Visualization of state transition graphs

Visualization of State Transition Graphs

Visualization of State Transition Graphs

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen op
donderdag 6 november 2008 om 16.00 uur

door

Andries Johannes Pretorius

geboren te Pretoria, Zuid-Afrika

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr.ir. J.J. van Wijk
en
prof.dr.ir. J.F. Groote

A catalogue record is available from the Eindhoven University of Technology Library

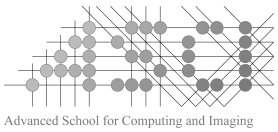
ISBN: 978-90-386-1405-2

To my parents

Eerste promotor:
prof.dr.ir. J.J. van Wijk (Technische Universiteit Eindhoven)

Tweede promotor:
prof.dr.ir. J.F. Groote (Technische Universiteit Eindhoven)

Kerncommissie:
prof.dr. G. Melançon (Université Bordeaux I, France)
prof.dr.ir. J.E. Rooda (Technische Universiteit Eindhoven)
prof.dr. J.T. Stasko (Georgia Institute of Technology, USA)



This work was financially supported by the Netherlands Organisation for Scientific Research (NWO) as part of the VoLTS project. NWO grant number 612.065.410.

This work was carried out in the ASCI graduate school (Advanced School for Computing and Imaging). ASCI dissertation series number 167.

Published by Technische Universiteit Eindhoven

Typeset in LaTeX
Cover by Oranje Vormgevers, Eindhoven
Printed by Gildeprint Drukkerijen, Enschede

Contents

Preface

As I walked back to Eindhoven's station on a scorching July afternoon in 2004, I thought it ironic that I was desperate for a cloud to block the sun. As a South African, I should have been happy with a spell of warm weather in the Netherlands, a country famous for being damp and cold.

I had not anticipated taking the train to Eindhoven the day before when Prof. Jarke J. van Wijk phoned me at a university lab in Brussels. Following an inquiry I had made, he wanted to see me urgently to discuss the possibility of doing doctoral research at Eindhoven University of Technology. This would be a joint project with his Visualization group and the System Design and Analysis group, headed by Prof. Jan Friso Groote. It was summer, it was hot, and Prof. Van Wijk was eager to leave for his holiday. "Please prepare a presentation. After that you'll be interviewed. How does tomorrow sound?"

A day was not much time to prepare, but I enthusiastically accepted. Prof. Van Wijk, whom I got to know as Jack, is widely respected for his visualization research, the field I wanted to do my PhD in. The possibility of a collaboration with Prof. Groote also seemed very promising.

I quickly learned that Jack's reputation is not overrated. He is incisively bright, resolute and creative, with high standards. Jack leads by example and conducts cutting edge research despite a considerable management workload. However, his students remain his absolute priority and I got to know Jack as someone who was genuinely interested in my research and my general well-being. I am certain that all the Van Wijk alumni would attest to this. Jack, you were a mentor in the true sense, something for which I am sincerely grateful. It really has been an honor and a privilege to have you as my PhD supervisor.

For my research, it was fundamental to have close collaboration with my target user group: system analysts. This was largely facilitated by my second supervisor, Prof. Jan Friso Groote, whom I would like to thank for introducing me to and guiding me through the often bewildering world of formal system analysis. Jan Friso, your enthusiasm for your subject and your Frisian sense of humor made this a particularly pleasant experience.

The caliber of people I crossed paths with at TU/e has been remarkable. Frank van Ham (IBM Research, USA) had booked promising results on transition graph visualization while completing his PhD in Eindhoven. This was an important starting point for my own research. Frank's company during my first year at TU/e is also much appreciated. Further, I thank Prof. Koos Rooda and Albert Hofkamp (Systems Engineering) and Prof. Wil van der Aalst and Eric Verbeek (Information Systems) for fruitful collaborations. I am pleased that Frank van Ham and Prof. Rooda have agreed to serve on my doctoral

committee.

A major perk of PhD research is having contact with clever and enthusiastic young people. Many thanks to Lucian Voinea, Dennie Reniers, Danny Holten, Koray Duhbaci, Yedendra Shrinivasan, Jing Li and Niels Willems, fellow doctoral students in the Visualization group, for creating an excellent environment to work in. Your input and our many debates, serious and not so serious, were always appreciated and were fundamental in defining my PhD experience. Similarly, I thank the senior members of our research group: Prof. Robert van Liere, Kees Huizing, Huub van de Wetering and Alex Telea. I also thank my system analysis colleagues for valuable contributions and good company: Aad Mathijssen, Muck van Weerdenburg, Bas Ploeger, Jeroen van der Wulp, Frank Stappers, Mohammad Mousavi, Wieger Wesselink and Michel Reniers.

As many graduate students have experienced, administrative red tape can be a significant challenge when enrolling at a foreign university. I thank Tineke van den Bosch and Elisabeth Melby for their friendly assistance in taming this complexity. I am also greatly indebted to the Netherlands Organisation for Scientific Research (NWO) for funding my PhD research as part of the VoLTS project (grant 612.065.410).

Being part of an international network of researchers has been particularly enriching. I thank Dr Tamara Munzner (University of British Columbia, Canada) for two intense and productive days of discussion when she visited Eindhoven in 2007. Furthermore, I am grateful to Prof. John T. Stasko (Georgia Institute of Technology, USA) and Prof. Guy Melançon (Université Bordeaux I, France) for reviewing an earlier draft of this dissertation and for serving on my doctoral committee. My thanks also go to Prof. Jos Roerdink (University of Groningen) for participating in my doctoral committee.

On a personal level, support and interest from South Africa were much appreciated. Thank you to my good friend Daan Jacobs, who phoned me every now and again, "just to check how things are going." Also to Nelis Franken, who knows all about PhD life, and to Jennifer Walker, for sharing her English language expertise.

This dissertation is dedicated to my parents. Ma, for your intellect, your determination and your insatiable curiosity. Pa, for your good character, your courage to take roads less traveled and your resolve to finish what you started. I was very fortunate to grow up in a loving home filled with books, music and art, where there were always various projects in the making and where not only academic achievement but also creativity was valued. All this, and the privilege to travel abroad as a child, have undoubtedly influenced me to pursue graduate studies in the Netherlands. For their part in the above, I also thank my brother Herman, my sister Jané and my extended family.

Finally, I wish to thank my wife, Tineke, and our 16-month-old son, Casper, for their unconditional love and support. Tineke, your commitment, determination and refusal to be satisfied with half-measures are traits I really appreciate. Casper, you cannot imagine the immense joy it gave me to put the finishing touches on this dissertation with you in the vicinity. Your ingenuity in coming up with ways to get me up from my chair to play and your wonderful sense of humor have been a true blessing.

Hannes Pretorius
Eindhoven
September 2008

Chapter 1

Introduction

> Programmed computers opened new possibilities, because they could execute long sequences of instructions at high speeds. Suddenly it was possible to explore models that were orders of magnitude more complex. This blessing also entailed a curse: you could run amok in detail, to the point that you would lose all possibility of uncovering overarching principles.
>
> (John H. Holland, *Emergence,* 1998)

In the above extract, John Holland, a pioneer in computer science, recounts how in 1952 he and colleagues were experimenting with the IBM 701 computer [34]. Holland writes of the excitement of experiencing how the advent of digital computers was ushering in a new era of computer-based modeling. For the first time far more complex models could be built and explored than had previously been possible. This came at a price, however. As Holland and his colleagues discovered, there is an inverse relationship between the complexity of a model and the ability of humans to understand the full extent of the resulting behavior.

1.1 Complex systems

During the six decades since Holland first encountered them, the application of computers has shifted from pure scientific work. Computer systems are becoming ubiquitous and are infiltrating many parts of professional and private life [1]. They now play a role in diverse arenas such as business, communication, education, entertainment, health care, leisure, and research and development.

The excerpt at the start of this chapter is from a book that explores the phenomenon of emergence; how sets of precise rules give rise to vast and complex systems [34]. Apart from being useful tools to study such systems, computer systems themselves suffer from the emergence of complexity.

All computer systems follow a number of rules in the form of instructions embedded in software. Computer programs are written by humans, usually in high level program-

ming languages. Such a language enables programmers to describe a list of instructions to be executed by a computer. The descriptive nature of programs makes them suitable for human examination [43]. This is done for a purpose: in addition to being machine interpretable, programs are meant to be understood by people [16].

Despite this, programmers and engineers are faced with immensely complex computer systems of which the true behavior is often not fully understood [23]. Communication of these systems with each other, via computer networks, and with their environment makes it even harder to correctly predict their behavior.

1.2 System behavior

One way to analyze computer systems is through modeling by formal specification [32]. In such models, an attempt is made to rigorously define only essential behavior. To do so, analysts typically exclude implementation specific aspects, which may vary between different target hardware configurations. As an example, consider the following description of the behavior of a traffic light consisting of a red, an amber and a green signal:

$$
\begin{aligned}
&\textbf{act} \quad to\_red, to\_amber, to\_green;\\
&\\
&\textbf{proc} \quad \mathit{TrafficLight}(red : \mathbb{B}, amber : \mathbb{B}, green : \mathbb{B})\\
&\qquad\qquad = red \rightarrow to\_green \cdot \mathit{TrafficLight}(false, false, true)\\
&\qquad\qquad + amber \rightarrow to\_red \cdot \mathit{TrafficLight}(true, false, false)\\
&\qquad\qquad + green \rightarrow to\_amber \cdot \mathit{TrafficLight}(false, true, false);\\
&\\
&\textbf{init} \quad \mathit{TrafficLight}(true, false, false);
\end{aligned}
$$

First, three actions are defined: to_red, to_amber, and to_green. These actions model operations the system can perform. Next, *TrafficLight* is defined as a process that takes three Boolean arguments: red, $amber$, and $green$. This vector of values represents the current system state; for each of the three signals (red, amber, and green) this vector encodes whether it is switched on or off.

The behavior of the process is specified by three conditional statements, or rules, of the form $condition \rightarrow result$, separated by $+$, the *else* operator. For example, if red evaluates to true then the to_green action is performed, followed by a recursive call to *TrafficLight* with a new state as argument. By calling $\mathit{TrafficLight}(true, false, false)$ the process is initiated in a state where the red signal is turned on and both the amber and green signals are turned off.

The above specification does not describe the implementation of the system. It also abstracts from time-related issues such as delays between signal switches. This results in a compact behavioral model which analysts can use to analyze the system's core behavior. Note though, that such specifications are implicit descriptions and that even for this small example, reasoning about the resulting behavior is not entirely trivial. For example, some deduction is required to determine whether the specification results in a correct sequence of signal changes.

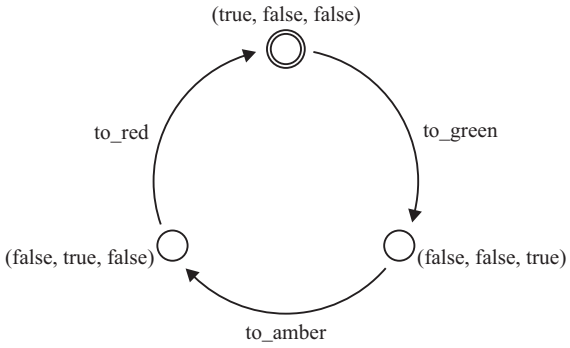


Figure 1.1: Behavior of a traffic light.

