Visualizing State Spaces with Petri Nets

H.M.W. Verbeek, A.J. Pretorius, W.M.P. van der Aalst, and J.J. van Wijk

Technische Universiteit Eindhoven
PO Box 513, 5600 MB Eindhoven, The Netherlands
{h.m.w.verbeek, a.j.pretorius, w.m.p.v.d.aalst, j.j.v.wijk}@tue.nl

Abstract. Many analysis techniques are based on state spaces, e.g., based on some system model or system log, state spaces are generated that can be used to verify behavioral properties (e.g., absence of deadlocks). State spaces can be inspected automatically without any human interpretation. However, for a good understanding of the system's behavior the analyst needs to inspect and interpret the corresponding state space. Therefore, *state space visualization* is important. Unfortunately, this aspect has been neglected and today's analysis tools typically represent states spaces in such a way that they are difficult to interpret and inspection is only possible for toy examples. *Attribute-based visualization methods* address this issue by enabling users to study state spaces in terms of data they understand: the attributes associated with every state. These techniques can deal with many types of attributes. In this paper we propose an approach based on *Petri nets*. Using existing synthesis techniques based on *regions*, we automatically derive a Petri net from a given state space. We consider the places of this Petri net as new derived state attributes. Using these, we combine attribute-based visualizations of the state space with diagrams of the associated Petri net. This approach has been implemented and provides an innovative and versatile way to visualize state spaces.

1 Introduction

State spaces are popular for the representation and verification of complex systems [4]. System behavior is modeled as a number of *states* that evolve over time by following *transitions*. Transitions are "source-action-target" triplets where the execution of an *action* triggers a change of state. By analyzing state spaces more insights can be gained into the systems they describe.

In this paper, we assume the presence of state spaces that are obtained via *model-based state space generation* or through *process mining* [1]. Given a process model expressed in some language with formal semantics (e.g., Petri nets, process algebras, state charts, EPCs, UML-ADs, MSCs, BPEL, YAWL, etc.), it is possible to construct a state space (assuming it is finite). Often the operational semantics of these languages are given in terms of transition systems, making the state space generation trivial. It is also possible to extract state spaces from event logs using process mining. Process mining techniques build models by analyzing observed causalities, e.g., if some activity A is always followed by B in

the event log, then this is reflected in the discovered model. Some process mining techniques discover higher level models (e.g., Petri nets) that can be mapped onto a state space while other process mining techniques directly construct a state space [3].

State spaces describe system behavior at a low level of detail. A popular analysis approach is to specify and check requirements by inspecting the state space, e.g., model checking approaches [7]. For this approach to be successful, the premise is that all requirements are known. When this is not the case, the system cannot be verified.

Interactive visualization is another technique for studying state spaces. We argue that it offers three advantages:

1. By giving visual form to an abstract notion, communication among analysts and with other stakeholders is enhanced.
2. Users often do not have precise questions about the systems they study, they simply want to "get a feeling" for their behavior. Visualization allows them to start formulating hypotheses about system behavior.
3. Interactivity provides the user with a mechanism for analyzing particular features and for answering questions about state spaces and the behavior they describe.

Attribute-based visualization enables users to analyze state spaces in terms of attributes associated with every state. Users typically understand the meaning of this data and can use this as a starting point for gaining further insights. In this paper we investigate the possibility of automatically deriving attribute information for visualization purposes. To do so, we use existing synthesis techniques to generate a Petri net from a given state space. The places of this Petri net are considered as new derived state attributes.

The remainder of the paper is structured as follows. In Section 2 we provide a concise overview of state space visualization before introducing our approach in Section 3. In Section 4 we discuss how we use Petri-net theory to realize our approach. This is followed by a discussion of a proof of concept in Section 5 and a case study in Section 6. The challenges we faced while implementing our approach are discussed in Section 7. Finally, we conclude in Section 8.

2 State space visualization

Since state spaces are graphs, most existing visualization techniques focus on their topology. For this off-the-shelf graph drawing tools are often used [11]. However, results usually do not scale well and are not effective for highlighting interesting aspects of system behavior. For example, even for the relatively small state space visualized in Figure 1, the corresponding behavior is difficult to interpret by system analysts. Figure 1 nicely illustrates the limitations of the classical visualization approaches typically used by existing tools, i.e., poor scalability and inability to show structure.

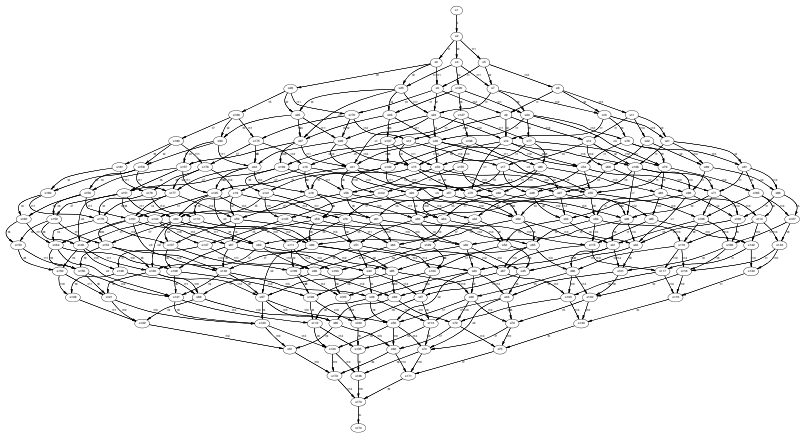


Fig. 1. State space visualization with off-the-shelf graph-drawing tools.

Fig. 2. State space visualization based on structural symmetry.

An effective technique for highlighting behavioral structure and symmetry is discussed in [10]. This approach generates a backbone on which states and transitions are positioned. The layers and branches of this backbone show phases of system behavior. However, once interesting aspects are identified it is difficult to study them further. Also, for systems where states are highly connected the backbone often contains no symmetrical branches (see Figure 2). Since such symmetries are generally taken as starting points for analysis, these cases are problematic.

Recent work [13, 14] suggests to visualize state spaces as a function of multivariate data associated with every state. States are considered as vectors of attribute values. One approach is to show low-dimensional projections of states onto 2D [13]. A related approach is based on interactive attribute-based clustering [14]. First, the user selects a subset of attributes to cluster on. All states are then partitioned based on the values they assume for these attributes. This results in a simplified aggregate state space that can then be analyzed. The ability to choose different attributes to cluster on serves as a powerful analysis technique.

