IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS, VOL. 12, NO. 5, SEPTEMBER/OCTOBER 2006

Visual Analysis of Multivariate State Transition Graphs

A. Johannes Pretorius, and Jarke J. van Wijk, *Member, IEEE*

Abstract—We present a new approach for the visual analysis of state transition graphs. We deal with multivariate graphs where a number of attributes are associated with every node. Our method provides an interactive attribute-based clustering facility. Clustering results in metric, hierarchical and relational data, represented in a single visualization. To visualize hierarchically structured quantitative data, we introduce a novel technique: the bar tree. We combine this with a node-link diagram to visualize the hierarchy and an arc diagram to visualize relational data. Our method enables the user to gain significant insight into large state transition graphs containing tens of thousands of nodes. We illustrate the effectiveness of our approach by applying it to a real-world use case. The graph we consider models the behavior of an industrial wafer stepper and contains 55 043 nodes and 289 443 edges.

Index Terms—Graph visualization, multivariate visualization, interactive clustering, state spaces, transition systems, finite state machines.

1 Introduction

State transition systems form the semantic framework underlying contemporary specification and programming languages [1]. As a result, many analysis techniques for computer-based systems involve the translation of behavioral specifications or programs into transition systems [2, 3].

A transition system consists of a set of states and a set of transitions. Transitions are source-action-target triples. The execution of an action triggers a change of state. The corresponding state transition graph is defined as $G = (V, E)$ where a node $s \in V$ represents a state and a directed edge $t = (s, s') \in E$ represents a transition from state s (source) to state s' (target). Such graphs are formalisms that describe the behavior of systems whose states evolve over time.

Conceptually, transition graphs are simple structures. However, for modern computer-based systems they tend to be large, often containing tens of thousands of nodes, or more. This increases their complexity and hinders analysis of the systems they model. For example, it is hard to tell if, when and why undesirable states are reached.

There are two popular approaches for addressing the complexity of large transition graphs. The traditional tactic is to derive and analyze an abstracted, much smaller variant (typically with fewer than 100 nodes). Although easier to study, these are incomplete static descriptions. Potentially interesting behavior is lost. The second approach is to formulate and check requirements formally (using model checking, for instance [2]). This assumes that all requirements are known in advance. If this is not the case, the system cannot be verified.

Interactive visualization is a third technique for studying large state transition graphs. We argue that the advantage is threefold:

1. We have observed that by giving visual form to an abstract notion, communication between analysts themselves and between analysts and other stakeholders is substantially enhanced.

2. Users often do not have precise questions regarding the system at hand, they simply want to "get a feel" for it. Visualization allows them to start formulating hypotheses about system behavior [15].

3. Interactivity provides the user with a mechanism for analyzing particular features and for answering particular questions about transition graphs and the behavior they describe [12].

• *The authors are with the Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, PO Box 513, 5600 MB Eindhoven, The Netherlands.*
E-mail: a.j.pretorius@tue.nl, vanwijk@win.tue.nl

Manuscript received 31 March 2006; accepted 1 August 2006; posted online 6 November 2006.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

This paper introduces a new method for the interactive visualization of state transition graphs. In Section 2 we discuss related research before introducing our approach in Section 3. To illustrate its merit, we discuss a real-world use case in Section 4. Finally, in Section 5 we conclude by providing a summary of our results and by identifying opportunities for future research.

2 Related work

Most attempts at visualizing state transition graphs use off-the-shelf graph drawing tools [4]. Results do not scale well and are not effective for highlighting interesting aspects of system behavior. A few tools specifically for the visualization of transition graphs have been developed [6, 8]. Since user interaction is rather limited, they do not readily allow for interactive analysis. We know of two tools that cater for richer user interaction: FSMView [15] and StateVis [12].

FSMView was conceived to show global structural symmetries in transition graphs. Pre-processing generates a structural backbone on which nodes and edges are positioned. Ranking and clustering result in layers and branches from which phases of system behavior are discernable. However, once interesting aspects have been identified, FSMView offers limited facilities for analyzing these further. Also, for systems that contain little parallelism, or where states are highly connected, the backbone often contains no symmetrical branches. Since such symmetries are generally taken as departure points for analysis, these cases are problematic.

StateVis approaches the visualization of transition graphs from a multidimensional perspective. Every node consists of a vector of data values. These are the union of all data variables in the original behavioral specification, the so-called *state variables* [3]. In combination with a visual overview of the values these variables assume, StateVis offers different methods for projecting the (high-dimensional) nodes to 2D. It is possible to answer questions related to state variables, but analyzing correlations between variables and behavior requires some effort. Although suggestive behavioral patterns are often shown, it is difficult to gain a full understanding of these.

3 Approach

State attributes generalize the notion of state variables (see Section 2) and also allow for derived attributes to be associated with nodes. Examples include graph-structural properties (fan-in and fan-out, for example) and equivalence classes (a concept from operational semantics [1]). A crucial observation is that users associate precise meaning with state attributes. They know what aspect of the modeled system an attribute describes and what it means when it assumes different values. This observation is based on users at our institution and at *CWI*, the Dutch Institute for Mathematics and Computer Science.

Formally, with a state transition graph $G = (V, E)$ we associate n attribute:type pairs, $[a_1 : \tau_1, ..., a_n : \tau_n]$. $D_i = \{d_{i,1}, ..., d_{i,k_i}\}$ is the domain