1.3 State transition graphs - a first look

A popular approach for studying models of system behavior, which addresses issues such as those raised above, is to explicitly enumerate all possible system states [4]. For the above specification the result is shown in Figure 1.1. Every system state is represented by a circular icon and labeled with a state vector; the values of the variables red, $amber$ and $green$ in the original system specification. The initial state is at the top, marked by a double outline. Transitions between states are shown as arrows and labeled with the action that causes a state change to occur.

This representation shows that the specification in Section 1.2 generates a system that cycles through exactly three states. Furthermore, from the initial state there is only one possible sequence of actions before returning to this state: to_green, to_amber, to_red.

Once the behavior of a single traffic light has been specified it becomes possible to construct a system that consists of a number of lights. Analysts may want to study the behavior of three lights at a street junction, for instance. Although the behavior of every individual light is still accurately described by the set of rules in Section 1.2, the result of interleaving the behavior of three traffic lights is far more difficult to predict than a single instance. Again, all states can be enumerated and visually represented. The result is shown in Figure 1.2, which is far more complex than Figure 1.1, and due to the visual clutter, difficult to study.

This reintroduces the notion of complexity briefly treated at the start of this chapter. To analyze a system of multiple traffic lights, details considered irrelevant have been successfully abstracted from. Yet, again the problem of complexity emerging from a set of apparently simple rules is encountered.

Figure 1.1 and Figure 1.2 are visual representations of state transition graphs [4]. In such a graph, nodes represent states a system can be in. Links, or directed edges, represent transitions between states. State transition graphs play an important role in computer science and engineering because they are used to analyze and understand the behavior of complex computer-based systems.

By depicting state transition graphs graphically, as node-link diagrams, system an-

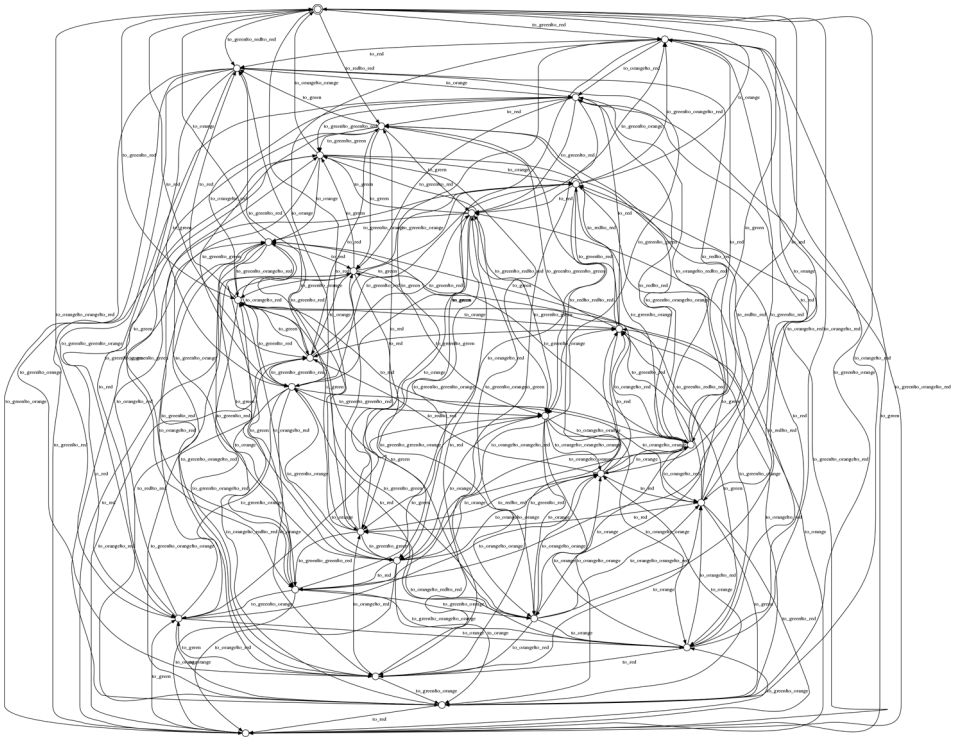


Figure 1.2: Behavior of three interleaved traffic lights.

alysts can reason about the behavior they describe. Visualizations of transition graphs need not be restricted to static representations like Figure 1.1 and Figure 1.2, however. As a result of spectacular advances in computing, computer graphics hardware has advanced to a point where millions of graphical primitives can be rendered to screen in real time [19]. This enables researchers to cross over from static graphics to interactive visualizations [12].

Research in visualization investigates the application of interactive computer graphics to understand large and complex data sets [38]. State transition graphs fall into this category; they often contain tens of thousands of nodes, or more, and tens to hundreds of thousands of edges. Even for relatively small graphs, like the one depicted in Figure 1.2 (27 nodes and 189 edges), the complexity of the behavior they describe hinders analysis and insight.

1.4 This dissertation

As shown in this dissertation, visualization can assist analysts during the investigation of state transition graphs. The central research question being addressed is as follows:

Research question. How can interactive visualization be used to gain insight into state transition graphs?

State transition graph visualization has a large solution space. Since very few techniques have been developed to explore this space, it is almost without precedent and best practices are unknown. This implies that confidently formulating a set of requirements for techniques to visualize state transition graphs is difficult.

It also presents a unique opportunity to take an experimental and exploratory approach toward developing visualization techniques for state transition graphs. A starting assumption for the research presented in this dissertation was that no single optimal solution for the visualization of state transition graphs is likely to exist. This offered the chance to investigate many different solutions and several new methods were developed in close collaboration with system analysts. Over an extended period of time, an approach based on iterative prototyping was undertaken. Techniques were continually updated and improved. This was achieved by considering evaluation results and by paying attention to user feedback. Another important factor was gaining a better understanding of the application domain.

Chapter 2, *Background,* provides a more thorough context for the work discussed in this dissertation. The domain of system analysis is described with emphasis on the role of state transition graphs. Arguments for the effectiveness of visualization and an introduction to the field of information visualization are provided. Finally, the methodology that was applied to develop the techniques presented in subsequent chapters is briefly outlined.

Chapters 3–7 contain the main contributions of this dissertation. In each of these chapters a novel visualization technique is presented. A rationale is presented first, followed by the details of the technique. Next, the technique is evaluated with the aid of one or more case studies. Finally, every chapter is concluded with a discussion. The only exception to this structure is Chapter 3, where a case study is interleaved with the description of the visualization technique. Since the presented technique consists of a number of sub-techniques, this results in a more fluent exposition.

In Chapter 3, *Selection and Projection,* the visualization of state transition graphs is considered as a multivariate visualization challenge. A number of techniques for projecting transition graphs to the 2D plane, based on attributes associated with their nodes, are presented. In addition to a case (the Alternating Bit Protocol), the ability of the approach to assist analysts in answering a number of questions about their data is considered. This work was first published in [59].

Chapter 4, *Attribute-Based Clustering,* introduces another approach for visualizing transition graphs as a function of their node attributes. The aim is to enable analysts to relate the position of nodes with the meaning of the associated attributes. This is achieved by providing an interactive attribute-based clustering facility. As validation a case study of a wafer stepper system is presented. This work was first published in [60].

In Chapter 5, *User-Defined Diagrams,* the tendency of analysts to reason about systems in terms of diagrammatic representations is investigated. To support this, a method is presented with which users can define diagrams and parameterize these by linking their graphical properties to state transition graphs. As validation two cases are discussed: a

wafer stepper and a paint factory. This work was first published in [61]. Section 5.1.2 is based on [48] with both authors contributing equally. Parts of Section 5.7 are based on [84], with all authors contributing equally.

In Chapter 6, *Multiple Views on Traces,* linear paths in state transition graphs are considered. Since analysts are often interested in studying such system traces, they are provided with different views on the data. The merit of the approach is illustrated with a case study of an industrial steam generator. This work was first published in [62].

Chapter 7, *Querying Nodes and Edges,* addresses the issue of including edge labels as first-class citizens in the visualization of state transition graphs. Via direct manipulation the user can interactively answer queries related to both node attributes and edge labels. To validate the approach it is contrasted with current practice. Three cases of the application of the method to state transition graphs that model real-world systems are also provided (a traffic regulator, a communication protocol and a board game). This work was first published in [63].

Chapter 8, *Reflections,* casts a retrospective view on the process of developing visualization techniques for the analysis of state transition graphs. It highlights how an iterative experimental approach results in a better understanding of both the problem space and the solution space. By taking a wider view it also sheds light on typical challenges that confront information visualization designers and how these can be dealt with.

Chapter 9, *Conclusion,* gives an overview of the main contributions of this dissertation and their implications. Open issues are also outlined and suggestions are made for future work.

Chapter 2

Background

In Chapter 1 the goal of this dissertation was introduced: to investigate interactive visualization techniques to help system analysts gain insight into state transition graphs. The current chapter provides a more thorough overview of the part that state transition graphs play in system analysis. The role of system analysts and approaches currently being used for transition graph visualization are considered. The field of visualization research is introduced next and a number of arguments for the merit of visualization are provided. Consequently, the discipline of information visualization, which focuses on abstract data such as transition graphs, is considered. A few multivariate and graph visualization techniques related to the work presented in this dissertation are also discussed. Finally, the methodology that was applied to develop the techniques presented in this dissertation is outlined.

2.1 State transition graphs

State transition graphs form the standard semantic framework underlying contemporary specification and programming languages [4]. Consequently, many system analysis techniques involve the translation of behavioral specifications into state transition graphs. To study the behavior of complex computer-based systems, computer scientists and engineers apply the following three-step methodology:

1. Formally describe the system being studied.
2. Generate a transition graph from the description obtained in Step 1.
3. Analyze the transition graph obtained in Step 2 to gain more insight into the original system.

A popular approach to formally describe system behavior is to use specification languages based on process algebra [18]. Using a language such as mCRL2 [24], its predecessor μCRL [11], or Chi [8], analysts characterize the behavior of different processes which can communicate. The specification in Section 1.2 is, in fact, a fragment of mCRL2 code.

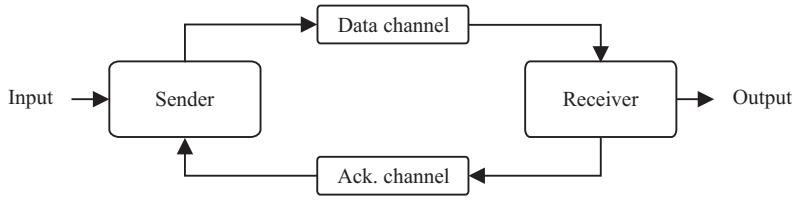


Figure 2.1: Simple communication protocol.