Attribute-based visualization enables users to approach state spaces in terms of data they understand. Users often associate meaning with attributes that describe states. They know what aspect of the modeled system an attribute describes and what it means when it assumes different values. However, when there are no state attributes or when their meaning is not well understood, we need a strategy to generate meaningful attributes.

We propose an approach based on Petri nets. For a given state space, we automatically synthesize a Petri net from it that describes its behavior. This allows us to describe every state in terms of the places of this Petri net. These become our newly derived state attributes. Since the semantics of these are clear, we are able to employ attribute-based visualization techniques. This enables users to analyze state spaces in a meaningful way even in cases where there are no initial state attributes.

3 Approach

Figure 3 illustrates our approach. The behavior of systems can be captured in many ways. For instance, as an *event log*, as a formal *model* or as a *state space*. Typically, system behavior is not directly described as a state space. However, as already mentioned in the introduction, it is possible to generate state spaces from process models (i.e., model-based state space generation) and event logs (i.e., process mining [1, 3]). This is shown by the two arrows in the lower left and right of the figure. The arrow in the lower right shows that using model-based state space generation the behavior of a (finite) model can be captured as a state space. The arrow in the lower left shows that using process mining the behavior extracted from an event log can be represented as a state space [2]. Note that an event log provides execution sequences of a (possibly unknown) model. The event log does not show explicit states. However, there are various ways to construct a state representation for each state visited in the execution sequence, e.g., the prefix or postfix of the execution sequence under consideration. Similarly transitions can be distilled from the event log, resulting in a full state space. Figure 3 also shows that there is a relation between event logs and models, i.e., a model can be used to generate event logs with example behavior and based on an event log there may be process mining techniques to directly extract models, e.g., using the α-algorithm [3] a representative Petri net can be discovered based on an event log with example behavior. Since the focus is on state space visualization,

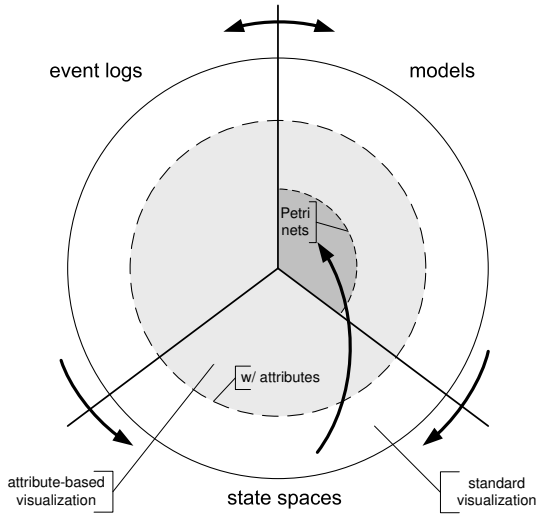


Fig. 3. The approach proposed in this paper.

we do not consider the double-headed arrow at the top and focus on the lower half of the diagram.

We make a distinction between event logs, models and state spaces that have *descriptive attributes* and those that do not (inner and outer sectors of Figure 3). For example, it is possible to model behavior simply in terms of transitions without providing any further information that describes the different states that a system can be in. Figure 1 shows a state space where nodes and arcs have labels but without any attributes associated to states. In some cases it is possible to attach attributes to states. For example, in a state space generated from a Petri net, the token count for each state can be seen as a state attribute. When a state space is generated using process mining techniques, the state may have state attributes referring to activities or documents recorded earlier.

It is far from trivial to generate state spaces that contain state attributes from event logs or models where this information is absent. Moreover, there may be an abundance of possible attributes making it is difficult to select the attributes relevant for the behavior. For example, a variety of data elements may be associated to a state, most of which do not influence the occurrence of events. Fortunately, as the upward pointing arrow in Figure 3 shows, there exist powerful synthesis techniques for deriving Petri nets from state spaces, e.g., the Theory or Regions [9, 12, 5]. An example of a synthesis tool is *Petrify* [6] which, given an arbitrary (finite) state space, is able to construct an equivalent Petri net. In this process Petri-net places are introduced that demarcate important milestones in the process. These places can be used as attributes when generating a state space from the synthesized Petri net. As a result, we can *transform a state space without attributes into a state space with attributes.*

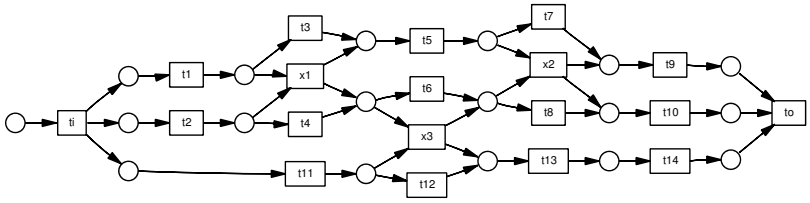


Fig. 4. Petri net synthesized from the state space in Figure 1.

4 Petri nets

As we outlined above, our aim is to derive and annotate a state space with attribute information. In order to use attribute-based visualization techniques, the semantics of these attributes should be understood by users. Petri nets [15] offer a well established visual formalism where the meaning of different components, such as places, is clear.

Consider the state space in Figure 1. Since it does not have any state attributes, we cannot employ attribute-based visualization techniques. When we perform synthesis, we derive a Petri net that is guaranteed to be bisimilar to this state space. That is, the behavior described by the Petri net is equivalent to that described by the state space [6]. Figure 4 shows a Petri net derived using the Theory of Regions [9, 12, 5].

A classical Petri net can represented as a triplet (P, T, F) where P is the set of places, T is the set of Petri net transitions[1], and $F \subseteq (P \times T) \cup (T \times P)$ the set of arcs. For the state of a Petri net only the set of places P is relevant, because the network structure of a Petri net does not change and only the distribution of tokens over places changes. A state, also referred to as *marking*, corresponds to a mapping from places to natural numbers. Any state s can be presented as $s \in P \rightarrow \{0, 1, 2, \ldots\}$, i.e., a state can be considered as a multiset, function, or vector. In the context of state spaces, we use places as attributes. In any state the value of each place attribute is known: $s(p)$ is the value of attribute $p \in P$ in state s.

A Petri net also comes with an unambiguous visualization. Places are represented by circles or ovals, transitions by squares or rectangles, and arcs by lines. Using existing layout algorithms, it is straightforward to generate a diagram for this. For example, the layout of the diagram in Figure 4 has been generated using *dot* [11].