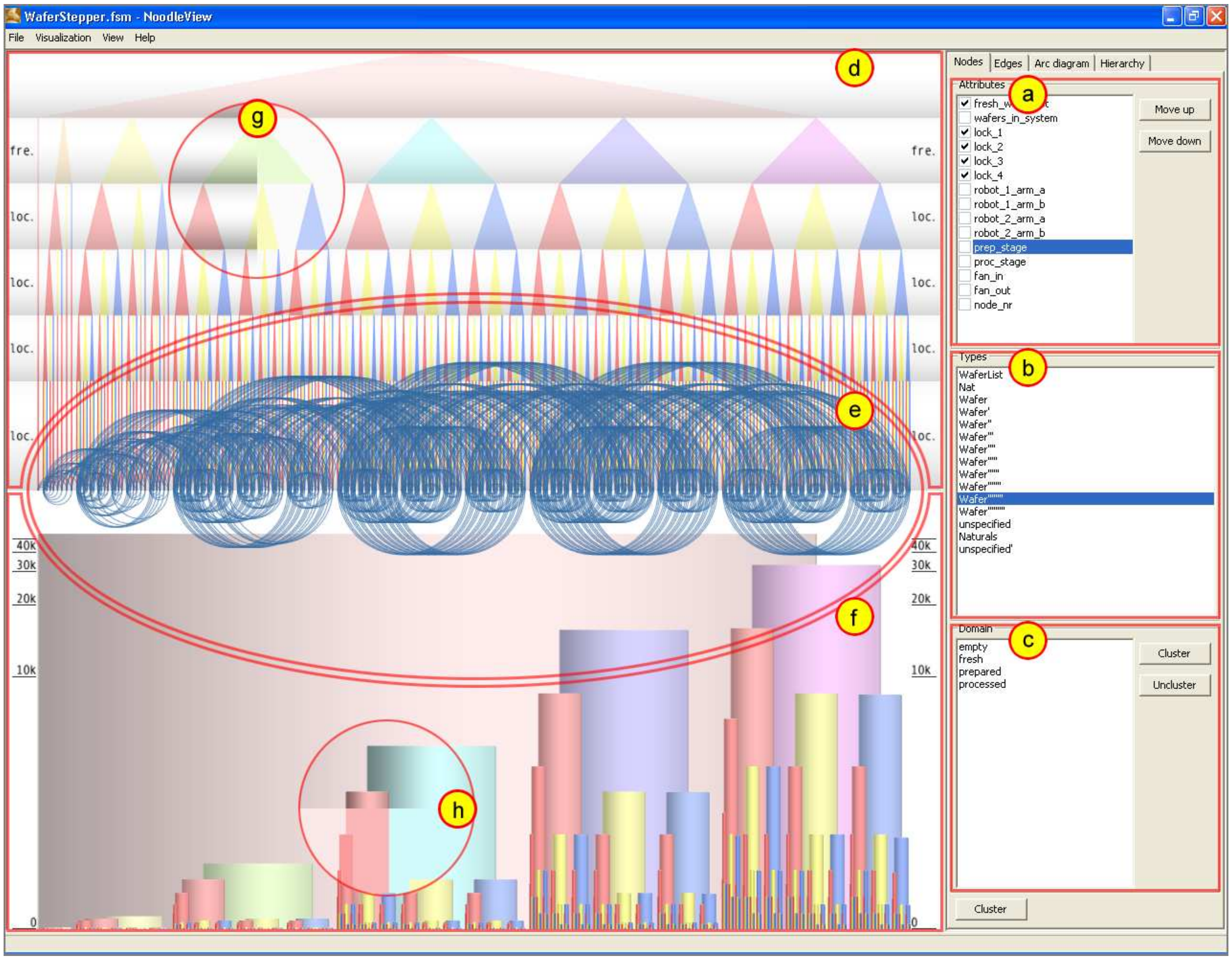

Fig. 1. Visualization of multivariate state transition graphs. The user can (a) select a subset of state attributes and (b) view their associated types. (c) Every type also has a domain of values. Attribute-based clustering results in three types of data: (d) a clustering hierarchy, visualized as a node-link diagram, (e) an abstract graph, visualized as an arc diagram and (f) metric data, visualized as a bar tree. All data are represented in a single visualization, enabling the identification of correlations. Subtle cushioning is used (g) to clearly distinguish levels in the hierarchy and (h) to give a layered effect illustrating the hierarchical nature of the bar tree.

of type τ_i. Every node $s \in V$ is defined as an n-dimensional (multivariate) tuple of attribute values: $s = [s_1, ..., s_n]$ with $s_i \in D_i$. We note that for transition graphs, the domain D_i of attribute a_i is usually discrete with a small cardinality (typically, $1 \leq k_i \leq 30$). It could be argued that these data features limit the generality of our approach. However, for larger or continuous domains, this can be addressed by partitioning the domain into a smaller number of discrete clusters (a practice known as *binning*). We therefore argue that our results are also applicable in the wider context of multivariate graph visualization.

Within our current scope, we exploit the multivariate nature of state transition graphs in order to:

1. Enable the user to reduce the complexity of transition graphs in a semantically rich way.

2. Have the reduction technique itself serve as a powerful mechanism or tool with which to conduct analysis.

The above strategy is supported by providing the user with a visual representation of the reduction results. To do so, we integrate the representation of three different types of data (hierarchical, metric and relational) in a single visualization. The objective is to enable the user to perceive and analyze correlations between these different elements. Below, we explain how our visualization is built by integrating different visualization techniques. To validate our approach, we have implemented it in a prototype to which we refer here and in subsequent sections.

3.1 Interactive hierarchical clustering

Provided we have a state transition graph, our prototype enables the user to select a subset of attributes (see Figure 1 (a)). The order of the attributes in this subset can be adjusted. The user is also given an overview of the associated attribute types and their value domains (see Figure 1 (b) and (c)).

Once a subset of attributes has been selected, the next step is to cluster the transition graph based on this selection. The complete set of nodes is considered as the root of a clustering hierarchy. The set is then partitioned based on the different values that nodes assume for the first attribute. This gives rise to a number of child nodes branching from the root (see Figure 2 (a)). Each of the resulting partitions is sub-partitioned in a similar fashion, based on the second attribute. This results in a third level of nodes in the clustering hierarchy (see Figure 2 (b)). Continuing in this fashion, a tree consisting of $p+1$

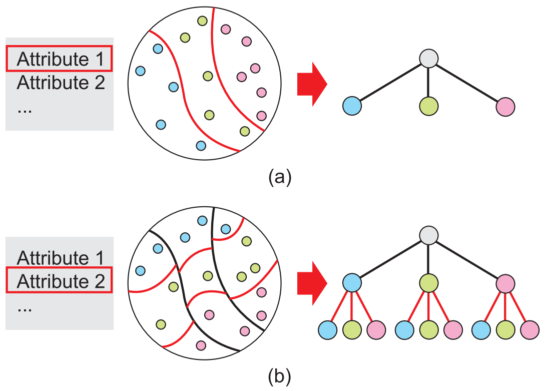

Fig. 2. Hierarchical clustering.

levels is generated for a subset of p attributes. Since the cardinalities of our domains are known, clustering has a complexity of $\mathcal{O}(p \cdot q)$, with p as above and q the number of nodes. This is efficient enough for clustering to be used as interaction mechanism, as we illustrate in Section 4.