Typically, an analyst will describe the behavior of a system by specifying every one of its components as a separate process that can exchange data with other processes. For example, consider the simple communication protocol shown in Figure 2.1. This system consists of a sender, a receiver and two communication channels, each of which will be specified separately. Taking such a modular approach makes it possible to describe every component of a system independently and at an appropriate level of detail.

To get a behavioral model of an entire system, analysts use automated tools to interleave the behavior described by the formal specifications of its different composite processes [18]. This generates a state transition graph where nodes represent system states and edges represent transitions between states.

After a transition graph has been generated, analysts use various methods to analyze it. When problems or mistakes are discovered in Step 3 of the above methodology, analysts have to decide whether these are bugs in the specification (Step 1) or real system issues. This implies that they first have to ensure that the specification is an accurate behavioral description. When they are convinced of this, analysts can analyze the actual system behavior.

Note that formal specifications are implicit functional descriptions of system behavior while state transition graphs are explicit enumerations of system states and transitions [18]. Because it is difficult to accurately predict the behavior of a system directly from its specification, particularly when it consists of a number of processes that communicate, state transition graphs are often studied to fully grasp system behavior.

The nodes of a state transition graph consist of a valuation of a vector of data attributes [4, 18]. This state vector corresponds to the union of all data parameters originally introduced in the specification of the system. As a result, there is a strong relationship between the values of the attributes in this vector and the formal description from which the transition graph was derived. For instance, every node of the transition graph that describes the behavior of the communication protocol introduced above will consist of a vector of variables that respectively describe the status of the sender, the receiver, and the two communication channels (see Figure 2.1).

The edges of a state transition graph represent transitions between states and are labeled. These labels represent the actions that result in state changes. For the communication protocol, which can perform actions to send and receive data packets, edges will have labels such as *send* and *receive*. To make the above more explicit, the following definition specifies the data that will be considered in this dissertation:

State transition graph. Let $D_1, ..., D_n$ and L be discrete domains. A state transition graph G is a triple (V, E, v_0) where

- $V \subseteq D_1 \times ... \times D_n$ is the set of nodes in G.
- $E \subseteq V \times L \times V$ is the set of directed edges in G.
- $v_0 \in V$ is the initial node of G.

Every node $v \in V$ has the form $v = (v_1, ..., v_n)$ with $v_i \in D_i$. Also, every directed edge $e \in E$ has the form $e = (v, l, v')$ for $v, v' \in V$ and $l \in L$. $D_1, ..., D_n$ are referred to as the node attribute domains and L is the set of edge labels. A transition graph has a special node that represents the state in which the modeled system starts, the initial node v_0. In the remainder of this dissertation, when v_0 is not relevant, a transition graph G will be denoted by a pair (V, E). State transition graphs are also known by the synonyms state transition systems, state spaces and (finite) state machines.

Conceptually state transition graphs are straightforward. They are not easy to study, however. Transition graphs often contain tens of thousands of nodes, or more, and tens to hundreds of thousands of edges. This phenomenon, known as the state explosion problem [4], hinders analysis and insight. Furthermore, transition graphs describe system behavior at a low abstraction level, making it difficult for analysts to map their domain knowledge to nodes and edges found in the data.

2.2 Transition graph analysis

Even though state transition graphs are not always easy to study, system analysts are interested in them because they enable them to do two types of work:

- Identify and correct behavioral problems of an existing system. For example, two errors in a shared memory implementation of the Java programming language were found in this way [55].
- Design the behavior of a new system by modeling it before it is implemented. For instance, a number of serious issues in a proposed automated parking system were identified during such verified design [48].

Analysts have devised a number of approaches for addressing the size and complexity of state transition graphs. The traditional tactic is to derive and analyze an abstracted much smaller variant. For example, by using bisumulation equivalence [18], nodes that are similar in terms of the branching structure of the graph, relative to the initial node, are mapped onto each other.

The second approach is to formulate and check requirements using a technique called model checking [14]. First, questions about the graphs are rephrased as formal predicates. Such a predicate serves as input for a model checker tool that searches the entire state transition graph to determine if the predicate always holds. The analyst is then presented with a *yes* or a *no* as a result. When a predicate has been found to be false, some tools

provide the user with paths from the initial node to those nodes where the condition does not hold.

Finally, analysts often resort to simulation tools [72]. A simulator lets the user consider a current state and an overview of all transitions branching out from it. When one of these transitions is selected, the simulator updates the current state and new transitions become possible.

All three of these approaches are indispensable for the analysis of system behavior. They also have drawbacks, however. Transition graphs that have been reduced in size with the aid of bisumulation equivalence retain relevant behavior, but it is difficult to follow the behavior represented by individual node attributes. Moreover, even after a significant reduction in size, such reduced graphs may be too large to understand without further support.

Model checking assumes that all, or at least some, system requirements are known in advance. If this is not the case, it is a challenge to adequately verify a system. Furthermore, system analysts have to understand systems very intimately in order to formulate predicates and interpret the results of model checkers.

Simulators offer a very restricted view of the transition graph and the behavior of the system it models. Most often analysts can only consider those transitions and states that are one step removed from the current state. Also, the transition graph usually has to be traversed from the initial node. Consequently, as is the case with the other techniques discussed above, analysis requires a significant investment of time and effort.

2.3 Transition graph visualization

From the above mentioned shortcomings, it is concluded that there is an opportunity and a need for alternative approaches for the analysis of state transition graphs. This dissertation presents a number of interactive visualization techniques for state transition graphs. To ensure that system analysts play a key role, the research question introduced in Section 1.4 is refined into two more concrete aims. The techniques that are developed should:

- Enable users to get a better intuition about the systems they study. Even if they do not have precise questions, users should be enabled to get a feeling for the data.
- Enable users to investigate particular features and answer specific questions about the systems described by their data.

Existing research results suggest that visualization can play an important role during the analysis of state transition graphs. Off-the-shelf graph drawing tools, like Dot and Neato developed at AT&T Labs [21], have been widely used. In fact, Figure 1.2 in the previous chapter was generated with Dot.

The primary drawback of general graph drawing tools is their lack of scalability. Although Figure 1.1 is very effective in illustrating the behavior of a single traffic light, visualizations for larger graphs, such as Figure 1.2, are harder to interpret. As another example consider Figure 2.2, in which the behavior of a simple lookup algorithm is visualized [44]. The graph consists of only 65 nodes and 433 edges, but in terms of the two

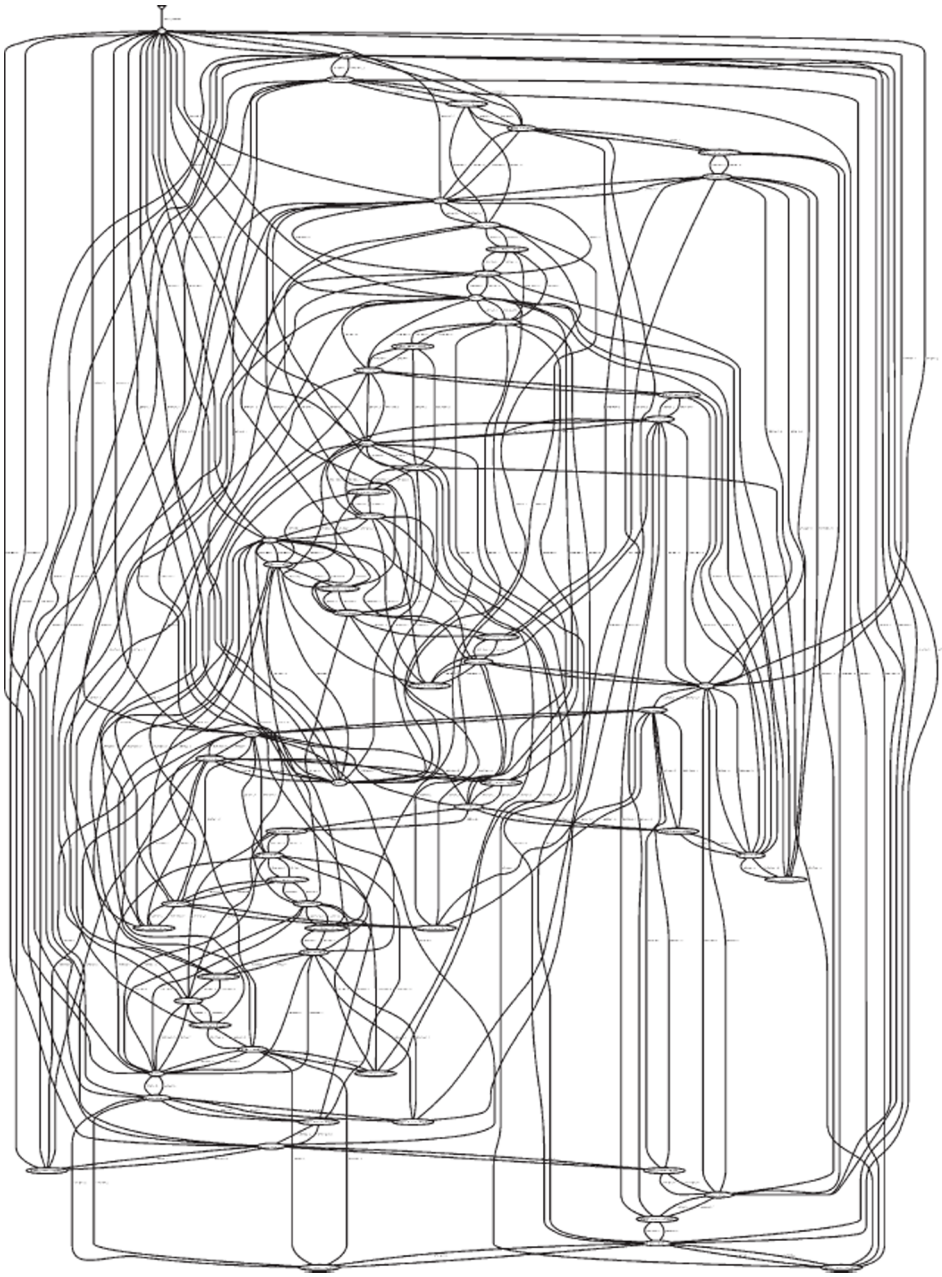


Figure 2.2: Behavior of a simple lookup algorithm [44].

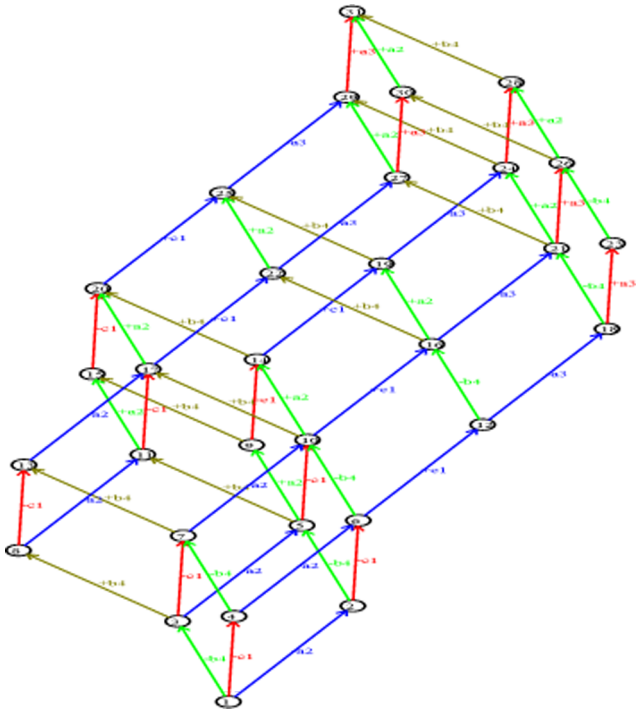


Figure 2.3: 3D layout of a state transition graph [37].

aims introduced above, this representation prevents the user from getting a good overview of system behavior. Visual clutter also makes it difficult to identify nodes with interesting properties. Some effort is needed, for instance, to determine whether all nodes in the graph have outgoing edges.