Note that our approach, as illustrated in Figure 3, does not require starting with a state space. Any Petri net can also be handled as input, provided that its state space can be constructed within reasonable time. For a bounded Petri net,

[1] The transitions in a Petri net should not be confused with transitions in a state space, i.e., one Petri net transition may correspond to may transitions in the corresponding state space. For example, many transitions in Figure 1 refer to the Petri net transition $t1$ in Figure 4.

this state space is its reachability graph, which will be finite. The approach can also be extended for unbounded nets by using the coverability graph. In this case, $s \in P \rightarrow \{0, 1, 2, \ldots\} \cup \{\omega\}$ where $s(p) = \omega$ denotes that the number of tokens in p is unbounded. This can also be visualized in the Petri net representation. We also argue that our technique is applicable to other graphical modeling languages with some form of semantics, e.g., the various UML diagrams describing behavior. In the context of this paper, we use state spaces as starting point since these are more generic than Petri nets. As proof of concept, the next section discusses a possible implementation of this approach.

5 Proof of concept

We have implemented our approach using three existing tools: *Petrify*, *DiaGraphica*, and *ProM*.[2] Figure 5 shows the architecture of this implementation.

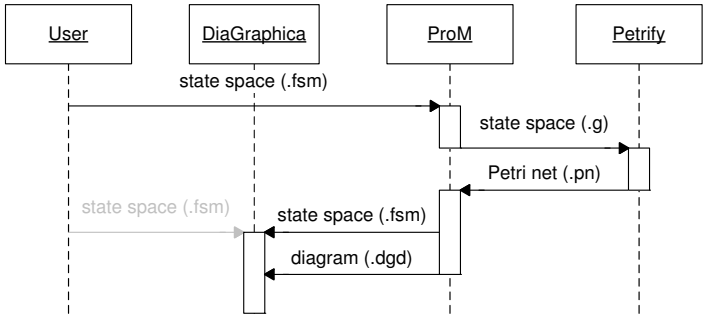


Fig. 5. Implementation architecture.

A state space in ".fsm" format can be loaded into *ProM*. This format is also supported by other tools such as *DiaGraphica* and the tool described in [10]. *ProM* can convert this into a state space suitable for *Petrify* and export it as a so-called ".g" file. Figure 5 does not show that *ProM* also provides other ways of constructing state spaces suitable for *Petrify*, e.g., using process mining. Section 6 will illustrate this by demonstrating that *ProM* can construct a space space based on the log of a workflow management system. In any case, using the export of *ProM* to *Petrify* and by using the synthesis capabilities of *Petrify*, a so-called ".pn" file is produced that can be loaded by *ProM* and represented as a Petri net. This Petri net can be used for many purposes (e.g., all kinds of Petri net based analysis techniques) including state space generation. The resulting state space can be exported to *DiaGraphica* using again the ".fsm"

[2] Note that the authors have been heavily involved in the development of *ProM* [8, 16] and *DiaGraphica* [13, 14].

format but now with attributes. Moreover, *ProM* can generate a visualization of the Petri net diagram usable by *DiaGraphica* using a so-called ".dgd" file. *Using the procedure shown in Figure 5 a state space without attributes can be converted into a state space with place attributes where each state is represented in terms of a synthesized Petri net.* It is important to note that Figure 5 shows one of many possible application scenarios. For example, as mentioned before, the state space may be generated by *ProM* using process mining techniques. Moreover, the resulting model does not need to be represented as a Petri net; as shown in [16] *ProM* allows for various mappings among a wide variety of process modeling languages (EPCs, YAWL, Petri nets, CPNs, etc.). As depicted by the grey arrow in Figure 5, any state space (in ".fsm" format, with or without attributes) can be visualized directly using *DiaGraphica*.

The remainder of this section describes the core ingredients of our approach shown in Figure 5.

5.1 Synthesis

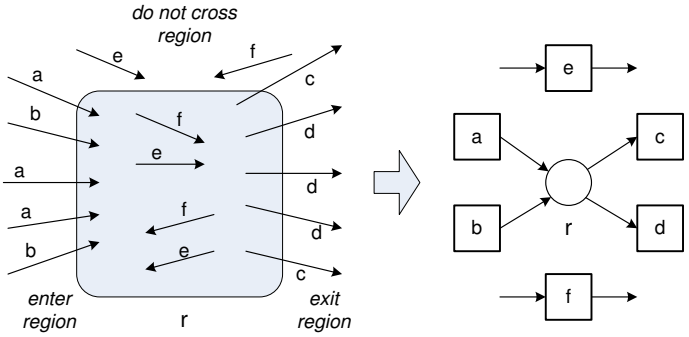
To derive a Petri net from a state space, we use the synthesis tool *Petrify* [6]. This tool is based on the Theory of Regions [9, 12, 5]. Using regions it is possible to synthesize a finite transition system (i.e., a state space) into a Petri net.

A (labeled) transition system is a tuple $TS = (S, E, T, s_i)$ where S is the set of states, E is the set of events, $T \subseteq S \times E \times S$ is the transition relation, and $s_i \in S$ is the initial state. Given a transition $TS = (S, E, T, s_i)$, a subset of states $S' \subseteq S$ is a *region* if for all events $e \in E$ one of the following properties holds:

- all transitions with event e *enter the region*, i.e., for all $s_1, s_2 \in S$ and $(s_1, e, s_2) \in T$: $s_1 \notin S'$ and $s_2 \in S'$,
- all transitions with event e *exit the region*, i.e., for all $s_1, s_2 \in S$ and $(s_1, e, s_2) \in T$: $s_1 \in S'$ and $s_2 \notin S'$, or
- all transitions with event e *do not "cross" the region*, i.e., for all $s_1, s_2 \in S$ and $(s_1, e, s_2) \in T$: $s_1, s_2 \in S'$ or $s_1, s_2 \notin S'$.

The basic idea of using regions is that each region S' corresponds to a place in the corresponding Petri net and that each event corresponds to a transition in the corresponding Petri net. Given a region all the events that *enter* the region are the transitions producing tokens for this place and all the events that *exit* the region are the transitions consuming tokens from this place. Figure 6 illustrates how regions can be used to "discover" suitable places.

In the original theory of regions many simplifying assumptions are made, e.g., elementary transitions systems are assumed [9] and in the resulting Petri net there is one transition for each event. Many transition systems do not satisfy such assumptions. Hence many refinements have been developed and implemented in tools like *Petrify* [5, 6]. As a result it is possible to synthesize a suitable Petri net for any transition system. Moreover, tools such as *Petrify* provide different settings to navigate between compactness and readability and one can specify



(a) State space with region r. (b) Petri net with place r.