Attribute-based clustering results in a tree structure that represents a recursive sub-partitioning of the original set of nodes. It also produces a less complex abstraction of the original transition graph which we discuss further in Section 3.3. We use the clustering hierarchy as the starting point of our visualization and show it as a node-link diagram in the top half of our visualization (see Figure 1 (d)). Parent-child relations are represented by triangles. We use subtle cushioning to differentiate levels better (see Figure 1 (g)) [17]. The different values assumed in a particular level are coded with distinct colors. The objective is not to facilitate exact matches between values and colors, but to enable the identification of recurring patterns. Feedback from users suggests that our straight-forward color mapping scheme suffices for this. In our prototype, we experimented with three other graphical representations for the hierarchy (see Figure 3). Users were unanimously in favor of triangles, citing that this strikes the best balance between effectively representing the hierarchy and highlighting repeated patterns in terms of the color-coded values that different attributes assume.

As an additional feature, the user is also enabled to cluster values of domains, thereby reducing their cardinality. This corresponds to the notion of *data abstraction*, as used by the formal methods community [2]. As we show in Section 4, this results in reduced complexity and is a powerful analysis technique in itself.

3.2 Metric data - bar trees

It is possible to compute quantitative values for every node in the clustering hierarchy. Such metrics are often cumulative: the values of non-leaf nodes can be obtained by summing the values of their children. As we show in Section 4, an elementary and useful example is the number of nodes contained in the corresponding cluster. To visualize such metrics we developed a new technique that uses nesting and layering to represent the hierarchy with which metrics are associated. We refer to this visualization as a bar tree.

Bar trees are constructed as follows (see Figure 4 (a) – (c)). Starting with the leaf nodes and working our way back to the root, we encode the metrics associated with nodes in every successive level as a bar chart. The bar chart associated with level i is positioned behind that of level $i+1$. For all levels, except the leaves, the widths of individual bars in level i are calculated to span the bars corresponding to its child nodes in level $i+1$. This results in a layered nesting that reflects the clustering hierarchy.

As with many other visualization techniques for hierarchical metric data (for example, [14, 16]) the aim is not to provide the user with a depiction from which exact values can be directly deduced. Rather, bar trees allow for the identification of patterns at different levels of the hierarchy. (In our prototype, exact values can be viewed by selecting the bars.) We position the bar tree directly below the clustering hierarchy in the bottom half of our visualization (see Figure 1 (f)). This further emphasizes the relationship between the bar tree and the clustering hierarchy. We color bars the same as matching nodes in the hierarchy and enable users to show or hide layers corresponding to different levels of this hierarchy. To amplify the layered effect, we use cushioning (see Figure 1 (h)).

Since the emphasis is on identifying patterns, particularly for lower levels of the hierarchy, a logarithmic distribution can be used on the y-axis. In Figure 5 (b), where this is done, much more detail of the lower levels of the bar tree is visible compared to Figure 5 (c), where the distribution is linear. In our prototype, the bar tree encodes cluster size. Users have indicated that the combination of selecting which levels to visualize in the bar tree and having control over the distribution function is an effective way to learn more about the partitioning of nodes resulting from hierarchical clustering. We experimented with fully nested bar trees, but users indicated that the hierarchy is not as clearly discernable (for instance, compare Figures 5 (a) and (b)).

Fully nested bar trees bear some resemblance to the "sum rendering method" proposed by Mihalisin et al. [9]. In general, the two methods differ in terms of their appearance, application and aim, however. As mentioned above, we visually emphasise the layered nature of the bar tree using depth cues [17]. Mihalisin et al. use their technique to conduct a visual statistical analysis of multivariate data items. Users

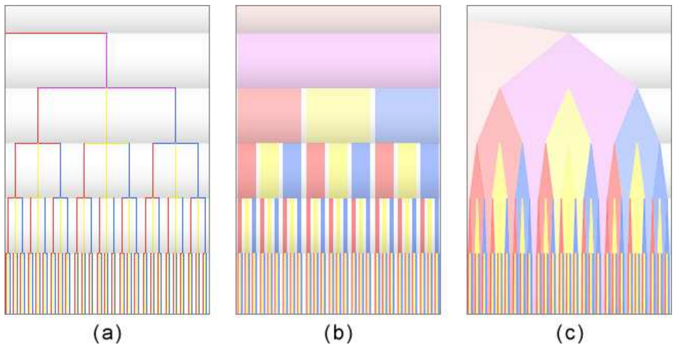

Fig. 3. Alternative graphical representations of the clustering hierarchy. (a) Dendrogram, (b) icicle plot and (c) buttresses.

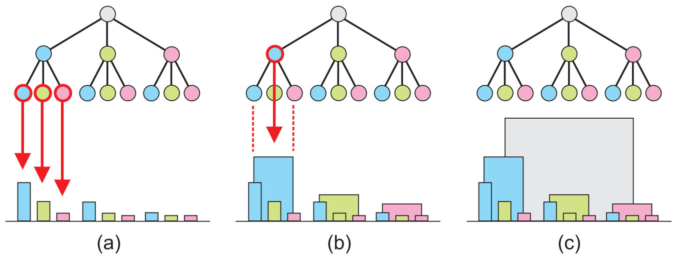

Fig. 4. Bar trees.

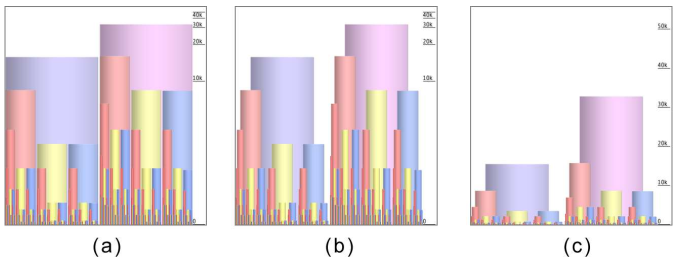

Fig. 5. Bar tree variations. (a) Fully nested, (b) logarithmic distribution and (c) linear distribution.

