goal, we use a method of approximation, starting out with a process containing just the places marked in the initial marking of the test purpose $\mathcal{P}$, which also defines the points of control and observation used to test the system. From this starting point, we add transitions and places to the processes in the current set which do not violate the test purpose, follow the behaviour of the system under test and enhance the behaviour of the test system.

This process of approximation will be terminated when a set of processes has been found that completely contain the behaviour of the test purpose. From these processes, we synthesize a so-called open branching process for the test system $\mathcal{T}$. This branching process will be the solution to the inequality

$$\mathcal{T} \circ \mathcal{S} \sqsubseteq \mathcal{P};$$

where $\mathcal{T}$ is the unknown, and the relation $\sqsubseteq$ between nets is defined suitably.

3.2 Preparing the Way

3.3 Approximating Test Cases

We will now show how distributed test cases can be generated in a process of approximation. Our idea is the following: We take the model $\mathcal{S}$ and unfold it step by step. That is, we successively generate prefixes $\mathcal{S}_i$ of its unique branching process. For each of these prefixes, we build a completion $\mathcal{T}_i$, such that $\mathcal{T}_i \circ \mathcal{S}_i$ is a closed component. After each step, we test whether the inequality

$$\mathcal{T}_i \circ \mathcal{S}_i \sqsubseteq \mathcal{P}$$

holds — if it does, we have found a test case for $\mathcal{P}$, if it does not, we proceed with the next step in the unfolding of $\mathcal{S}$.

For the unfolding of the component $\mathcal{S}$ we use an algorithm similar to the unfolding algorithm given in [**?**]. We first define the notion of a *possible extension* of a branching process with respect to a given system net.

Definition 7. Let $\beta = (N', \rho)$ be a branching process of a component $\mathcal{C} = (\Sigma, I, O)$ with $\Sigma = (N, M_0)$ and $N = (P, T; F)$. A *possible extension* of β is a tuple (t, B), where B is a conflict–free set of conditions of β or nodes b mapping to places in I, and t is a transition of Σ such that $\rho(B) = \bullet t$ and β contains no event e satisfying $\rho(e) = t$ and $\bullet e = B$. $PE(\beta)$ denotes the set of possible extensions of β.

This definition deviates slightly from the one given for possible extensions in [**?**]. Basically, we allow tokens to "magically" appear at input places. With this definition, we proceed to formulate the unfolding algorithm.

Definition 8. Let $\mathcal{C} = (\Sigma, I, O)$ be a component with $\Sigma = (N, M_0)$ and $N = (P, T; F)$. Let π_0 be the function projecting 2–tuples onto their first argument. We define the ith unfolding of $\mathcal{C}$, denoted $\mathcal{C}_i$, as $\mathcal{C}_i = (\Sigma_i, I_i, O_i)$ with $\Sigma_i = (N_i, M_i')$. The net of $\mathcal{C}_i$ is defined to be the net component of the branching process $\beta_i = (N_i, \rho_i)$, which is defined recursively in the following way:

- $\beta_0 = (N_0, \rho_0)$, where $N_0 = (\emptyset, P_0; \emptyset)$ with $P_0 = \{(p, 0) \in P \times \mathbb{N} \mid M_0(p) > 0\}$ and $\rho_0 = \pi_0|_{P_0}$
- $\beta_{i+1} = (N_{i+1}, \rho_{i+1})$, where $N_{i+1} = (P_{i+1}, T_{i+1}; F_{i+1})$ with

$$P_{i+1} = P_i \cup \bigcup_{(t_j, B_j) \in PE(\beta_i)} (\bullet t_j \times \{i+1\})$$

$$T_{i+1} = T_i \cup \bigcup_{(t_j, B_j) \in PE(\beta_i)} \{(t_j, i+1)\}$$

$$F_{i+1} = F_i \cup \bigcup_{(t_j, B_j) \in PE(\beta_i)} \{((b, i), (t_j, i+1)) \in P_{i+1} \times T_{i+1} \mid b \in \bullet t_j\}$$

$$\cup \bigcup_{(t_j, B_j) \in PE(\beta_i)} \{((t_j, i+1), (b, i+1)) \in P_{i+1} \times T_{i+1} \mid b \in t_j \bullet\}$$

  and $\rho_{i+1} = \pi_1|_{P_{i+1}}$

The initial marking M_i' of Σ_i is defined as $M_i'((p, 0)) = M_0(p)$ and $M_i'((p, j)) = 0$ for $j > 0$.