The algorithms used by most general purpose graph drawing tools attempt to find a balance between a number of aesthetic criteria and constraints [7]. Aesthetics include aspects such as minimizing the number of edge crossings and keeping the area covered by the drawing small. Constraints include orienting paths, or sequences of consecutive directed edges, in a top-to-bottom fashion as far as possible.

For graphs where the average degree of every node is low (that is, where most nodes have few direct neighbors), these algorithms generally perform well. Due to the high prevalence of parallelism in transition graphs, however, this property cannot be taken for granted. In fact, as a result of the high degree of many nodes in transition graphs, general graph drawing tools often generate unsatisfactory visualizations such as Figure 1.2 and Figure 2.2. To address the drawbacks of traditional graph drawing algorithms a few techniques specifically for the visualization of state transition graphs have been developed.

Jéron and Jard [37] proposed a layout algorithm for viewing state transition graphs in 3D. This algorithm guarantees that every node is allocated a unique position in 3D space. As Figure 2.3 shows, concurrency is also highlighted by diamond-shaped lattices. Based

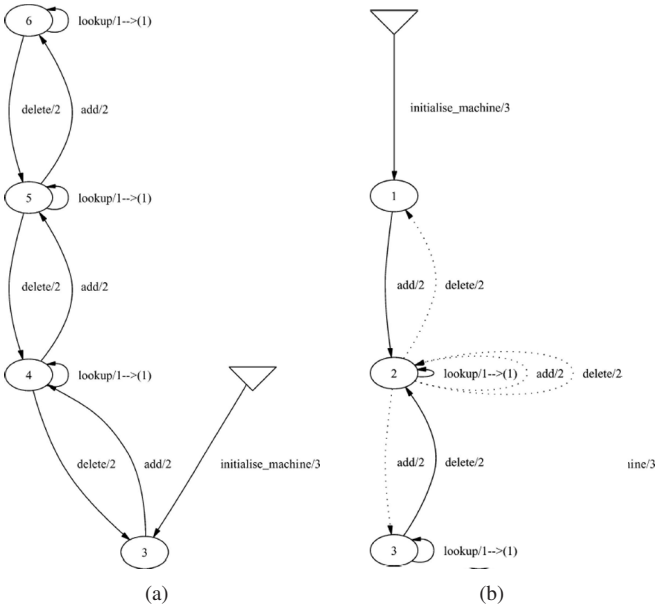


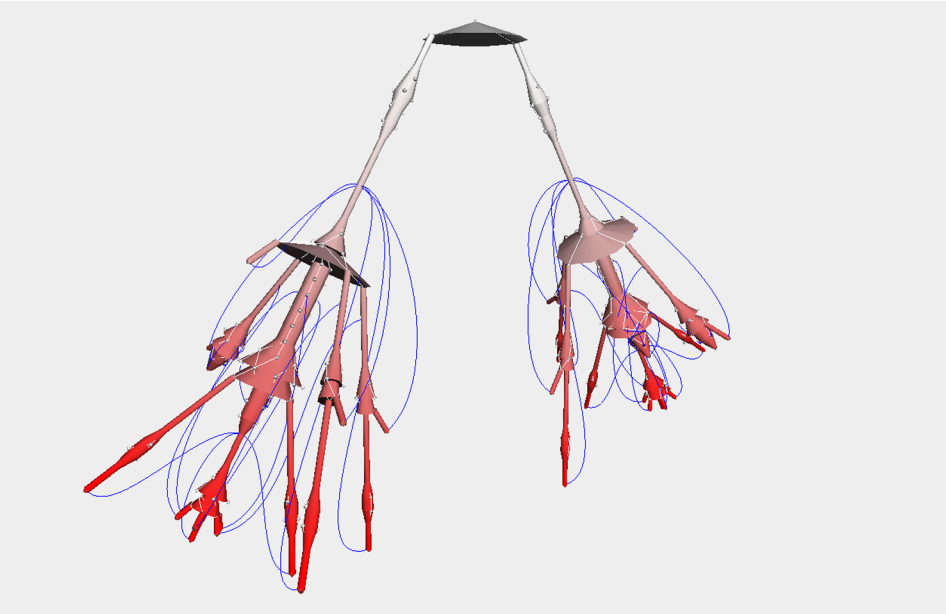
Figure 2.4: Two-stage reduction of Figure 2.2 [44].

on this, the technique seems to meet both requirements introduced above. Unfortunately, it suffers from a lack of scalability. As Jéron and Jard noted, readable results can only be guaranteed for graphs with fewer than a hundred nodes. This makes their approach unsuitable for the large transition graphs that system analysts typically study.

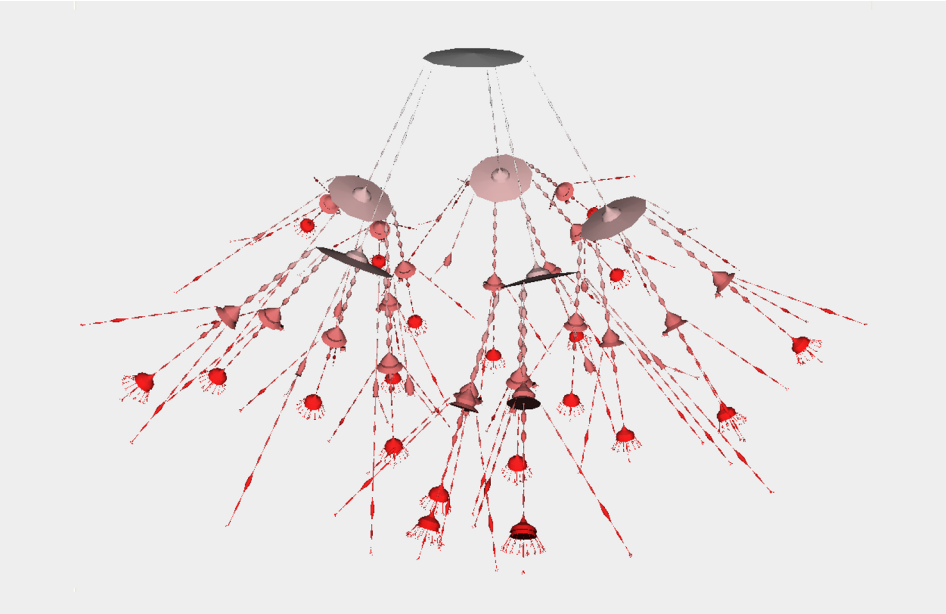
With all techniques discussed so far, every individual node in the transition graph is visualized. Leuschel and Turner [44] developed a technique to reduce the complexity of transition graphs by merging nodes. Their two-part strategy first combines nodes based on the labels of incoming edges and then merges nodes that have outgoing edges with identical labels. Figure 2.4(a) and Figure 2.4(b) show the results of this strategy on the transition graph in Figure 2.2. The reduced graph is visualized with general graph drawing techniques such as those discussed at the start of this section.

To justify their approach, Leuschel and Turner emphasized the large reduction in size of the final transition graph compared to the original (compare Figure 2.4(b) with Figure 2.2, for instance). Leuschel and Turner showed that for a set of 47 state transition graphs the approach produced an average reduction of 27.7% in the number of nodes and 22.2% in the number of edges. For 80% of the graphs the number of nodes was reduced by 56.6% and the number of edges was reduced by 40.6%. These numbers suggest that the technique scales reasonably well.

It is difficult to judge, however, how well the resulting graphs meet the two aims introduced above. This is because the semantic implications of the proposed reductions are not entirely clear. The reduction techniques are rather ad hoc and do not appear to be based on a sound theory of the operational semantics of state transition graphs. As a



(a) Two jacks.



(b) Five jacks.

Figure 2.5: Behavior of a modular hydraulic jack system [25].

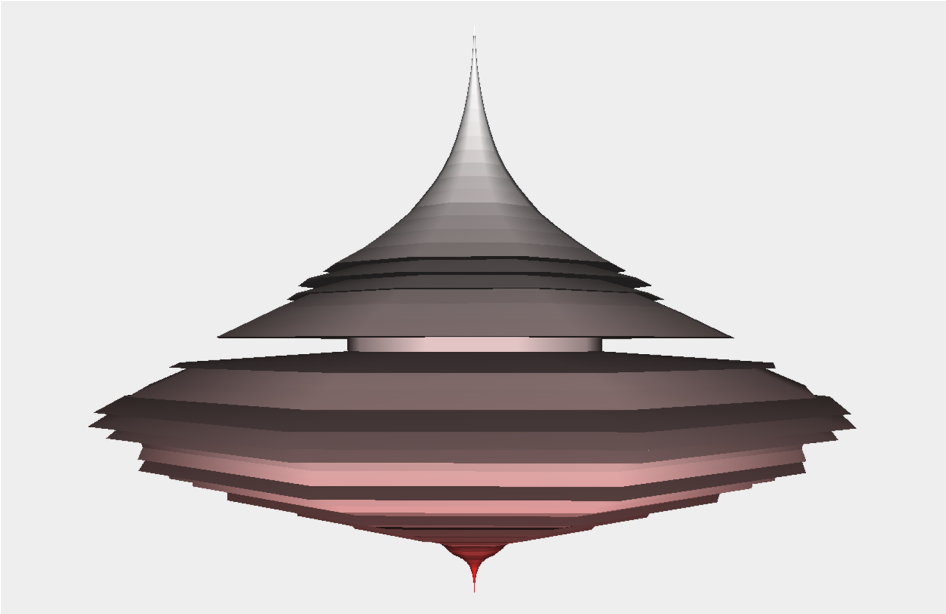


Figure 2.6: No symmetrical phases are found for graphs with highly connected nodes [25].

result, it is not easy to say what the resulting abstract graphs represent in terms of system behavior.

This could be addressed by enabling the user to interactively steer the clustering process. Another alternative, and one that is often taken by system analysts, as noted in Section 2.2, is to use reduction algorithms based on rigorous computational theory. These include techniques such as isomorphy, language equivalence, trace equivalence, and bisimulation equivalence [4].

Van Ham et al. [25] developed another visualization technique for state transition graphs based on clustering nodes. Their approach was conceived to emphasize global structural symmetries. Preprocessing, including ranking and clustering of nodes, generates a structural backbone on which nodes and edges are positioned. This results in layers and branches which highlight phases of behavior.