Fig. 6. A region r referring to a set of states in the state space is mapped onto a place: a and b enter the region, c and d exit the region, and e and f do not cross the region.

desirable properties of the target model. For example, one can specify that the Petri net should be free-choice. For more information we refer to [5, 6].

With a state space as input *Petrify* derives a Petri net for which the reachability graph is bisimilar to the original state space. We already mentioned that the Petri net shown in Figure 4 can be synthesized from the state space depicted in Figure 1. This Petri net is indeed bisimilar to the state space. For the sake of completeness we mention that we used *Petrify* version 4.1 (www.lsi.upc.es/petrify/) with the following options: `-d2` (debug level 2), `-opt` (find the best result), `-p` (generate a pure Petri net), `-dead` (do not check for the existence of deadlock states), and `-ip` (show implicit places).

5.2 Visualization

DiaGraphica is a prototype for the interactive visual analysis of state spaces with attributes and can be downloaded from www.win.tue.nl/~apretori/diagraphica/. It builds on a previous work [14] and it primary purpose is to address the gap between the semantics that users associate with attributes that describe states and their visual representation. To do so, the user can define custom diagrams that reflect associated semantics. These diagrams are incorporated into a number of correlated visualizations.

Diagrams are composed of a number of shapes such as ellipses, rectangles and lines. Every shape has a number of Degrees Of Freedom (DOFs) such as position and color. It is possible to define a range of values for a DOF. Such a DOF is then parameterized by linking it with a state attribute. For an attribute-DOF pair the values for the DOF are calculated by considering the values assumed by the attribute.

In the context of this paper this translates to the following. Suppose we have a state space that has been annotated with attributes that describe the different

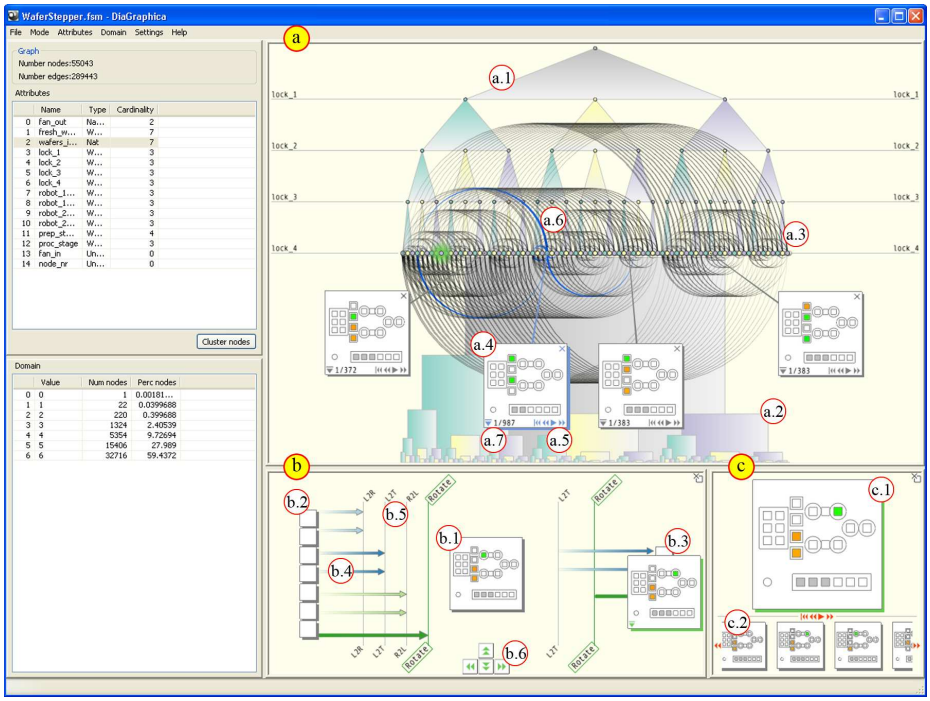


Fig. 7. DiaGraphica incorporates a number of correlated visualizations that use parameterized diagrams.

places in its associated Petri net. It is possible to represent this Petri net with a diagram composed out of a number of circles, squares and lines corresponding to its places, transitions and arcs. Now, we can parameterize all circles in this diagram by linking one of more of their DOFs with the attributes representing their corresponding places. For example, their colors can be parameterized such that circles assume a specific color only when the corresponding place is marked.

DiaGraphica has a file format for representing parameterized diagrams. The tool was originally developed with the aim of enabling users to edit and save custom diagrams. However, this facility also makes it possible to import diagrams regardless of where they originate from. Moreover, this allows us to import Petri nets generated with *Petrify* as diagrams (where *ProM* automatically generates the parameterized diagram including its layout). In this way we are able to realize the strategy introduced in Section 3.

Parameterized diagrams are used in a number of correlated visualizations. As starting point the user can perform attribute based clustering, as discussed in Section 2. The results are visualized in the cluster view (see Figure 7(a)). Here a node-link diagram, a bar tree and an arc diagram are used to represent the clustering hierarchy (cf. Figure 7(a.1)), the number of states in every cluster (cf. Figure 7(a.2)), and the aggregated state space (cf. Figure 7(a.3)) [14]. By clicking on clusters they are annotated with diagrams where the DOFs of shapes

are calculated as outlined above. A cluster can contain more than one state and it is possible to step through the associated diagrams and transitions. Transitions are visualized as arcs (Figure 7(a.5) and (a.6)). The direction of transitions is encoded by the orientation of the arcs which are interpreted clockwise.

The user can also load a diagram into the simulation view as shown in Figure 7(b). This visualization shows the "current" state as well as all incoming and outgoing states as diagrams (Figure 7(b.1)–(b.3)). This enables the user to explore a local neighborhood around an area of interest. Transitions are visualized by arrows and an overview of all action labels is provided (Figure 7(b.4) and (b.5)). The user can navigate through the state space by selecting any incoming our outgoing diagram, by using the keyboard or by clicking on navigation icons, cf. Figure 7(b.6). Consequently, this diagram slides toward the center and all incoming and outgoing diagrams are updated.

The inspection view enables the user to inspect interesting diagrams more closely and to temporarily store them (see Figure 7(c)). First, it serves as a magnifying glass as shown in Figure 7(c.1). When the user moves the mouse over a diagram in any view, it is shown here at a larger magnification. Second, the user can use the temporary storage facility. Users may, for instance, want to keep a history, store a number of diagrams from various locations in the state space to compare, or keep diagrams as seeds for further discussions with colleagues. These are visualized as a list of diagrams through which the user can scroll, cf. Figure 7(c.2).