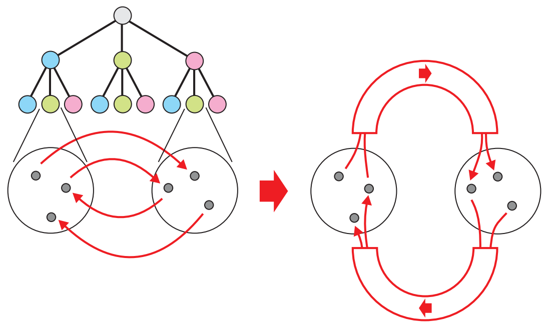
Fig. 6. Arc diagram.

are enabled to interactively permute the order of the variables in order to impose a hierarchy on the data. The actual values assumed for the different variables are then visually depicted. In terms of utility, this more closely resembles our hierarchical clustering facility. Finally, we are interested in representing summary information of an existing hierarchy and not the actual data dimensions themselves.

3.3 Abstract graph

The third and final element of our visualization represents the abstracted, less complex, transition graph that results from hierarchical clustering. It can be argued that this is the most important part of our visualization: an encoding of the behavior of the system being studied.

The leaves of the clustering hierarchy contain collections of nodes that are interconnected by a number of directed edges. It is important that the user is enabled to identify repeated behavioral patterns and symmetries represented by these edges (or by their absence). We capitalize on the fact that our leaf nodes are positioned on a horizontal line by visualizing behavior using an arc diagram positioned in the center of our visualization (see Figure 1 (e)). It has been shown that arc diagrams are well suited for identifying repeated patterns in relational data [18, 7].

We proceed as follows. Firstly, we bundle co-directional transitions. These bundles are represented by semi-circular arcs that span from the source to the target cluster (see Figure 6). The thickness of an arc is proportional to the number of bundled edges. Additionally, we encode

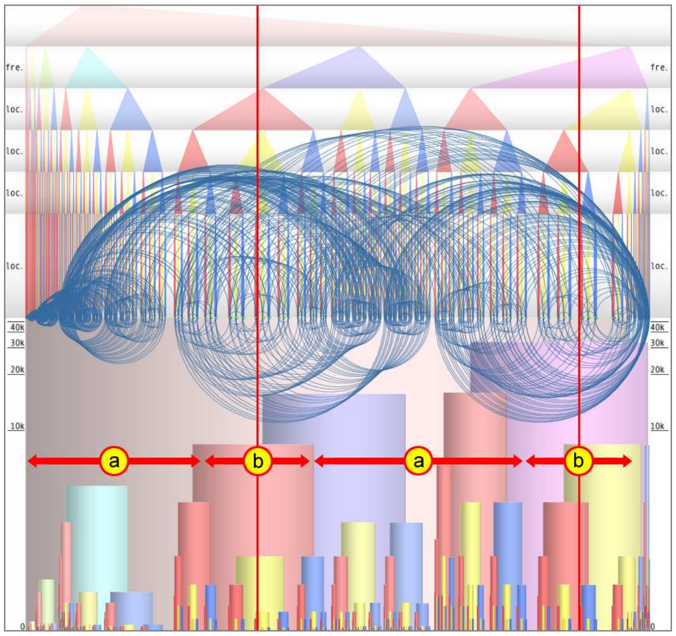

Fig. 7. Focus bands. (a) Compressed and (b) magnified regions.

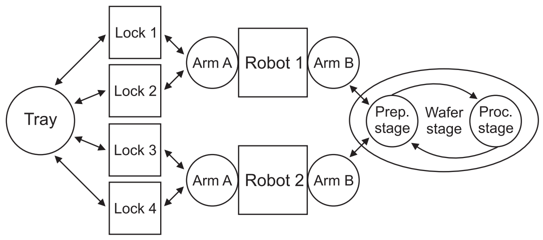

Fig. 8. Wafer stepper.

the direction of these bundles in the orientation of the arcs: arcs are always interpreted in a clockwise fashion. User feedback from this and previous experiments has been quite positive in this regard and users seem to have little trouble adopting this convention [12]. This makes visual pattern detection more meaningful since the directions of transitions play an important role in understanding the behavior that the transition graph describes. It also makes it unnecessary to clutter the screen with other visual cues such as arrow heads.

To deal with larger data sets, the user can define a number of vertical focus bands (see Figure 7). The visualization is magnified at these positions and compressed further away [13]. Although distortion-oriented techniques can be confusing, our users found it useful when applied in only one dimension. Since the magnification factor varies continuously, repeated patterns and symmetries are still discernable. Users did prefer a single focus band, though.

4 USE CASE

To show how our approach can assist users in the visual analysis of state transition graphs we present a real-world use case. More concretely, we apply our prototype to analyze the state transition graph of a generalization of the wafer handling of an industrial wafer stepper [5].

The wafer stepper we consider is used to manufacture integrated circuits as follows (see Figure 8). Fresh, unprocessed wafers enter the system via the tray, which can hold six wafers. From here they are moved to one of four locks that have a capacity of one wafer each. Wafers wait here to be transported by one of two robots. Robot 1 is allocated to locks 1 and 2 while robot 2 serves locks 3 and 4. The robots have two arms, A and B, that can pick up a single wafer each. When stationary, a robot has one arm facing the locks and one arm facing the wafer stage. To transport a wafer, a robot rotates on its axis. Wafers are first prepared on the preparation stage and then processed on the processing stage. Both have a capacity of one wafer. To exit the system, wafers follow a similar path in reverse.

The state transition graph that describes the wafer stepper contains 55 043 states and 289 443 transitions. It has 15 associated attributes,

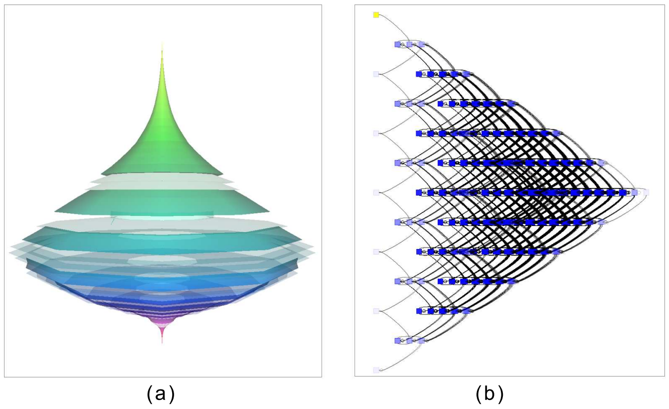

Fig. 9. Wafer stepper state transition graph visualized with (a) FSMView and (b) StateVis.