To validate their approach Van Ham et al. studied the behavior of real-world systems. For example, the behavior of a modular hydraulic jack system, where a variable number of identical jacks can be combined to function in unison, was analyzed. Figure 2.5(a) is a visualization of the transition graph of this system configured with two jacks. Figure 2.5(b) visualizes its behavior when it is configured with five jacks. These figures illustrate that despite the differences in complexity, the behavior of these two different setups are very similar. Starting from the top, where the initial node is positioned, both visualizations show an initialization phase, after which they respectively split into two and five symmetrical branches.

The above approach scales well. For instance, Figure 2.5(b) shows a graph with 70,926 nodes and 145,915 edges. Using the technique it is possible to identify interesting aspects of transition graphs such as the identical symmetrical phases of behavior in Figure 2.5(a) and Figure 2.5(b). However, once these have been identified, it is difficult to study them further in terms of node attributes and edges labels. Also, as shown in Figure 2.6, for systems that contain little parallelism, or where nodes are highly connected, the backbone often contains no symmetrical branches. Since symmetries are generally taken as departure points for analysis, these cases are problematic.

2.4 The case for visualization

Visualization techniques for analyzing state transition graphs have been enthusiastically received by analysts. For example, Van Ham et al. [25] collaborated with system analysts to analyze a number of real-world data sets. Using their visualization technique, many discoveries about these systems' behavior were made.

But why does visualization work? There are three main arguments. The first emphasizes the cognitive advantages of visualization and the second focuses on the power of the human perceptual system. A third factor that plays an important role in computer-based visualization is involving the user in the exploration experience by means of interaction.

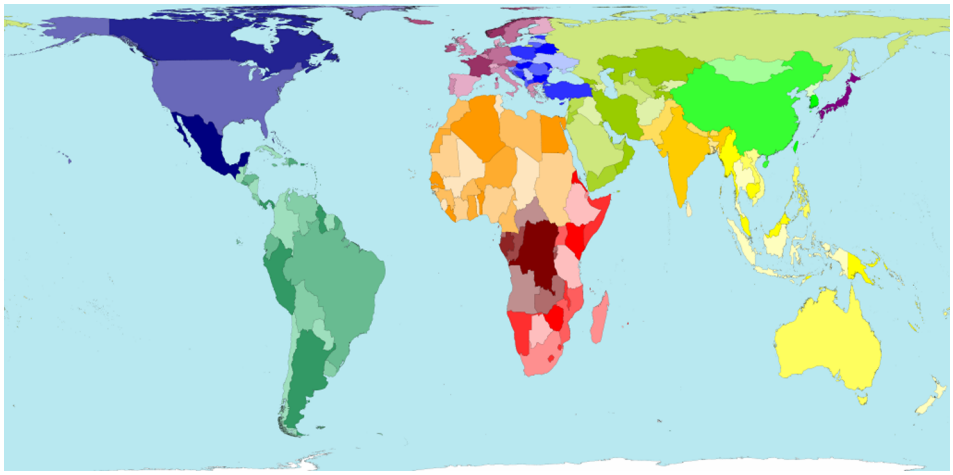
2.4.1 External cognition

The effectiveness of visual representations to aid human cognition is often attributed to the principle of externalization. This school of thought argues that the act of representing problems in the physical world, outside the mind, greatly enhances problem solving. This is supported by a body of results from cognitive psychology [51].

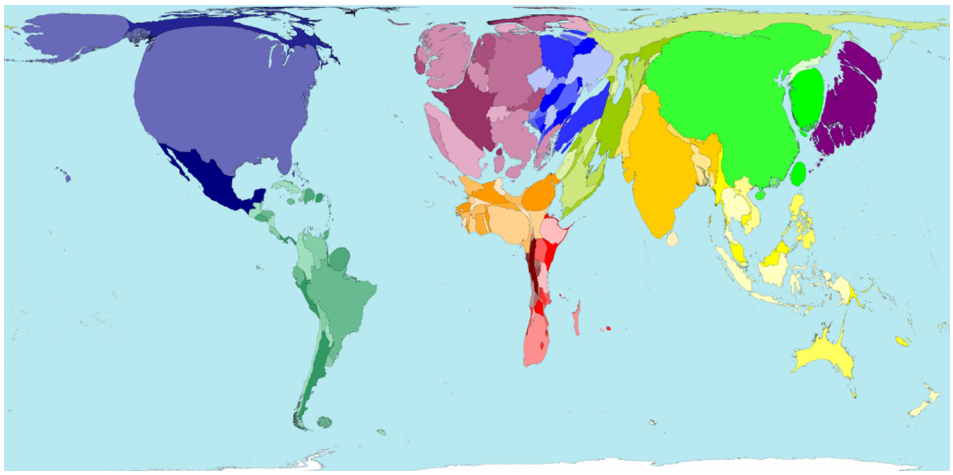
External cognition refers to the way in which the external world aids thought and reasoning [68]. According to Norman [53], the interweaving of internal mental action, "knowledge in the head," and external perception and manipulation, "knowledge in the world," is the essence of how expanded intelligence is achieved. Scaife and Rogers [68] characterize external cognition in three ways:

- *Computational offloading.* External visual representations reduce the degree of effort needed to solve problems. As an example, Card et al. [12] contrast mental and paper-based multiplication. Multiplying a pair of two-digit numbers using pen and paper is generally much faster than performing the calculation purely mentally. By recording and looking up partial results, the problem is split into smaller computations without the need to store intermediate values in the mind.

- *Re-representation.* Different external representations of the same data and with the same structure can enhance or inhibit problem solving relative to each other. Zhang and Norman [99] compare the level of difficulty of carrying out multiplication using Roman and Arabic numerals. Although the same numerical structure and content is represented by both systems, it is much harder to find the result of LXVIII $\times$ X than 68×10.



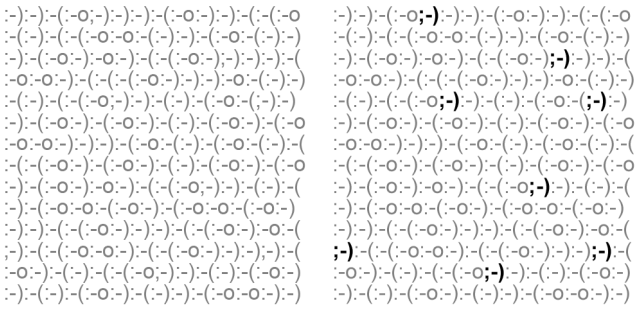
(a) World map, area encodes geographic area.



(b) Area encodes ecological footprint.

Figure 2.7: Ecological footprint of the world [17].

- *Graphical constraining.* Graphical elements have the ability to lead to specific types of interpretations about the underlying data. For example, when a map of the world (see Figure 2.7(a)) is distorted so that the area of a territory corresponds to its ecological footprint (Figure 2.7(b)), it is immediately clear that the northern hemisphere, particularly North America, Western Europe and Eastern Asia, has a far more substantial impact on the environment than the southern hemisphere [17].



(a) How many winking smileys? (b) With visual cues.

Figure 2.8: Pre-attentive processing.

2.4.2 Perception

Humans have evolved to visually perceive the physical world [87]. Consequently, people's main source of information input is their sight and more information reaches the mind through vision than through all the other senses combined. Ware [87] presents the following, much simplified, two-stage model of the process of visual perception:

1. *Extract basic features.* When light enters the eye, it triggers 125 million photosensitive neurons, or nerve cells, on the retina at the back of the eye [65]. These convert light into a pattern of nerve impulses which are transmitted to the brain via the optic nerve. There are subsets of neurons in the eye that work in parallel to extract particular features such as the orientation, color, texture and movement of perceived visual elements. Extraction of these features is largely involuntary and very fast.

2. *Examine the environment.* Next, the signals that have entered the brain are split to be processed by two specialized subsystems. The first recognizes objects in the environment by also involving memory. The second subsystem guides interaction with the external environment. These two subsystems process perceived objects sequentially, one at a time. As a result, the second step in perceptual processing is much slower than the first.

It is possible to exploit the properties of the perceptual system by visually representing data in such a way that some characteristics are perceived without conscious effort. For example, in Figure 2.8(a) it is much more difficult to count the number of occurrences of the character sequence ";-)" compared to Figure 2.8(b). This is because color and size, with which Figure 2.8(b) has been annotated, are pre-attentively processed. Other pre-attentive cues include orientation, shape and motion.

2.4.3 User interaction

The arguments provided in Section 2.4.1 and Section 2.4.2 only scratch the surface in terms of explaining why vision is such a powerful sense. Although a more thorough analysis of cognition and perception is beyond the scope of this dissertation, the cited examples illustrate that visual representations can significantly reduce the burden of a number of data-related tasks.

Visual representations communicate meaning to the user. To optimize this relationship, however, a feedback loop from the user to the visualization is also required [87]. This role is fulfilled by user interaction. By enabling users to steer their inquiry, through interaction in real time, they move from being spectators to being participants in discovery and reasoning [80].

From both a cognitive and a perceptual perspective (Section 2.4.1 and Section 2.4.2), user interaction plays an important role in visualization. In fact, Tweedie [83] argues that the effectiveness of visualization lies precisely in the coupling of presentation with interactivity. As an example, Card et al. [12] describe how a maritime navigation chart serves both as a computation aid, by facilitating the calculation of a ship's heading, and as external memory, by enabling navigators to record information on it. Hence, a significant part of the value of such a chart can be ascribed to the ability of users to interact with it for calculation and record keeping purposes. The second step of the model of perception introduced in Section 2.4.2 also emphasizes the role of the perceptual system to guide interaction with the environment.

2.5 Information visualization

Visualization research combines visual representations of data with the ability to interact with them [12]. The rapid development of computer graphics and interaction technologies have served as the enabling driving force for the field [19]. Contemporary consumer-grade computers can render millions of graphical primitives to screen and, based on user input, can update these in real time.

Taking into account the cognitive benefits of graphical representations, the characteristics of the human perceptual system and the ability to interact with visual depictions, visualization offers users the following advantages [12, 78, 87]:

- Comprehension of very large amounts of data.
- Reduced search time.
- Understanding of both large- and small-scale features of data.
- Formation of hypotheses.
- Identification of unanticipated properties and patterns in data.
- Identification of problems with data sets.
- Direct manipulation of data.

When computer-aided visualization first became possible, visualization techniques were almost exclusively applied in the physical science and engineering domains [15]. Visualization of data sets representing physical attributes, such as volume renderings of the human body for medical applications, were developed. As a result, the term scientific visualization has become synonymous with the visualization of data sets that have natural spatial representations.

It was soon realized that data for which no spatial representation exists, such as software source code [6], could also be visually analyzed. Consequently, information visualization, a new research discipline that investigates techniques to visualize abstract discrete data, was established during the early 1990s. Due to the lack of natural physical representations, finding appropriate mappings from data to visual representations is a key challenge of the field. Card et al. [12] define information visualization as follows:

Information visualization. The use of computer-supported, interactive visual representations of abstract data to amplify cognition.