Diagrams can be seamlessly moved between different views by clicking on an icon on the diagram (see Figure 7(a.7)). To maintain context, the current selection in the simulation or inspection view is highlighted in the clustering hierarchy.

5.3 Integration

To realize the approach introduced in Section 3 we need to integrate the results discussed in the previous two sections. We do this by using *ProM* [8]. More precisely, we use *ProM* to convert state spaces to the input format required by *Petrify*, to convert resulting Petri nets to a parameterized diagram suitable for *DiaGraphica*, and to construct attribute-annotated state spaces corresponding to these Petri nets.

The *ProM* framework was originally developed to extract process models from event logs for process mining purposes [1, 3]. However, in recent years the scope of the framework has become broader and it now includes features such as process verification, model transformation, model integration, social network analysis, conformance checking, and verification based on temporal logic. Moreover, the framework supports a wide variety of process models including state spaces, Petri nets, EPCs, YAWL, MSCs, and BPEL [16]. Since plug-ins can be seamlessly added to the framework and model interoperability is well supported [16], *ProM* provides an ideal platform for gluing tools such as *Petrify* and *DiaGraphica* together. Note that *ProM* is open-source software and can be downloaded from www.processmining.org.

5.4 Implementation

We have implemented two plug-ins in *ProM*: one for input and one for output. We used *ProM* version 4.0 as working base. The import plug-in takes a state space specified in a generic finite-state-machine (".fsm") format and converts it into a transition system model in *ProM*. Using an existing export plug-in (*TS to Petrify*) we then convert the transition system to the (".g") format that *Petrify* uses as input for generating a Petri net [2]. Our export plug-in exports this Petri net as a diagram (".dgd") and also produces an attribute-annotated state space (".fsm"). These latter two files serve as input for *DiaGraphica*.

When we apply our implementation to the state space in Figure 1 and load the resulting annotated state space and Petri net diagram into *DiaGraphica* we get results as shown in Figure 8. In the top part of Figure 8 we have chosen to cluster on the places `p1`, `p3` and `p12`. In the first level of the resulting clustering hierarchy, all states have been partitioned into two clusters: to the left are those states where `p1` is unmarked (Figure 8(a)) and to the right those that are marked (Figure 8(b)). When partitioning in marked and unmarked clusters different colors are used. The node shown in Figure 8(a) is green indicating that in the states in this cluster place `p1` is unmarked while the node shown in Figure 8(b) is yellow indicating that `p1` is marked. In the second level of the hierarchy these two clusters are again partitioned based on `p3`. We can see, for instance, that there are no states where both places `p1` and `p3` contain tokens: there is only one cluster branching from the cluster where `p1` contains a token (Figure 8(c)). Using the colors on the paths from top to bottom (i.e., green and yellow) it is easy to make this observation. Finally, every cluster in the second level of the hierarchy is again partitioned based on place `p12` being marked or not.

The above clustering results in an aggregated state space depicted by an arc diagram positioned on the leaves of the clustering hierarchy. Here the bundled transitions are represented by arcs that connect the leaf nodes (Figure 8(d)). Using these arcs we can see, for example, that there are only transitions possible between specific configurations of the places on which we have clustered. It is possible to view the action labels of the bundled transitions by rolling over the arcs with the mouse.

From the leaf nodes in the clustering in Figure 8 we see that there does not exist any state where all three places `p1`, `p3` and `p12` contain tokens. We can observe this by interpreting the clustering hierarchy as a binary decision tree. However, it is also possible to view the Petri nets associated with individual states contained within every one of these clusters (Figure 8(e)). As we mentioned in Section 2, this can be done by clicking on the clusters. Using this technique it is much more intuitive to interpret states as markings of the Petri net diagram.

The currently selected diagram has also been loaded into the simulation view. In this visualization it is presented as a marked Petri net in the center (Figure 8(f)). From this visualization, we learn that in this state the transitions `t1` and `t14` are enabled (Figure 8(g)). Also, we see that this state can be reached by firing either transition `t10` or `t13` (Figure 8(h)). Furthermore, we can see exactly what the predecessor and successor states are by considering the diagrams

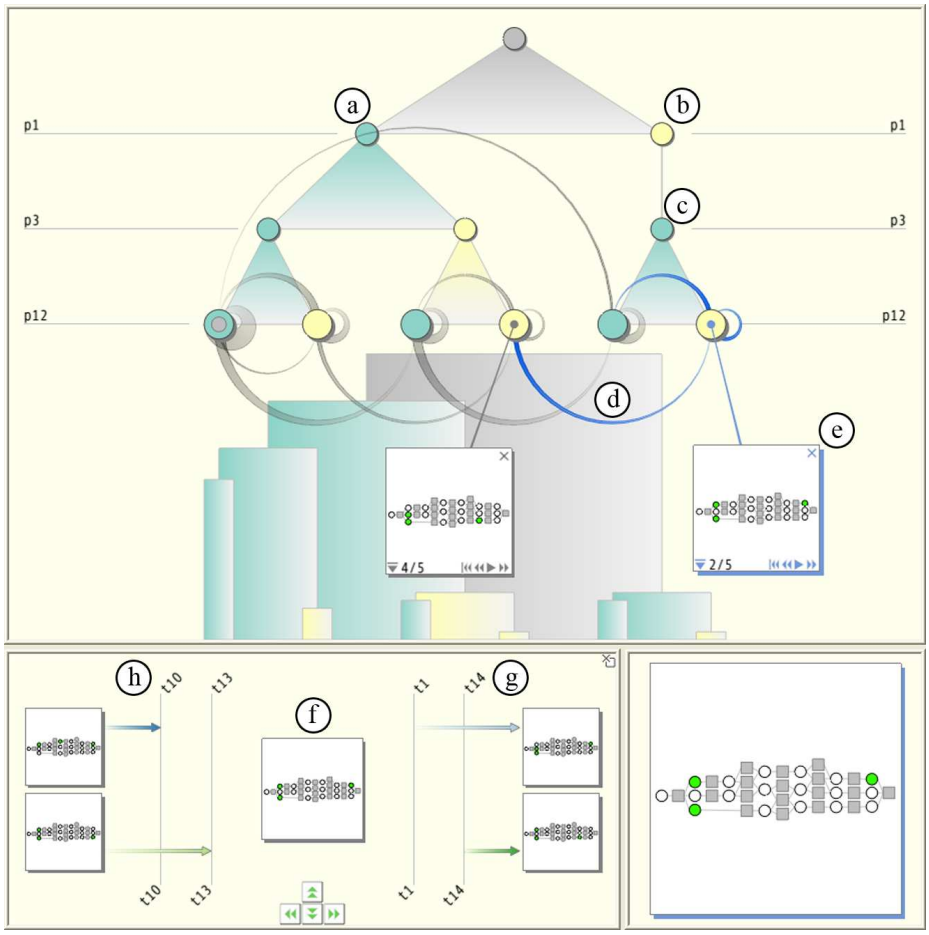


Fig. 8. The result of applying our approach to the state space in Figure 1. It is important to note that the Petri net representation and the corresponding visualizations are generated automatically from the unstructured state space depicted in Figure 1 without using any additional information.

toward the left and right of the current state. By rolling the mouse over these diagrams they are enlarged.