The interfaces I_i and O_i of the component $\mathcal{C}_i$ are defined as $I_i = (I \times \mathbb{N}) \cap P_i$ and $O_i = (O \times \mathbb{N}) \cap P_i$, respectively.

The last two definitions can be completely analogously given for labelled branching processes and unfoldings of labelled components.

So now, we are able to successively unfold the model of the system under test. These unfoldings again will be (labelled) open components, which we have to combine with a test component in order to arrive at a closed system. Again, we first give a definition for possible extensions of a test $\mathcal{T}$.

Definition 9. Let $\mathcal{S} = (\Sigma_\mathcal{S}, I_\mathcal{S}, O_\mathcal{S}, \lambda_\mathcal{S})$ be a system component, and let $\mathcal{T} = (\Sigma_\mathcal{T}, I_\mathcal{T}, O_\mathcal{T}, \lambda_\mathcal{T})$ be a test component. A *possible input extension* for $\mathcal{T}$ with respect to $\mathcal{S}$ is a tuple (P, T, F, I, λ) such that:

- $P = \{p_0, p_1, p_2\}$, where p_0 is a maximal place of $\Sigma_\mathcal{T}$, and p_1, p_2 are places not contained in $\Sigma_\mathcal{T}$,
- $T = \{t\}$, with t being a transition not contained in $\Sigma_\mathcal{T}$,
- $F = \{(p_0, t), (p_1, t), (t, p_2)\}$
- there exists a place p in $O_\mathcal{S}$ labelled $pco : m$, which is not in conflict with p_0,
- $\lambda_\mathcal{T}(p_0) = tc$ for some $tc \in TC$, and
- $\lambda : \begin{cases} t \mapsto pco?m@tc \\ p_1 \mapsto pco : m \\ p_2 \mapsto tc. \end{cases}$

The set of all possible input extensions of a test $\mathcal{T}$ with respect to a system $\mathcal{S}$ is denoted $PI(\mathcal{T}, \mathcal{S})$.

Possible output extensions of a test $\mathcal{T}$ with respect to a system $\mathcal{S}$ and the set $PO(\mathcal{T}, \mathcal{S})$ of all possible output extensions are defined analogously.

A possible (input or output) extension simply defines an additional transition for a test $\mathcal{T}$, together with its pre- and postset, the flow relation around it and the labelling of places and transition. The condition of the intended interface place of the system not being in conflict with the place the new transition will attach to stems from the fact that the nets of $\mathcal{S}$ and $\mathcal{T}$ are branching processes — the place to interface to in $\mathcal{S}$ could lie on a branch different from those containing the remaining nodes that the transition would be connected to in $\mathcal{T}$.

To generate a test fulfilling the given test purpose $\mathcal{P}$, generate a sequence of test components $(\mathcal{T}_i)_{i\in\mathbb{N}}$ based on the sequence of unfoldings $(\mathcal{S}_i)_{i\in\mathbb{N}}$ we derived above. Each of the tests $\mathcal{T}_i$ has to complement an unfolding $\mathcal{S}_i$ such that the composition $\mathcal{T}_i \circ \mathcal{S}_i$ is closed. Again, we define the sequence of test components recursively.