Since 1995, when the first IEEE Symposium on Information Visualization was organized, the information visualization design space has been extensively explored and many presentation techniques have been developed. For example, many graph visualization techniques have been proposed.

With most graph visualization techniques an edge-centric approach is taken. For example, many methods have been developed to minimize edge crossings or to position nodes in close proximity when they share a large number of edges [31]. These methods consider nodes and edges as flat entities that only define the structure of the graph.

As is the case with state transition graphs, however, the nodes and edges of many graphs are multivariate. That is, they have additional associated data (see Section 2.1). For example, in social networks actors are represented by nodes. These nodes are described by node attributes such as name, gender and age. Below, common approaches for visualizing multivariate data, in general, are first considered. Next, a few techniques developed specifically for visualizing multivariate graphs are discussed.

2.5.1 Multivariate visualization

A number of successful strategies for visualizing 2D and 3D data have been put forward by the information visualization and statistics research communities. However, the visual representation of data with more than three dimensions remains a challenge.

A popular approach is to use low-dimensional projections onto 2D or 3D space. In the latter case, 3D scenes are usually rendered on a 2D monitor, giving rise to a number of issues. These include occlusion [74] and several problems related to the limitations of human spatial perception on a 2D display medium [88]. As a result, 2D projections are almost always more appropriate and are widely used.

Several researchers have argued that users should be directly involved in choosing an adequate 2D projection by considering the different dimensions of a dataset individually and with respect to others. With Seo and Shneiderman's rank-by-feature framework [69], the user first selects a ranking or ordering criterion (see Figure 2.9(a)). This is a metric

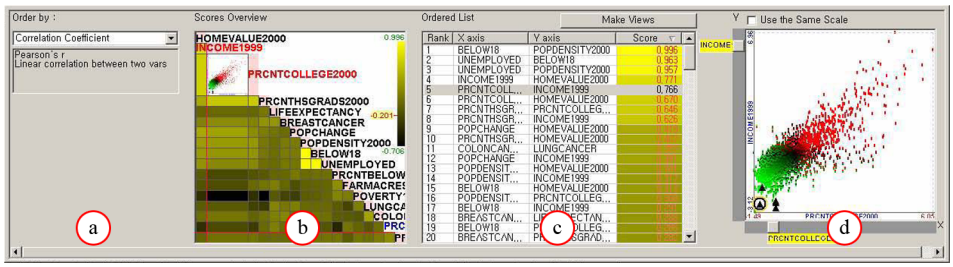


Figure 2.9: Rank-by-feature framework [69].

that is defined for a single dimension or for pairs of dimensions found in a dataset. In the case of ranking pairs, the more complex of the two types of ranking criteria, the user is consequently presented with a rank-by-feature prism (Figure 2.9(b)) and an ordered list of pairs of data items (Figure 2.9(c)).

The rank-by-feature prism provides an overview of scores generated by the selected metric for all pairs of dimensions. This is visualized as a color-coded matrix. In the ordered list, pairs of dimensions are linearly ordered and color-coded based on their scores. These two representations serve as selection devices for a third visualization: when the user selects a pair of dimensions in the rank-by-feature prism or the ordered list of dimensions, the multivariate data items are projected to a scatterplot of which the axes are defined by the two selected dimensions (Figure 2.9(d)).

The framework supports users in identifying pairs of interesting dimensions. For instance, they may be interested in finding combinations of dimensions that generate very low or very high values for a particular metric. Once such outliers have been found, users can study individual data items with respect to the dimensions in question.

Visual Hierarchical Dimension Reduction (VHDR) is another strategy for visualizing multivariate data. This technique, proposed by Yang et al. [98], uses a similarity measure to construct a clustering hierarchy with which users can interact. The original dimensions found in a dataset form the leaves of this hierarchy and the most similar pairs are grouped into clusters. Likewise, pairs of similar clusters are combined to form higher level clusters. The user explores this hierarchy to identify clusters that represent interesting combinations of dimensions. When a cluster is selected, the dataset is visualized in 2D by only considering these dimensions.

Yang et al.'s strategy does not impose a particular visualization technique on users, who are free to choose from traditional multivariate visualization techniques such as parallel coordinates [36] or scatterplot matrices [13]. In this way, the clustering hierarchy serves as a device to navigate through visualizations of high-dimensional data.

The above strategies for visualizing multivariate data emphasize the role of the user in finding suitable low-dimensional projections of data items. This is in contrast to automated techniques, such as Asimov's grand tour [5], which animate through the infinitely many possible low-dimensional projections. It is argued that by involving users directly they have a better understanding of the results they are presented with.

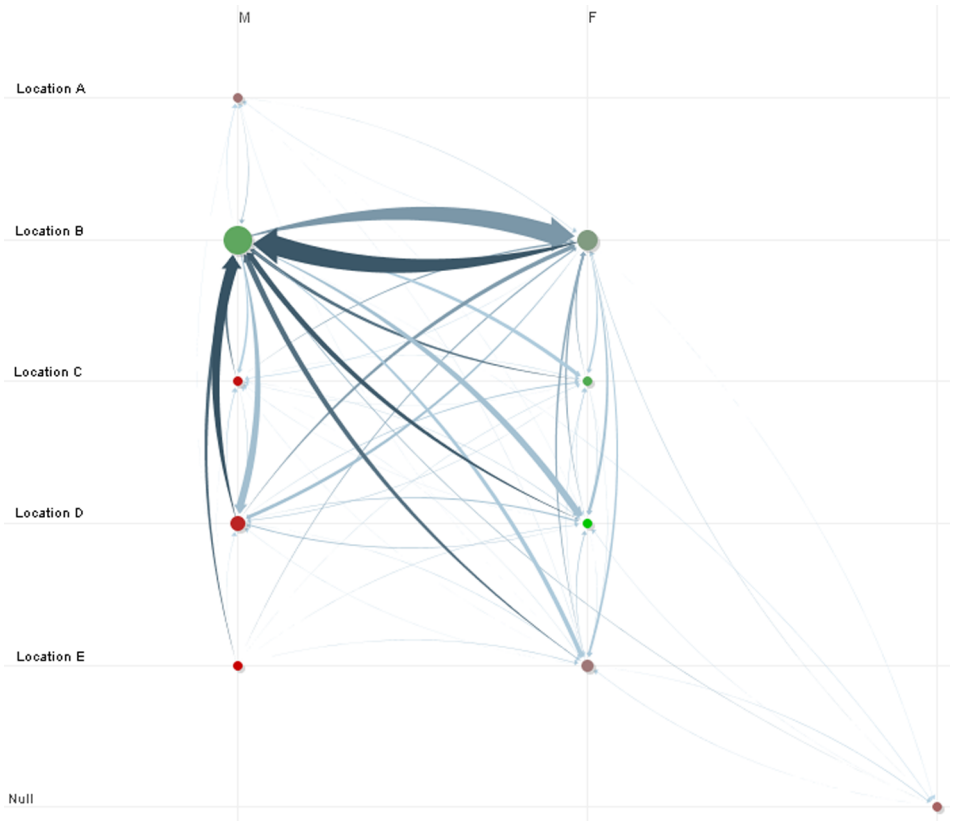


Figure 2.10: Communication graph of a company [91].

2.5.2 Multivariate graph visualization

Multivariate graphs have data associated with their nodes and edges. A few techniques have been developed to visualize such graphs as a function of the multivariate data associated with their nodes.

Wattenberg [91] developed a technique that enables the user to generate a summary graph by selecting a maximum of two node attributes at a time. The graph is rolled up by aggregating all nodes that have the same values for these attributes. Edges are contracted accordingly. The aggregate node clusters are positioned on a 2D grid where the x- and y-axis represent the two user-defined dimensions. The number of nodes aggregated on a particular coordinate and the number of edges that span between clusters are encoded with size and color intensity.

Since users know what node attributes mean, results are easy to interpret. Figure 2.10 shows a summary graph for a network that represents the communication habits of employees in a large company [91]. The graph has been rolled up on gender (x-axis) and office location (y-axis). The visualization shows that most communication is toward

males in location B. It also shows that within location B, male and female colleagues tend to communicate much more than at other locations. The cluster at the bottom right of Figure 2.10 highlights another interesting fact about the data set; not all actors are well-defined in terms of gender and location.

Although the technique proposed by Wattenberg enables users to substantially reduce the complexity of a graph, only two node attributes can be considered at a time. Users are able to rapidly switch between different pairs of attributes, however. The transitions between different clusterings are animated to help the user maintain context [91].

Shneiderman and Aris [75] proposed a technique that positions multivariate nodes in non-overlapping regions defined in terms of node attributes. Inside such a semantic substrate, nodes are positioned by existing techniques (for example, force-directed layouts or temporal strategies). Since users typically understand the meaning of node attributes, this approach enables them to interpret node positions in terms of their existing knowledge.

With this technique special attention is paid to interaction. User-steered filtering can be used to reduce edge clutter. For example, users can specify that only edges between certain regions should be shown. Dynamic query sliders are also provided for specifying a neighborhood around user selected nodes for which outgoing or incoming edges are shown. In this way, relations between node attributes and edges can be analyzed.

In Figure 2.11, a graph that represents legal precedents is visualized [75]. In this graph, a node represents a judicial hearing and an edge denotes a citation of another hearing. In the figure, there are three regions that represent supreme court, circuit court and district court hearings. Within every region, hearings are ordered by date.

In the figure, the user has selected a number of circuit court hearings in 1995 and 1996. Consequently, arrows that represent all hearings that were referenced by these cases are displayed. The current selection suggests that circuit court cases are more likely to cite other circuit court cases, as opposed to those heard by the supreme or district courts. This is seen by observing that while many arrows point to cases within the region representing circuit courts, only one arrow crosses from it to the region representing district courts.

One drawback of the approach proposed by Shneiderman and Aris is that screen real-estate is not used very effectively. Their strategy of generating a region per node attribute soon runs out of space, even for a small subset of attributes. The technique has been shown to work well for graphs with a few thousand nodes, but its ability to scale to larger graphs may pose a problem. By combining nodes in clusters, as Wattenberg does [91], this issue may be addressed.

A technique developed by Archambault et al. [3] gives edges a more prominent role by emphasizing the topology of the abstract graph resulting from attribute-based clustering. After the user has selected one or more attributes, nodes where these attributes assume the same values are grouped into a hierarchy of clusters. A topological layout algorithm is then used to calculate the positioning of the resulting abstract graph.

Selection and pattern matching on node attributes enable the user to interactively coarsen or refine clusters in a way that leaves topological features intact. For example, Figure 2.12 provides an example of using Archambault's technique to analyze a co-author graph. In this graph, nodes represent research articles and edges represent shared authors.