6 Case study

To illustrate how our approach can assist users we now present a small case study. Figure 9 illustrates the route we have taken in terms of the strategy introduced in Section 3. We started with an event log from the widely used workflow management system *Staffware*. The log contained 28 process instances (cases) and 448 events. Figure 10 shows a fragment of this log after importing it

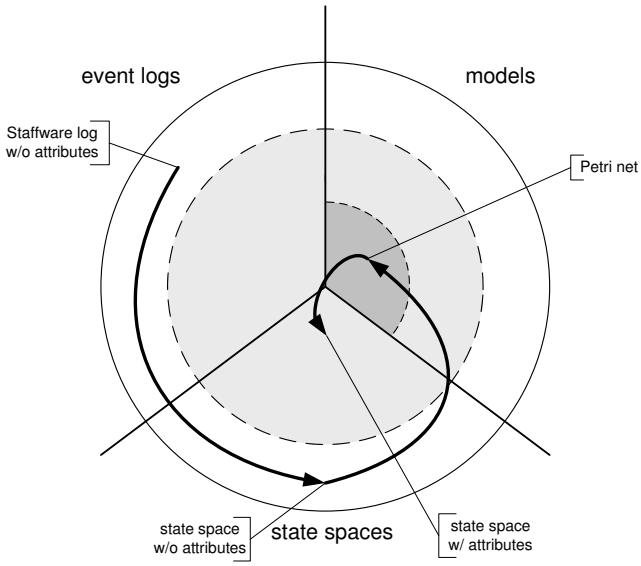


Fig. 9. The approach taken with the case study.

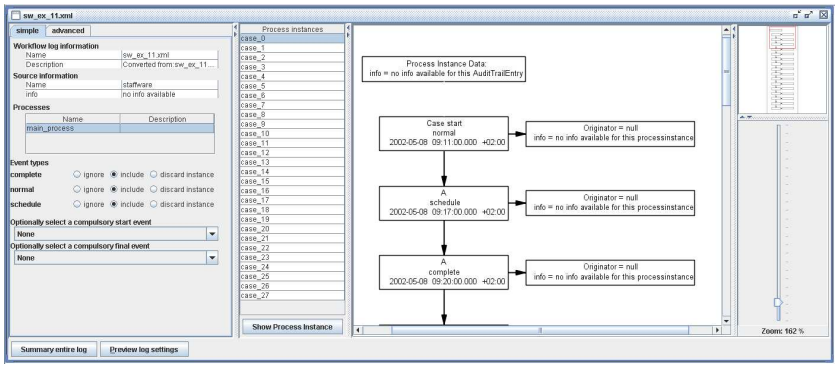


Fig. 10. A snapshot of *ProM* with the imported *Staffware* log.

into the *ProM* framework. Next, we generated a state space from the event log, using the *Transition System Generator* plug-in in *ProM*[3]. The resulting state space is shown in Figure 11. From the state space, we derived a Petri net using *Petrify*[4] (see Figure 12). Finally, all states in the state space were annotated with the places of this Petri net as attributes.

[3] In *ProM* the following options were selected: *Generate TS with Sets (Basic Algorithm)*, *Extend Strategy (Algorithm adds Additional Transitions)*, *The Log has Timestamps*, *Use IDs (numbers) as State Names*, and *Add Explicit End State* [2].

[4] This step took less than a second on a Pentium(R) 4 CPU 3.00 GHz computer with 3.5 GB RAM running Windows XP Professional Version 2002 SP 2.

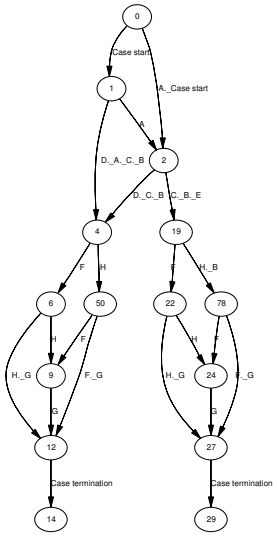


Fig. 11. The state space generated from the *Staffware* log.

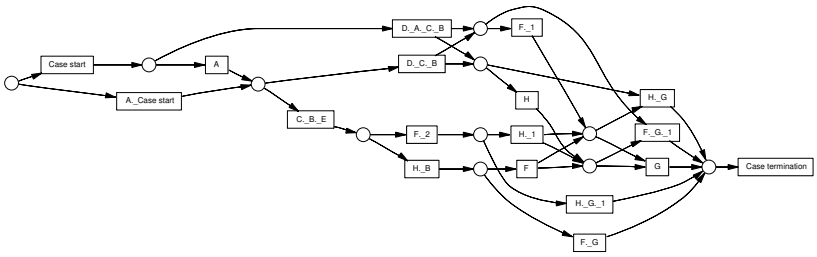


Fig. 12. The Petri net derived from the state space in Figure 11.

It is possible to take different perspectives on the state space by clustering based on different subsets of state attributes. For example, we were interested in studying it from the perspective of the places `p3`, `p6`, `p7` and `p10` (see Figure 13). When we clustered the state space based on these places we got the clustering hierarchy shown at the top of Figure 13.

Next, we clicking on the leaf clusters and considering the marked Petri nets corresponding to these. From these diagrams we learned that `p3`, `p6`, `p7` and `p10` contain either no tokens (Figure 13(a)) or exactly two of these places contain a token (Figure 13(b)–(f)). By considering the clustering hierarchy and these diagrams we also discover the following place invariant: `(p3 + p10) - (p6 + p7)`. That is, if `p3` or `p10` are marked, then either `p6` or `p7` is marked and vice versa.