Definition 10. Let $(\mathcal{S}_i)_{i\in\mathbb{N}}$ be a sequence of unfoldings of a given system component $\mathcal{S}$ as defined in Definition **??**. Let $n = |\mathcal{TC}|$. We define a sequence of test components $(\mathcal{T}_i)_{i\in\mathbb{N}}$ recursively:

- $\mathcal{T}_0 = (\Sigma_0, I_0, O_0, \lambda_0)$ where $\Sigma_0 = (N_0, M_0)$ and $N_0 = (P_0, T_0; F_0)$, with
  - $P_0 = \{p_1, p_2, \ldots, p_n\}$ being "fresh" places,
  - $T_0 = F_0 = I_0 = O_0 = \emptyset$,
  - $M_0(p) = 1$ for $p \in P_0$, and
  - $\lambda : p_i \mapsto tc_i$ for $i \in \{1, 2, \ldots, n\}$.
- $\mathcal{T}_{i+1} = (\Sigma_{i+1}, I_{i+1}, O_{i+1}, \lambda_{i+1})$, where $\Sigma_{i+1} = (N_{i+1}, M_{i+1})$ and $N_{i+1} = (P_{i+1}, T_{i+1}; F_{i+1})$, where
  - $P_{i+1} = P_i \cup \bigcup_{(P',T',F',I',\lambda')\in PI(\mathcal{T}_i,\mathcal{S}_{i+1})} P' \cup \bigcup_{(P',T',F',O',\lambda')\in PO(\mathcal{T}_i,\mathcal{S}_{i+1})} P'$,
  - $T_{i+1} = T_i \cup \bigcup_{(P',T',F',I',\lambda')\in PI(\mathcal{T}_i,\mathcal{S}_{i+1})} T' \cup \bigcup_{(P',T',F',O',\lambda')\in PO(\mathcal{T}_i,\mathcal{S}_{i+1})} T'$,
  - $F_{i+1} = F_i \cup \bigcup_{(P',T',F',I',\lambda')\in PI(\mathcal{T}_i,\mathcal{S}_{i+1})} F' \cup \bigcup_{(P',T',F',O',\lambda')\in PO(\mathcal{T}_i,\mathcal{S}_{i+1})} F'$,
  - $I_{i+1} = I_i \cup \bigcup_{(P',T',F',I',\lambda')\in PI(\mathcal{T}_i,\mathcal{S}_{i+1})} I'$
  - $O_{i+1} = O_i \cup \bigcup_{(P',T',F',O',\lambda')\in PO(\mathcal{T}_i,\mathcal{S}_{i+1})} O'$,
  - $\lambda_{i+1} = \lambda_i \cup \bigcup_{(P',T',F',I',\lambda')\in PI(\mathcal{T}_i,\mathcal{S}_{i+1})} \lambda' \cup \bigcup_{(P',T',F',O',\lambda')\in PO(\mathcal{T}_i,\mathcal{S}_{i+1})} \lambda'$, and

- $M_{i+1}(p) = \begin{cases} M_i(p), \text{ if } p \in P_i \\ 0, \quad \text{otherwise.} \end{cases}$

So by now we have two sequences of components $(\mathcal{S}_i)_{i\in\mathbb{N}}$ and $(\mathcal{T}_i)_{i\in\mathbb{N}}$ as approximations for the behaviours of system under test and test system, respectively. But where do we terminate this process of approximation? As our ondition for termination, we stated above the inequality

$$\mathcal{T}_i \circ \mathcal{S}_i \sqsubseteq \mathcal{P}.$$

A suitable choice for the relation $\sqsubseteq$ is that $\mathcal{P}$ is embedded in the component $\mathcal{T}_i \circ \mathcal{S}_i$.

Definition 11. Let $\mathcal{C}_1 = (\Sigma_1, I_1, O_1, \lambda_1)$ and $\mathcal{C}_2 = (\Sigma_2, I_2, O_2, \lambda_2)$ be two components with $\Sigma_i = (N_i, M_i)$ and $N_i = (P_i, T_i; F_i)$ for $i \in \{1, 2\}$. Assuming that the transitive closures of F_1 and F_2 are acyclic, we say that $\mathcal{C}_1 \sqsubseteq \mathcal{C}_2$ if there exists a morphism $h : P_2 \cup T_2 \to P_1 \cup T_1$ such that:

- $h(P_2) \subseteq P_1$ and $h(T_2) \subseteq T_1$,
- $h(I_2) \subseteq I_1$ and $h(O_2) \subseteq O_1$,
- $\forall p \in P_2.M_2(p) = M_1(h(p)) \wedge \lambda_2(p) = \lambda_1(h(p))$, and
- $\forall n_1, n_2 \in P_2 \cup T_2.n_1 \leq_{N_2} n_2 \Rightarrow h(n_1) \leq_{N_1} h(n_2)$, where $\leq_{N_i}$ is the partial order .