Circular regions represent clusters and rectangular icons represent articles. The nesting of icons and regions within other regions represents the hierarchical clustering of the

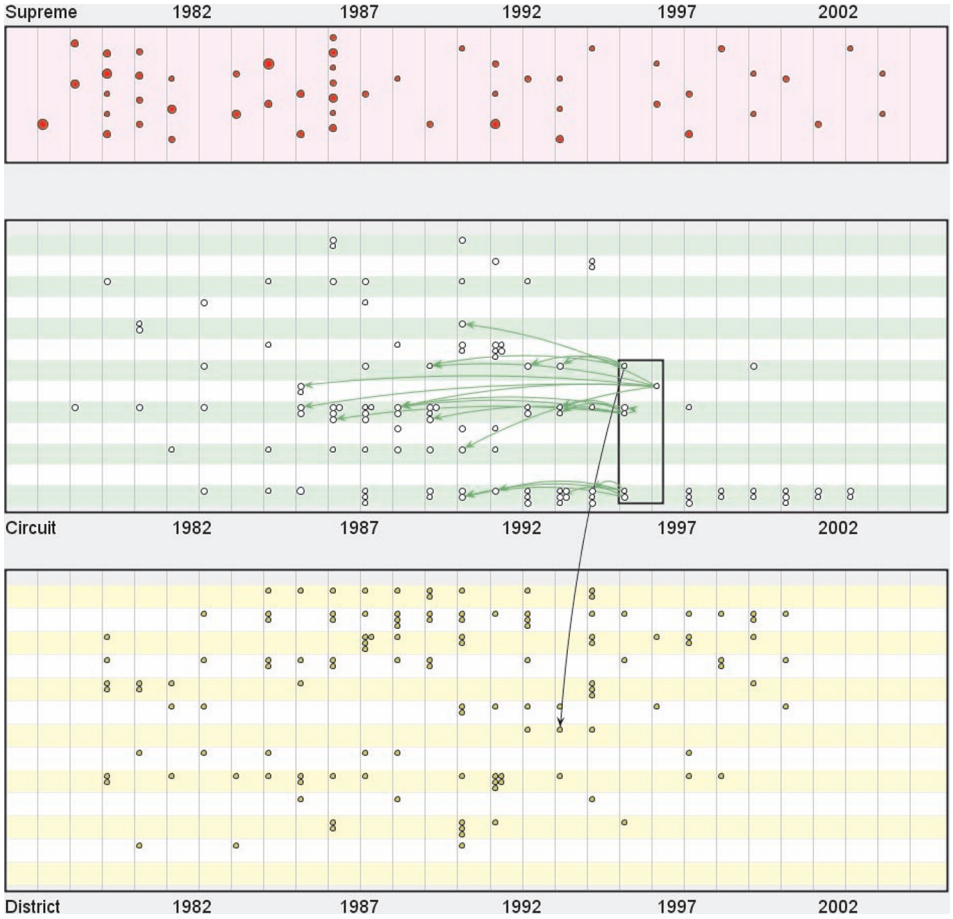


Figure 2.11: Court case reference graph [75].

graph. For example, the large gray region represents the entire collection of articles. This collection has been partitioned into a number of clusters represented by five yellow and one pink region. The cluster at the top has been partitioned further into four clusters represented by three blue and one green region.

In Figure 2.12 the entire set of articles has been clustered based on the results of the search queries "Stuart K. Card" and "Stuart Card". All Stuart K. Card's articles appear inside the cluster represented by the pink region at the center of the visualization. Furthermore, a small cluster of articles authored by Stuart Card, without the middle initial, are shown inside a cluster represented by a green disk toward the top right.

A disadvantage of using topological layout algorithms is that the positions of clusters shift as users adjust the hierarchy. For example, users may want to determine whether Stuart K. Card and Stuart Card is really the same person. To do so, they are likely to perform further clustering, for instance, to identify co-authors. When this is done the

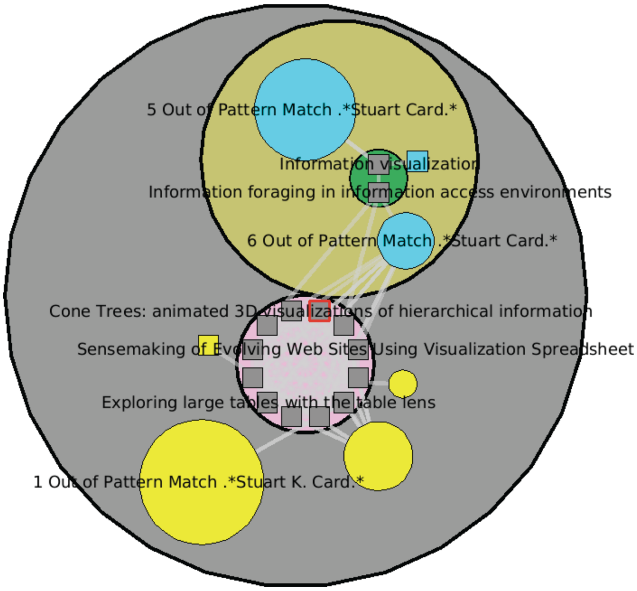


Figure 2.12: Co-author graph clustered on author name [3].

layout will change, and the positions of clusters relative to each other will be different from those in Figure 2.12. This makes it harder to relate the positions of clusters to the semantics associated with node attributes.

2.6 Methodology

In following chapters a number of techniques for visualizing state transition graphs are presented. Typical for the field of information visualization, a user-centered approach was taken toward their development. Many different solutions were investigated and evaluated by closely involving target users.

2.6.1 Target users

The target users of the visualization techniques presented in this dissertation are system analysts who use formal methods for understanding system behavior. The author closely collaborated with a group of system analysts at the departments of Computer Science and Mechanical Engineering at Eindhoven University of Technology. There was also limited contact with system analysts at CWI, the Dutch Institute for Mathematics and Computer Science.

System analysts are knowledgeable domain experts and they regularly use state transition graphs to model and study system behavior (see Section 2.1and Section 2.2). Due to the expertise required to perform this type of analysis, the target user group was rela-

tively small, varying in size between seven to ten users. As a consequence of the time and effort needed to gain this knowledge, only one or two users typically study the behavior represented by a specific transition graph.

When system analysts were approached, they accepted that visualization could play an important role in their work. Nevertheless, it was very difficult for them to formulate what this should be. Furthermore, due to the varying nature of their work it was not possible to outline general questions or a set of routine tasks to support with visualization techniques. As is often the case with visualization research, the insight being sought by users was hard to define. North [54] argues that this results from the characteristics of what it means to gain insight:

- *Complex.* To gain insight, large and complex data sets have to be understood in a synergistic way. Once one property of the data is understood, this will influence further insights. Based on these, the original insight may have to be reconsidered.
- *Deep.* Time is needed to accumulate layers of insight that build on each other. Insight gained in this fashion is likely to raise new questions, of which users are unaware when they start their analysis [58].
- *Qualitative.* Insight cannot be exactly measured since it is uncertain and subjective.
- *Unexpected.* Insight is unpredictable. Great discoveries are rare and cannot be guaranteed [58].
- *Relevant.* Gaining insight into data is highly dependent on existing domain knowledge. The goal of users is relevant domain impact and not just dry data analysis.

2.6.2 Evaluation

To deal with unclear requirements and the issues mentioned above, an experimental and exploratory approach was adopted. Prototypes were developed, refined and extended in several iterative rounds, over an extended period of time, finishing each with an evaluation.

Plaisant [58] identifies three main approaches toward the evaluation of information visualization techniques. These include controlled experiments, usability evaluations and case studies. In a controlled experiment a novel visualization system is compared with existing approaches to determine whether it performs better. However, as explained, the work presented in this dissertation is exploratory and, as is shown in following chapters, it differs significantly from the few existing techniques for visualizing state transition graphs. This makes direct comparisons unsuitable.

Usability evaluation provides feedback on the problems encountered while users interact with a system. Users are judged on the accuracy or efficiency of completing certain tasks [71]. Usability evaluation is inappropriate for the work presented in this dissertation for two reasons. First, as noted above, defining low-level requirements in terms of tasks to support was not possible. Second, the user population considered in this dissertation was too small for formal statistical analysis.

Longitudinal case studies make it possible to consider the feasibility of visualization techniques by involving real users performing real data analysis in their natural work environment [58]. Such studies are becoming increasingly popular in visualization research [76]. They enable visualization designers to cope with unclear requirements and help them to focus on adding value for users performing their actual work. Furthermore, longitudinal studies address the difficulty of accurately defining what insight is in the context of a specific application domain. Longitudinal evaluation has the following characteristics:

- *Multi-dimensional.* Different evaluation techniques can be used to assess user performance and the utility of visualization techniques.
- *In-depth.* There is intense engagement, collaboration and partnership of the designer/evaluator with expert users.
- *Long-term.* Users are involved over an extended period of time. After new techniques are introduced to users, this is followed up with studies that investigate the influence of these techniques on the way in which users study and approach their data.
- *Case studies.* A small number of users are considered as they work on real problems, preferably in their work environment.

To evaluate the suitability of the techniques for visualizing state transition graphs presented in following chapters, a longitudinal approach was taken. This included case studies, semi-structured interviews and informal user evaluations. These resulted from close collaboration with system analysts over a period of three-and-a-half years. In this fashion, user feedback on any particular visualization technique also influenced subsequent work.

Chapter 3

Selection and Projection

It is natural to consider the visual representation of state transition graphs as a graph visualization problem. Since every node is also a point in n-space, however, state transition graphs also pose an interesting multivariate visualization challenge. In this chapter, by focusing on the dimensionality of their nodes, a number of approaches for projecting transition graphs to the 2D plane are examined. To do so, existing techniques for visualizing multivariate data are drawn on, extended and refined to suit the task at hand. To assess the techniques that are presented, a case study is analyzed. Also, the ability of the techniques to help users address questions that are difficult to answer with conventional system analysis methods are considered.

3.1 Rationale

An intuitive approach for visualizing state transition graphs is to focus on their topology. By taking such a view, nodes and edges are considered as flat entities that define a graph's structure. As noted in Section 2.3, system analysts have traditionally taken such an approach by using general purpose graph drawing tools to visualize transition graphs. Even techniques that are tailored for transition graph visualization, such as those proposed by Jéron and Jard [37], Leuschel and Turner [44] and Van Ham et al. [25], focus on graph topology to generate visual representations.

The above mentioned techniques consider the visualization of state transition graphs as a pure graph or network visualization problem. However, it is also a multivariate visualization challenge.

Consider the definition of a state transition graph given in Section 2.1. Every node $v = (v_1, ..., v_n)$ in a transition graph consists of a vector of attribute values in n-space, where v_i is a valuation of the i^{th} node attribute. The values that a particular attribute can assume are restricted to D_i, typically a rather small discrete set. Put differently, the cardinality $|D_i|$ of the i^{th} dimension is typically quite low, in the order of 2–20. Furthermore, the dimensionality n of a transition graph is usually higher, in the order of 10–40.