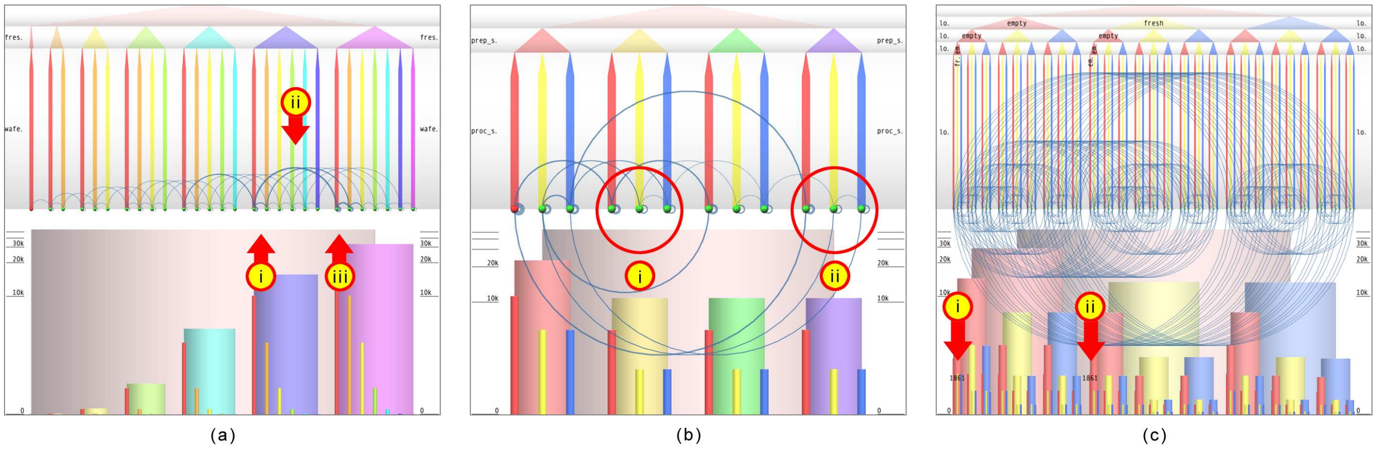

Fig. 10. Taking different perspectives. (a) From the tray's perspective, the system exhibits deterministic "forward" behavior. (b) Important requirements can be checked by considering the system from the wafer stage's perspective. (c) The regular nested patterns of the arcs indicate "symmetrical" behavior from the locks' perspective.

of which we consider the following:

- *fresh_wafer_list*: a list of fresh wafers on the tray;
- *wafers_in_system*: the number of wafers in the system;
- *prep_stage*: the contents of the preparation stage;
- *proc_stage*: the contents of the processing stage;
- *lock_1* through *-4*: the contents of the four locks;
- *robot_1_arm_a* and *-b*: the contents of the two arms of robot 1;
- *robot_2_arm_a* and *-b*: the same as above, but for robot 2;
- *fan_out*: a derived attribute that represents the number of outgoing edges for every node.

All attributes describing the contents of a particular component have the domain $\{empty, fresh, prepared, processed\}$.

When visualized using some of the more conventional graph drawing techniques, this transition graph basically resembles a "tangled ball of yarn". Consequently, very little can be learned. The visualization generated with FSMView [15] does not branch into symmetrical substructures (see Figure 9 (a)). That is, we are dealing with an example of the problematic graphs we referred to in Section 2. When viewed with StateVis [12], some suggestive behavioral patterns are seen (see Figure 9 (b)), but it is difficult to relate these back to concrete insights.

4.1 Taking different perspectives

Using our technique, it is possible to take specific perspectives on the system being studied. Below, we view the wafer stepper from three perspectives. This enables us to make important observations regarding system behavior and to check specific requirements.

Tray's perspective

To view the system from the tray's perspective we select the *fresh_wafer_list* and *wafers_in_system* attributes, in this order. Clustering results in Figure 10 (a). Our first observation is that there are three types of transitions: (i) those internal to *fresh_wafer_list* clusters and between *wafers_in_system* clusters, (ii) those between *fresh_wafer_list* clusters and (iii) self-loops. The latter, self-loops, can be thought of as *internal actions*. This is a notion borrowed from process algebra and refers to the hiding of certain actions with the aim of highlighting the influence of the remaining ones [3]. In this sense, our approach allows for dynamic action hiding.

An important insight from Figure 10 (a) is that from the tray's perspective, the system displays a clear determinism: there is a clear forward progression from an initial cluster (far left) to a final cluster (far right), where all wafers have been processed. This is seen by observing that there are no "back-pointing" arcs. By selecting corresponding bars in the bar tree visualization, we can confirm that the initial and final clusters contain a single node each. Along the way, the system gets exponentially more complex, but then starts tapering off toward the final node. Again, the bar tree visualization supports and strengthens this observation. We are now in a position to say something about the *liveness* of the system. This term, from the model checking literature [2], expresses that "a desirable state should eventually be reached".

Wafer stage's perspective

Clustering on the attributes *prep_stage* and *proc_stage* lets us consider the system from the perspective of the wafer stage (see Figure 10 (b)). With this result, we can validate important requirements.

Firstly, it is impossible for an unprepared wafer to be processed. If we consider the third level of the clustering hierarchy (corresponding to *proc_stage*), we see that only three values occur and that *fresh* is not one of them. (The same three colors are repeated and it suffices to check the values they encode only once.) This means that if the processing stage is not empty, it can only contain a prepared or a processed wafer. A second important and related requirement is that it should never be possible for the contents of the preparation and processing stages to be swapped when the preparation stage contains a fresh wafer. This is confirmed by noting that there are no arcs connecting (i) clusters where *prep_stage* has the value *fresh* to (ii) clusters where it assumes the value *processed*.

These requirements would call for significant effort to validate using other techniques. For instance, using formal methods, they would have to be meticulously formulated and their validity proved.

Locks' perspective

The final perspective that we consider is that of the locks. To do so, we select the attributes *lock_1* through *lock_4*. Figure 10 (c) shows the result after clustering. There is a clear recurring nested pattern in the visualization. Although it is most striking in terms of the arc diagram, we also see that there are corresponding regular patterns in the clustering hierarchy and bar tree. This illustrates our claim that our technique makes it possible to match correlating patterns in the three types of data being visualized.