Hence, when we have $\mathcal{T}_i \circ \mathcal{S}_i \sqsubseteq \mathcal{P}$, we know that the net of the test purposes' component $\mathcal{P}$ is embedded in the closed component $\mathcal{T}_i \circ \mathcal{S}_i$, and we are able to terminate the approximation. If the inequality does not hold, we proceed with the next level of unfolding.

But as soon as the inequality holds, we do not have that $\mathcal{T}_i$ is a test case, as it still can contain "dead branches" not leading to a state where the test purpose $\mathcal{P}$ has been successfully completed. We have to prune out these branches to arrive at a suitable test case.

Let h be a morphism proving $\mathcal{T}_i \circ \mathcal{S}_i \sqsubseteq \mathcal{P}$ with $\mathcal{P} = ((N_\mathcal{P}, M_\mathcal{P}), I_\mathcal{P}, O_\mathcal{P}, \lambda_\mathcal{P})$. Furthermore, let P' be the maximal set with respect to set inclusion for which $h(P') = Max\, N_\mathcal{P}$ holds. We define the test component $\mathcal{T}$ to be $\mathcal{T} = (\Sigma, I, O, \lambda)$ with $\Sigma = (N, M)$ and $N = (P, T; F)$, where $P = P_i \cap \downarrow P'$, $T = T_i \cap \downarrow P'$, $F = F_i \cap (\downarrow P' \times \downarrow P')$, $M = M_i|_{\downarrow P'}$, $I = I_i \cap \downarrow P'$, $O = O_i \cap \downarrow P'$ and $\lambda = \lambda_i|_{\downarrow P'}$.

By construction, the test component $\mathcal{T}$ exhibits the desired property.

4 Conclusions and Further Work

So far, we have provided an approach for the generation of distributed test cases from a system specification and a requirement to be checked for. As the class of models we used Petri nets for this. This decision opens up two different ways: On the one hand, we are able to provide an interleaving semantics for Petri nets, giving an alternative approach to the usual test case generation, on the other hand, we are able to use a non–interleaving semantics for Petri nets to get to something new, while our approach remains invariant. We have shown how to generate a test system by a process of successive approximations,

as we have already done in the case of an interleaving model in [1], and how to generate a test case from this system.

How will we proceed from this point on? Our current approach focusses just on the control flow of systems. But for practical test generation, also the communication of data values plays an important role. Hence, we will try to extend our approach to also cover data aspects of systems. For this, we will try to abstract from marking structure, token structure and flow structure, as is shown in the *Petri Net Cube* of [2]. We do not expect much difficulties with this, as up to now, our approach does only take the structure of the net into account.

Also, with the increasing importance of real–time systems and the need for (automated) test generation for such systems, an extension of our approach to such systems would be desirable. For this, we will try to use an Petri Net Cube extended with a further dimension of timing information, to be as flexible as possible.

References

1. volume 0. Springer–Verlag, 1997.
2. volume 0. Springer–Verlag, 1998.
3. H. Ehrig, A. Merten, and J. Padberg. How to Transfer Concepts of Abstract Data Types to Petri Nets? *EATCS Bulletin*, 1997(62):106–114.
4. ISO. Information Technology, Open Systems Interconnection, Conformance Testing Methodology and Framework. International Standard IS–9646. ISO, Geneve, 1991.
5. E. Kindler. A Compositional Partial Order Semantics for Petri Net Components. In P. Azema and G. Balbo, editors, *Application and Theory of Petri Nets 1997, Proceedings*, volume 1248 of *LNCS*. Springer–Verlag, 1997.