By considering the arcs between the leaf nodes of the clustering hierarchy we learned that there is no *unrelated* behavior possible in the net while one of the places `p3`, `p6`, `p7` or `p10` is marked: every possible behavior changes at least

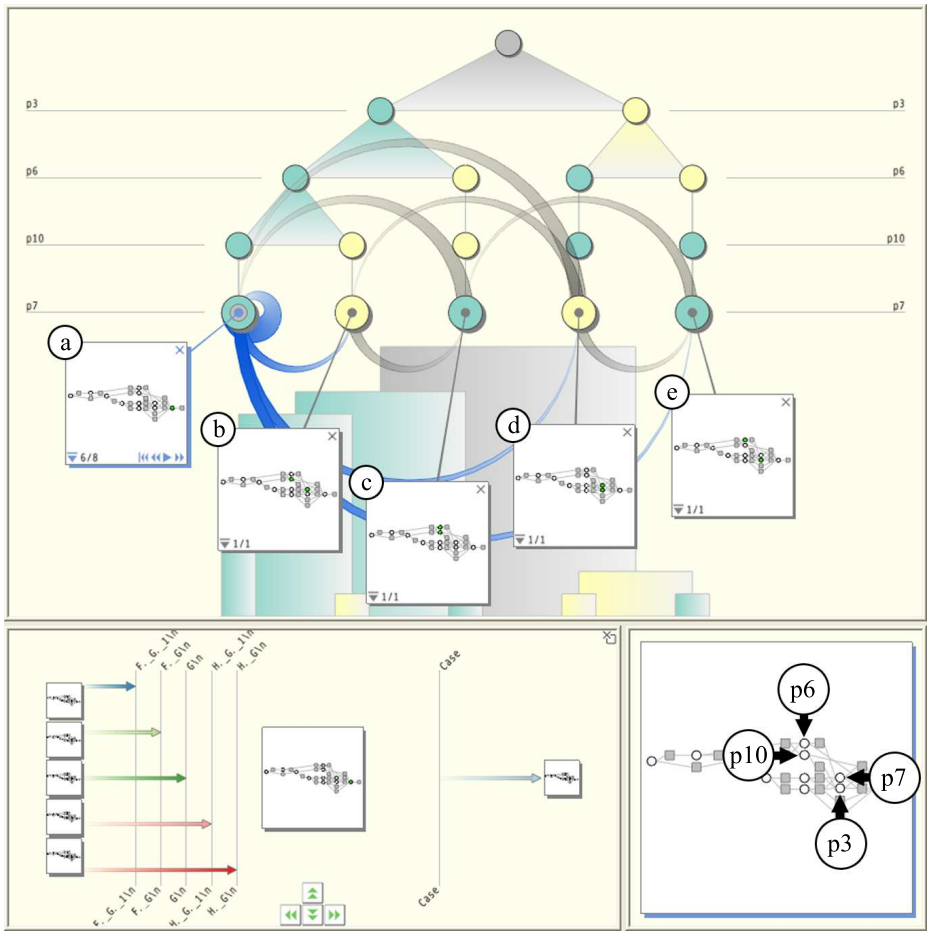


Fig. 13. Visualizing event log behavior using a state space and Petri net.

one of these four places. This holds because the only leaf node that contains a self-loop, represented by an arc looping back to its originating cluster, is the left-most cluster. However, as we noted above, this cluster contains all states where neither `p3`, `p6`, `p7` nor `p10` are marked. As an aside, by loading the current state into the simulation view at the bottom right, we saw that this state has five possible predecessors but only a single successor.

We also clustered on all places not selected in the previous clustering. This results in the clustering in Figure 14. In a sense this can be considered as the dual of the previous clustering. Here we found an interesting result. Note the diagonal line formed by the yellow clusters in Figure 14. Below these clusters the clustering tree does not branch any further. This means that only one of the places, apart from those we considered above (`p3`, `p6`, `p7` and `p10`), can be marked at the same time. Again, this observation is much more straightforward

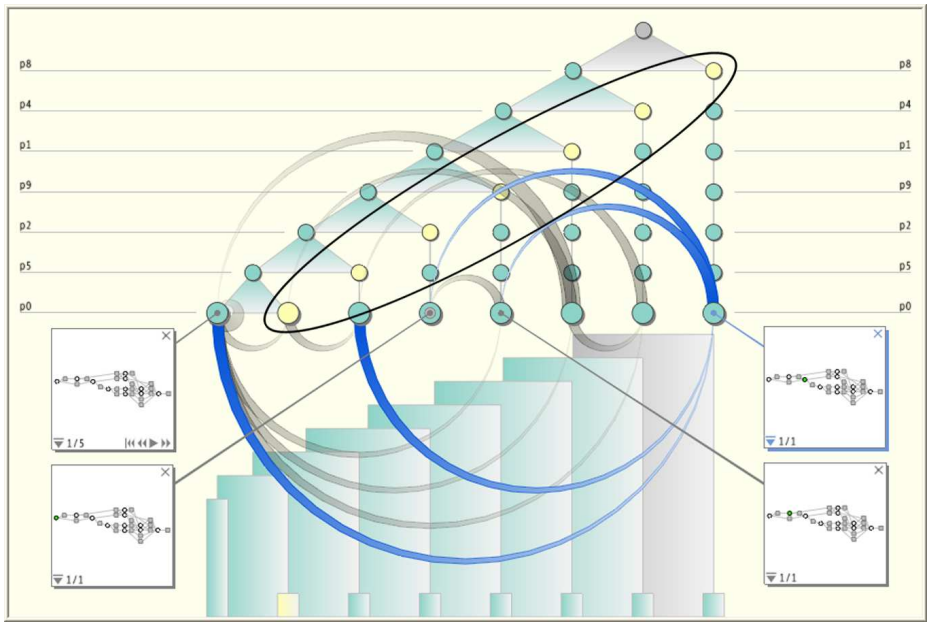


Fig. 14. Visualizing the clustering dual of Figure 13.

when we consider the diagrams representing these clusters. The leaf clusters below the diagonal contain no self loops. Similar to an earlier observation, this means that there is no unrelated behavior possible in the net while any one of the places we consider is marked.