Since the behavior of any one lock is independent of any other lock (the same behavioral pattern is present at all four corresponding levels), we can speak of behavioral symmetry. Furthermore, it possible to check that for similar permutations of values assumed by the attributes

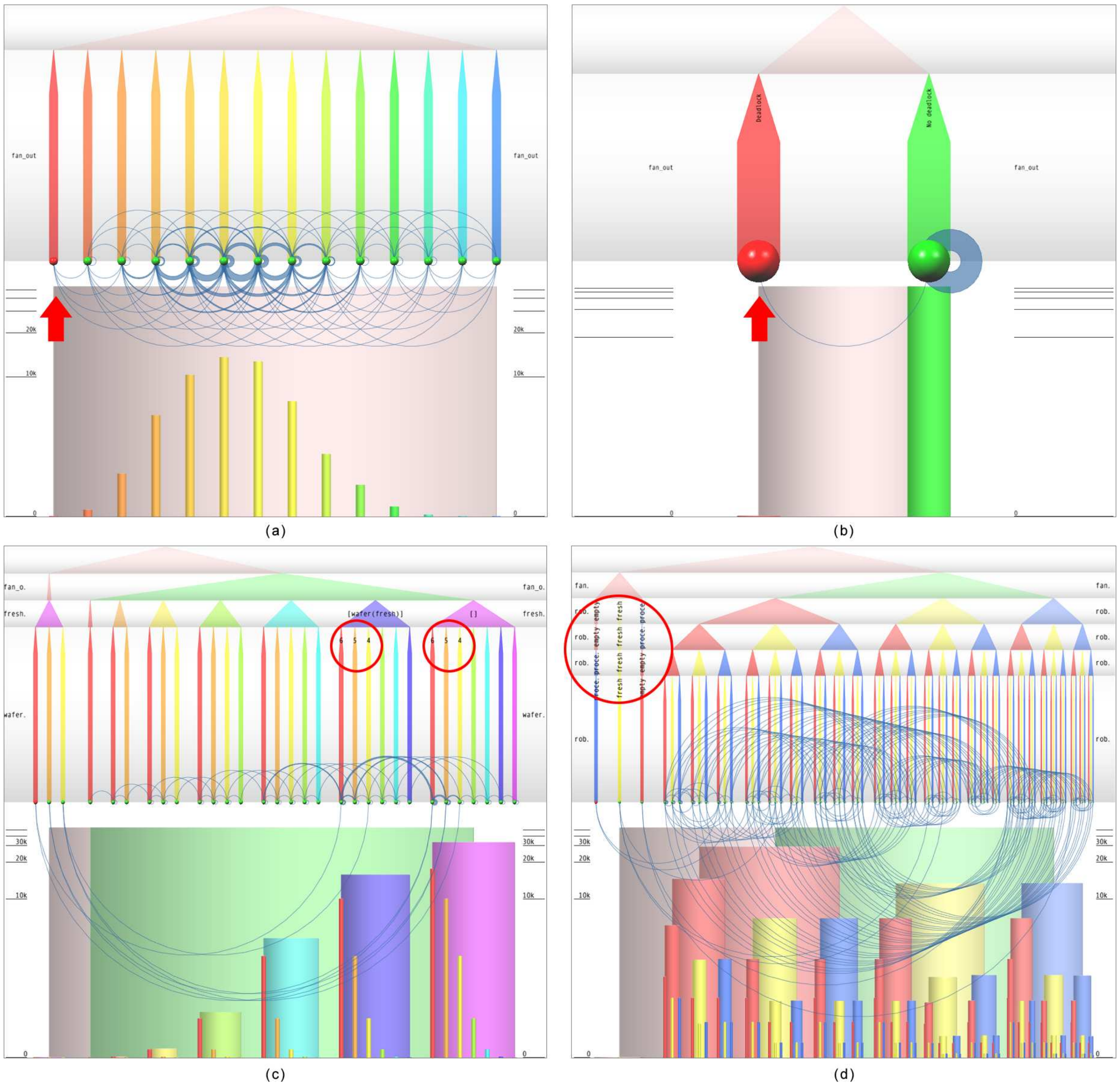

Fig. 11. Deadlock detection. (a) By clustering on *fan_out*, all deadlock nodes are in a single cluster on the far left. (b) To simplify the remaining analysis, the domain of *fan_out* is clustered to only distinguish between deadlock (left) and non-deadlock nodes (right). (c) Combining *fan_out* with attributes related to the tray suggests that deadlock can be prevented by limiting the number of wafers in the system. (d) Combining *fan_out* with attributes related to the robots reveals that deadlock only occurs when the robots have specific configurations.

(for example, (i) when locks 1–3 are empty and lock 4 fresh versus (ii) when lock 1 contains a fresh wafer and locks 2–4 are empty), the system contains an equal number of nodes (using the bar tree visualization). From this we can deduce that the four locks exhibit identical behavior. We put this to good use during the more detailed analysis in the next section.

4.2 Focused analysis

We now illustrate how our method can assist in a more in-depth, focused analysis. A common objective of system analysis is to identify *deadlock* states. These are occasions when a computer-based system "hangs" or ceases to function. In our state transition graph, these are nodes that do not have any outgoing edges.

Deadlock detection

The first step of deadlock analysis is to detect deadlock nodes. To do so, we utilize the attribute *fan_out*. When we cluster the wafer stepper transition graph on *fan_out*, we get the visualization in Figure 11 (a) where we can immediately detect the cluster of deadlock nodes on the far left. By interaction with the bar tree, we determine that there are exactly eight deadlock nodes in our system. In this respect, although FSMView [15] offers deadlock detection and highlighting facilities, the user still has to scan the visualization, panning and zooming, to identify such nodes. This does not guarantee that all such nodes are

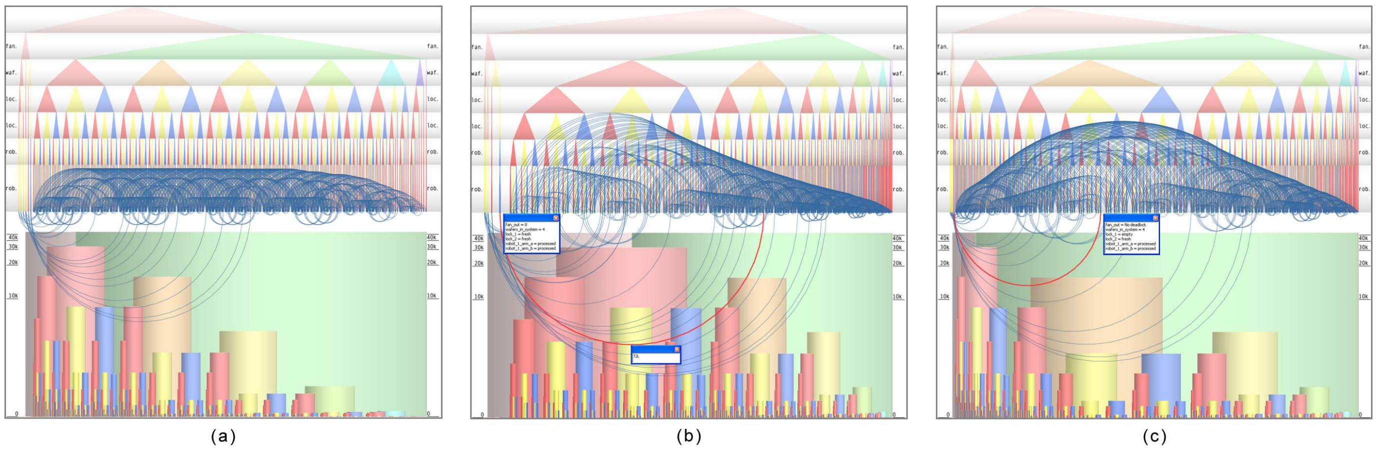

Fig. 12. Deadlock analysis. (a) As a result of attribute-based clustering, all deadlock nodes are positioned at the far left of the visualization. (b) The user can magnify the region containing deadlock nodes, view their attribute values and select incoming edges. (c) By locating the origin of such a highlighted edge and changing the focus position, the user can detect nodes leading to deadlock. The exact conditions resulting in deadlock are identified by analyzing the attributes of these nodes.

found. During a comparative study, we were only able to identify three deadlock nodes.

We also note that there are many (non-visual) tools that offer ways to detect deadlock. However, after detection, the user has to perform complicated analysis in order to learn more about the conditions that give rise to these occurrences. This includes scanning many lines of unformatted text output. With our technique it is much easier, as we show below.

Starting with the results in Figure 11 (a), we greatly simplify our analysis by noting that we are only interested in deadlock. We cluster the domain values of the *fan_out* attribute accordingly. When we now reapply hierarchical clustering, this results in a binary partitioning of our nodes as illustrated in Figure 11 (b) where all deadlock nodes are in the cluster on the left and all non-deadlock nodes are in the cluster on the right.

As suggested above, we want to learn more about the occurrence of deadlock in relation to other attributes. To see if there is any relation to the state of the tray, we cluster the transition graph on *fan_out, fresh_wafer_list* and *wafers_in_system*, in that order and with the domain of *fan_out* still clustered. This provides us with an interesting result. In Figure 11 (c) we see that deadlock nodes can only occur from nodes where there is a single or no unprocessed wafers left on the tray (this corresponds to the orange and pink triangles in the second level of the hierarchy). We observe this by noting the wide arcs at the bottom of the visualization (interpreted in clockwise fashion) leading to the deadlock nodes. Such transitions are only possible with four or more wafers in the system (consider the third level of the hierarchy). At this stage, we can already formulate a potential solution to prevent deadlock: limit the number of wafers inside the system at any point in time.

Another interesting observation can be made by clustering on the attributes *fan_out* and the four robot arms (see Figure 11 (d)). Since we know that all deadlock nodes are clustered toward the left of the visualization, we can immediately determine that deadlock only occurs under very specific conditions. That is, when either:

1. One of the robots has a processed wafer in each arm while both arms of the other robot are free.

2. When both robots have fresh wafers in both arms.

Deadlock analysis

Finally, we consider a specific deadlock occurrence. We assume that it suffices to analyze a single "wafer path". That is, we restrict our analysis to a single robot and the locks that it serves. This assumption is justified by our observation about the symmetric behavior of the locks in Section 4.1. In order to identify deadlock nodes, we also take *fan_out* into account when clustering. Consequently, we are presented with the results depicted in Figure 12 (a).

Notice at this stage, that we are dealing with a high-dimensional data set. However, despite the large size of our data, there is a clear visual structure due to the visualization of the clustering hierarchy. Also note that there are clear patterns visible in the arc diagram and bar tree. Even more interesting is the fact that the overriding regular pattern of the arc diagram is clearly disrupted by the arcs representing transitions to deadlock nodes.

When we zoom in on the deadlock nodes toward the left and select one of these (see Figure 12 (b)), this results in a pop-up window with an overview of all the selected attributes and the values they assume. We can also select one of its incoming transitions, which consequently lights up. By right-clicking, we also get an overview of all actions bundled in the arc. It is easy to visually trace this transition and zoom in on its originating cluster (see Figure 12 (c)).

Using this approach, we were able to identify the exact conditions that lead to this occurrence of deadlock. This is illustrated in Figure 13 and is related to our earlier observation regarding the states of the robot arms when deadlock occurs. In the initial state, robot 1 contains

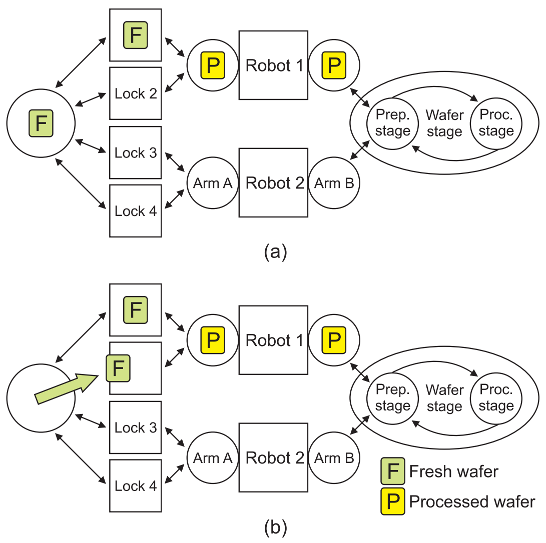

Fig. 13. Pre and post conditions of wafer stepper deadlock.

two processed wafers, lock 1 contains a fresh wafer and there is one fresh wafer on the tray. This is illustrated in Figure 13 (a). When a *tray_to_lock* action occurs, whereby this fresh wafer is transported to lock 2 (see Figure 13 (b)) we have deadlock. The reason for this is that once wafers have been processed, they cannot move “forward” to the wafer stage a second time. Furthermore, fresh wafers cannot move “backward” to the tray without first having been processed.