7 Challenges

In practice we have found the synthesis of Petri nets from state spaces to be a bottle-neck. More specifically, we have found the space complexity to be an issue when attempting to derive Petri nets from state spaces using existing tools such as *Petrify*. *Petrify* is unable to synthesize Petri nets from large state spaces, especially if there is little "free" concurrency. If a and b can occur in parallel in state s_1, there are transitions $s_1 \xrightarrow{a} s_2$, $s_1 \xrightarrow{b} s_3$, $s_2 \xrightarrow{b} s_4$, and $s_3 \xrightarrow{a} s_4$ forming a so-called "diamond" in the state space. If there are fewer "diamonds" in the state space, this results in a poor confluence and Petri nets with many places. Figure 15 shows a Petri net obtained by synthesizing a relatively small state space consisting of 96 nodes and 192 edges. The resulting net consists of 50 places and more than 100 transitions (due to label splitting) and is not very usable and even less readable than the original state space. This is caused by the poor confluence of the state space and the resulting net nicely shows the limitations of applying regions to state spaces with little true concurrency.

Figure 15 illustrates that current synthesis techniques are not always suitable. If the system does not allow for a compact and intuitive representation in

Fig. 15. Suboptimal Petri net derived for a small state space.

terms of a labeled Petri net, it is probably *not useful to try and represent the system state in full detail*. Hence more abstract representations are needed when showing the individual states. The abstraction does not need to be a Petri net. However, even in the context of regions and Petri nets, there are several straightforward *abstraction mechanisms*. First of all, it is possible to split the sets of states and transitions into interesting and less interesting. For example, in the context of process mining states that are rarely visited and/or transitions that are rarely executed can be left out using abstraction or encapsulation. There may be other reasons for removing particular transitions, e.g., the analyst rates them as less interesting. Using abstraction (transitions are hidden, i.e., renamed to τ and removed while preserving branching bisimilarity) or encapsulation (paths containing particular transitions are blocked), the state space is effectively reduced. The reduced state space will be easier to inspect and allows for a simpler Petri net representation. Another approach is not to simplify the state space but to generate a model that serves as a simplified *over-approximation of the state space*. Consider for example Figure 15 where the complexity is mainly due to the non-trivial relations between places and transitions. If places are removed from this model, the resulting Petri net is still able to reproduce the original state space (but most likely also allows for more behavior). In terms of regions this corresponds to only including the "interesting" regions resulting in an over-approximation of the state space. Future research aims at selecting the right right abstractions and over-approximations.

8 Conclusion

In this paper we have presented an approach for state space visualization with Petri nets. Using existing techniques we derive Petri nets from state spaces in an automated fashion. The places of these Petri are considered as newly derived attributes that describe every state. Consequently, we append all states in the original state space with these attributes. This allows us to apply a visualization technique where attribute-based visualizations of state spaces are annotated with Petri net diagrams.

We have presented an implementation of our approach that provides the user with two representations that describe the same behavior: state spaces and Petri nets. These are integrated into a number of correlated visualizations. By presenting a case study, we have shown that the combination of state space visualization and Petri net diagrams assists users in visually analyzing system behavior.

We argue that the combination of the above two visual representations is more effective than any one of them in isolation. For example, using state space visualization it is possible to identify all states that have a specific marking for a subset of Petri net places. Using the Petri net representation the user can consider how other places are marked for this configuration. If we suppose that the user has identified an interesting marking of the Petri net, he or she can identify all its predecessor states, again by using a visualization of the state space. Once these are identified, they are easy to study by considering their Petri net markings.

In this paper, we have taken a first step toward state space visualization with automatically generated Petri nets. As we have shown in Section 5.4 and Section 6 the ability to combine both representations can lead to interesting discoveries. Our approach also illustrates the flexibility of parameterized diagrams to visualize state spaces. In particular, we are quite excited about the prospect of annotating visualizations of state spaces with other types of automatically generated diagrams. As indicated in Section 4, any diagram for which formalized behavioral semantics exist could potentially be used. Moreover, as indicated in Section 7, current synthesis techniques are not always suitable. Therefore, we are interested in automated abstraction techniques and over-approximations of the state space.

Acknowledgements

We are grateful to Jordi Cortadella for his kind support on issues related to the *Petrify* tool. Hannes Pretorius is supported by the Netherlands Organization for Scientific Research (NWO) under grant 612.065.410. The development of *ProM* is supported by EIT, NWO, and STW through various projects.

References

1. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow mining: A survey of issues and approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.
2. W.M.P. van der Aalst, V. Rubin, B.F. van Dongen, E. Kindler, and C.W. Günther. Process Mining: A Two-Step Approach using Transition Systems and Regions. Technical Report, BPMcenter.org, 2006.
3. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.
4. A. Arnold. *Finite Transition Systems.* Prentice Hall, 1994.
5. J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Synthesizing petri nets from state-based models. In *ICCAD '95: Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design*, pages 164–171, Washington, DC, USA, 1995. IEEE Computer Society.
6. J. Cortadella, M. Kishinvesky, L. Lavagno, and A. Yakovlev. Deriving petri nets from finite transition systems. *IEEE Transactions on Computers*, 47(8):859–882, August 1998.
7. D. Dams and R. Gerth. Abstract interpretation of reactive systems. *ACM Transactions on Programming Languages and Systems*, 19(2):253–291, 1997.
8. B.F. van Dongen, A.K.A. de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM framework: A new era in process mining tool support. In G. Ciardo and P. Darondeau, editors, *Applications and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer, Berlin, Germany, 2005.
9. A. Ehrenfeucht and G. Rozenberg. Partial (Set) 2-Structures - Part 1 and Part 2. *Acta Informatica*, 27(4):315–368, 1989.
10. F. van Ham, H. van de Wetering, and J.J. van Wijk. Interactive visualization of state transition systems. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):319–329, 2002.
11. E.R. Gansner, E. Koutsofios, S.C. North, and K.-P. Vo. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*, 19(3):214–230, 1993.
12. M. Nielsen, G. Rozenberg, and P. S. Thiagarajan. Elementary transition systems. In *Selected papers of the Second Workshop on Concurrency and compositionality*, pages 3–33, Essex, UK, 1992. Elsevier Science Publishers Ltd.
13. A.J. Pretorius and J.J. van Wijk. Multidimensional visualization of transition systems. In *Proceedings of the Ninth International Conference on Information Visualization (IV05)*, pages 323–328, 2005.
14. A.J. Pretorius and J.J. van Wijk. Visual analysis of multivariate state transition graphs. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):685–692, 2006.
15. W. Reisig and G. Rozenberg, editors. *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science. Advances in Petri Nets.* Springer, Berlin, Germany, 1998.
16. H.M.W. Verbeek, B.F. van Dongen, J. Mendling, and W.M.P. van der Aalst. Interoperability in the ProM framework. In T. Latour and M. Petit, editors, *Proceedings of the CAiSE'06 Workshops and Doctoral Consortium*, pages 619–630, Luxembourg, June 2006. Presses Universitaires de Namur.