5 Conclusion

We have presented a new method for the visual analysis of multivariate state transition graphs. Our technique is based on attribute-based clustering which results in a significant reduction in complexity and serves as a useful analysis mechanism. Clustering produces hierarchical, metric and relational data that we represent in a single visualization. Unlike approaches that focus on graph topology, our method allows the user to consider the data in terms of knowledge they already possess: the semantics associated with the node attributes. Based on this knowledge, they are able to gain further insight into other aspects, such as relations, associated with the data. We are not alone in advocating an attribute-based approach for visualizing multivariate relational data. Recent work by Wattenberg also focuses on node attributes and makes a strong case for such a line of attack [19].

We have demonstrated the advantages of our approach with a use case from industry (containing 55 043 nodes and 289 443 edges). To do so, we have implemented a prototype that can be downloaded at [11]. Using our prototype, we have shown that for a number of problems our approach works better than existing visualization techniques for state transition graphs. To this end, we have introduced a novel visualization for encoding hierarchically structured quantitative data. We refer to it as the bar tree. We have also extended existing visualization techniques by combining arc diagrams with hierarchical information. We know of only one other approach that combines relational and hierarchical data using arc diagrams [10]. However, this approach uses containment and is limited in the number of hierarchical levels and relations that can be effectively visualized. Additionally, we have illustrated how the encoding of direction in the orientation of arcs enhances the visual language of arc diagrams.

With the aid of our prototype and use-case we have demonstrated how the benefits introduced in Section 1 can be realized. Our technique enables users to get a better intuition for the system being studied by taking different perspectives. Our technique also enables the user to conduct more focused analysis. In this regard, we have shown that it is possible to check requirements and to identify and clarify scenarios that lead to deadlock. We can also report that we have observed users talking about transition graphs and the behavior they describe in terms of the visual results of our prototype, using terms such as “symmetry”, “irregularity”, “third level” and “green cluster”. Users agree that this enhances communication between themselves and they expect that this will also hold for other stakeholders.

One open question is whether this approach is useful for arbitrary multivariate graphs (email traffic and citation networks, for example). In our graphs, nodes have multivariate attributes that are discrete and of low cardinality. Larger, possibly continuous domains can be handled with domain clustering (see Section 3). For instance, an age attribute associated with email users can be clustered into a small number of age groups. Our graphs are generated from system specifications [3] and generally contain some degree of regularity in terms of their edges (representing behavior). To cater for more irregular cases our approach can be used to study higher level selections. For instance, to study the relations between gender and country in email traffic. Another interesting solution would be to extend our approach with edge filtering. Using such a facility, it would be possible to analyze email traffic between specific geographic regions, for instance. These are promising possibilities and we aim to apply our approach to data sets from other domains to test its general applicability.

As far as the visual analysis of state transition graphs is concerned, our results suggest a number of interesting opportunities for future research. We would like to incorporate more ideas from abstract interpretation [2]. Details about so-called *may*- and *must*-transitions are high in our list of priorities. When edges between two clusters of nodes are bundled, a number of different actions are usually grouped together. Some of these actions are possible from all nodes in the originating cluster *(must)* while some are not *(may)*. Related to this, is the notion of transition attributes. For instance, it is often the case that a number of data parameters are associated with actions. Although not straight-forward, we would like to investigate whether and how such multivariate edges could be accommodated in our approach.

Acknowledgements

This research was funded by the Netherlands Organization for Scientific Research (NWO) under grant 612.065.410. We gratefully acknowledge ASML for granting us permission to present the wafer stepper use case. We also thank Jeroen van der Wulp (TU/e) who created the behavioral model from which our data set was generated. We thank the members of the TU/e VIS and OAS groups, particularly Lucian Voinea and Bas Ploeger, for their valuable suggestions and input. Finally, we thank the anonymous referees for their constructive feedback on an initial draft of this paper.

References

[1] A. Arnold. *Finite Transition Systems*. Prentice Hall, 1994.

[2] D. Dams and R. Gerth. Abstract interpretation of reactive systems. *ACM Transactions on Programming Languages and Systems*, 19(2):253–291, 1997.

[3] W.J. Fokkink. *Introduction to Process Algebra*. Springer, 2000.

[4] E.R. Gansner and S.C. North. An open graph visualization system and its applications to software engineering. *Software - Practice and Experience 2000*, 30(11):1203–1233, 2000.

[5] M. Hendriks, B. van den Nieuwelaar, and F. Vaandrager. Model checker aided design of a controller for a wafer scanner. In *Proceedings of the First International Symposium on Leveraging Applications of Formal Methods 2004 (ISOLA '04)*, pages 201–209, 2004.

[6] T. Jeron and C. Jard. 3D layout of reachability graphs of communicating processes. In *Proceedings of the DIMACS International Workshop on Graph Drawing (GD '94)*, pages 25–32, 1994.

[7] B. Kerr. Thread Arcs: an email thread visualization. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '03)*, pages 27–34, 2003.

[8] M. Leuschel and E. Turner. Visualizing larger state spaces in ProB. In *Proceedings of the Fourth International Conference of B and Z Users*, pages 6–23, 2005.

[9] T. Mihalisin, J. Timlin, and J. Schwegler. Visualization and analysis of multi-variate data: a technique for all fields. In *Proceedings of IEEE Visualization (Visualization '91)*, pages 171–178, 1991.

[10] P. Neumann, S. Schlechtweg, and S. Carpendale. ArcTrees: visualizing relations in hierarchical data. In *Proceedings of the Eurographics IEEE VGTC Symposium on Visualization (EuroVis 2005)*, pages 53–60, 2005.

[11] A.J. Pretorius. www.win.tue.nl/˜apretori/noodleview, 2006.

[12] 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.

[13] M. Sarkar and M. H. Brown. Graphical fisheye views of graphs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '92)*, pages 83–91, 1992.

[14] B. Shneiderman. Tree visualization with Tree-Maps: 2-D space filling approach. *ACM Transactions on Graphics*, 11(1):92–99, 1992.

[15] 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.

[16] F. van Ham and J.J. van Wijk. Beam Trees: compact visualization of large hierarchies. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '02)*, pages 93–100, 2002.

[17] C. Ware. Designing with a 2 1/2D attitude. *Information Design Journal*, 10(3):255–262, 2001.

[18] M. Wattenberg. Arc Diagrams: visualizing structure in strings. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis '02)*, pages 110–116, 2002.

[19] M. Wattenberg. Visual exploration of multivariate graphs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '06)*, pages 811 – 819, 2006.