3.2 Challenge

The high dimensionality of most state transition graphs reflects their complexity. State transition graphs are finite state machines that model the behavior of complex systems. The sequence of values that node attributes assume in such a graph represents this behavior. Still, using conventional system analysis methods like those discussed in Section 2.2, it is often hard to gain insight into the behavior reflected by attributes. A number of questions, all related to node attributes, are difficult to answer:

- *Question 1.* Are there particular subsets of node attributes that exhibit interesting behavior? Conversely, which node attributes are not interesting and may be disregarded?
- *Question 2.* For a specific subset of node attributes, in what way does the system modeled by the transition graph behave?
- *Question 3.* Are there interesting correlations between subsets of node attributes? For example, are certain node attributes dependent on others? Or, to what extent do the values of particular attributes influence the behavior of other node attributes? In particular, it is often difficult to detect correlations between more than two attributes.
- *Question 4.* Are there close correlations between certain actions, represented by edge labels, and node attributes? For instance, are the values that particular node attributes assume dependent on particular edge labels?
- *Question 5.* For a subset of node attributes, are there nodes that assume a particular combination of values? Also, do edges exist from such nodes to others?

The ability to answer these questions enables system analysts to verify functional requirements that have been stipulated for a system. The questions are also useful for verifying that the algebraic specifications from which transition graphs are generated are correct. That is, to confirm that the behavior that these specifications are intended to describe match that of the actual resulting behavioral model (see Section 2.2). Finally, these questions are of interest for gaining insight into systems about which relatively little is known.

Although it is possible to come up with answers to some of these questions using algebraic proof techniques, this requires significant effort and is prone to human error (see Section 2.2). Motivated by these challenges and inspired by the related work discussed in Chapter 2, a number of approaches for visualizing state transition graphs from a multivariate perspective are considered below.

3.3 Analysis of dimensions

As shown in Section 2.5.1, a common approach for reducing the complexity of a multivariate data set is to visualize the data only with respect to a subset of the original dimensions. For system analysis, users are likely to be interested in different subsets of

node attributes at different times. For instance, they may want to consider specific attributes that allow them to verify whether certain functional requirements are satisfied. Alternatively, they may simply be interested in identifying variables that exhibit expected or unexpected behavior and to consider the state transition graph with respect to these.

Existing multivariate visualization techniques, such as those discussed in Section 2.5.1, suggest that in order to identify and select a subset of dimensions, the user must be provided with a good overview. This overview should make it possible to interactively compare dimensions for similarities, differences and other interesting behavior such as outliers or alternating patterns [74].

Both approaches discussed in Section 2.5.1 preprocess multivariate data by using metrics and similarity measures. However, despite the vast amount of theoretical results underpinning the field of formal system analysis, such metrics have not been defined. Also, system analysts are typically not only interested in identifying similar node attributes. In fact, sometimes the differences between attributes may be important. For this reason a technique was developed to enable users to compare dimensions themselves.

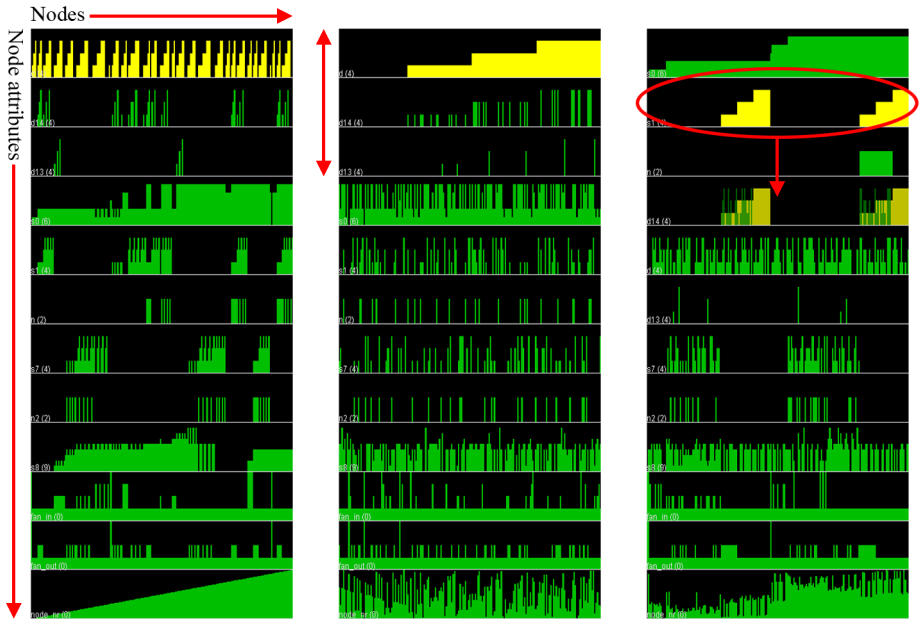
The point of departure is to consider the m nodes of dimension n in a transition graph as n arrays of length m. These n arrays correspond to the node attributes and are visualized as n parallel histograms (see Figure 3.1(a)). Nodes are distributed along the x-axis, typically according to a breadth-first traversal from the initial node of the corresponding state transition graph. To enable the user to identify patterns, the histograms are normalized, making them scale invariant [2].

Users can discover interesting node attributes, or dimensions, by visually identifying patterns, correlations, gaps and outliers in the data set. Histograms are also labeled, enabling them to identify specific node attributes. Individual histograms may be selected, deselected, dragged to and dropped in new vertical positions.

As is shown in Section 3.5, the vertical ordering of dimensions has a direct influence on the positions to which nodes are rendered in 2D projections of the graph. Deselected dimensions have no influence. In this way, the parallel histograms serve as a filtering and zooming device and facilitate interactive dimension reduction. The use of parallel histograms combined with an approach to view the resulting low-dimensional projections also adheres closely to Shneiderman's mantra [74], "overview first, zoom and filter, then details on demand."

Nodes may also be sorted based on a selected attribute. That is, nodes are reordered along the x-axis according to an ascending or descending ordering of the values that this attribute assumes. All histograms are updated to reflect this ordering (see Figure 3.1(b)). This further enables the user to detect interesting correlations or differences in a fashion analogous to Bertin's permutation matrices [9]. When subsequent dimensions are selected and sorted, existing partitions are maintained; nodes are reordered only for intervals where the values of previously sorted variables are equal. This allows for the detection of patterns within patterns.

Parallel histograms resemble the approach taken with Rao and Card's table lens [64]. However, it has been extended with a feature that makes it particularly easy to visually compare different dimensions. When a specific dimension is selected it may be dragged over another, resulting in an overlay of the two (see Figure 3.1(c)). Whereas most automated techniques perform similarity analysis [2], such a visual comparison also makes



(a) Unordered. (b) Sorted. (c) Comparison of attributes.

Figure 3.1: Parallel histograms (Alternating Bit Protocol).

it intuitively clear how dimensions differ from each other and allows the user to detect interesting behavior such as alternating patterns.

3.4 Case - Alternating Bit Protocol

The state transition graph visualized in Figure 3.1 was generated from a formal specification of the Alternating Bit Protocol (ABP) [10]. It has 191 nodes, 244 edges and is described by 12 node attributes.

ABP is a communication protocol for data transmission over an unreliable channel using positive and negative acknowledgments. As shown in Figure 3.2, ABP consists of a sender, a receiver, a data channel and an acknowledgment channel. Both the data and the acknowledgment channel can corrupt messages.

The sender reads a data value (message) at its input. This message is extended with a control bit (0 or 1) and is sent over the data channel. The sender repeatedly sends this appended message until it receives an acknowledgment from the receiver over the acknowledgment channel. After a successful transmission, the sender flips its control bit and submits a new message read from its input.

After the receiver reads a message from the data channel, it is assumed that it is able to determine whether this message has been corrupted. If the message is correct, the receiver checks whether the attached control bit matches its internal control bit. If so, the receiver

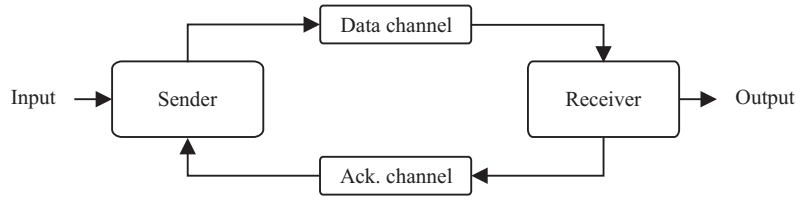


Figure 3.2: Alternating Bit Protocol.

sends an acknowledgment with this control bit to the sender and delivers the message (without the control bit) via the output. The receiver then flips its control bit and waits for another message. In other cases, the receiver sends a negative acknowledgment and waits for retransmission.

The top three histograms in Figure 3.1(b) correspond to the data values of the sender, the data channel and the receiver, respectively. Nodes have been reordered based on an ascending sorting of the first attribute (sender). Except for the default value 0, it is clear that the information passed through the data channel (second histogram) and the data received by the receiver (third histogram) are dependent on the value of the original datum transmitted by the sender (first histogram).

For instance, from the data channel's perspective (second histogram), it receives the correct datum, changes its own value to reflect this, transmits the value and defaults back to 0. Alternatively, if the datum was corrupted, the data channel does not change its value from 0. The above can be seen in the way that the profile of the second histogram resembles that of the first. With such reasoning analysts were able to confirm that the behavior of the above three attributes conforms to what was intended with the original formal specification.

The top three histograms in Figure 3.1(c) show three variables corresponding to the operational modes of the sender, the data channel and the value of the alternating bit for this channel. Initially, analysts were baffled by the fact that the alternating bit assumes a value of 1 for a relatively few number of nodes (third histogram).

After careful visual analysis they were able to deduce that the states where the top attribute assumes the values of 2 and 5 correspond to an alternating bit value of true and false at the sender. They were then able to infer that only in such states is the data channel able to assume a value other than its default (0). This explains the two step-like shapes in the second histogram for nodes where the top histogram assumes the value of 2 or 5.

Furthermore, analysts were able to deduce that the value of the alternating bit is only stored by the data channel during two of its modes of operation (corresponding to the values of 1 and 2 in the second histogram). Since no distinction is made between a default value and an alternating bit with the value 0, this explains why the alternating bit takes the value 1 for so few nodes; most of the time it simply assumes its default.

For other systems, analysts were able to identify node attributes that were superfluous. Indeed, it is trivial to visually identify an attribute that never fluctuates. In one such case, by removing a number of such unused variables from the formal specification from which

the transition graph had been generated, it was possible to reduce the size of the graph by more than 30%.

3.5 Projection to 2D

The analysis and selection of dimensions with the parallel histograms mechanism leads to a lower-dimensional dataset that has to be mapped to 2D in order to visualize the transition graph. Let there be p dimensions to consider, where typically p is far less than n. These correspond to the set of visible dimensions, or those that have been selected by the user (see Section 3.3). To visualize the state transition graph, the positioning of nodes is approached from a multivariate perspective by considering this reduced set.

Any node $v = (v_1, ..., v_p)$ is considered as a linear combination

$$v = \sum_{i=1}^{p} v_i \cdot e_i$$

where v_i is the valuation of the i^{th} node attribute (see Section 2.1) and e_i is the i^{th} basis vector. To find the projection v' of v on the 2D plane it is defined as

$$v' = \sum_{i=1}^{p} v_i \cdot e'_i$$

where e'_i is some projection of e_i in 2D. Hence, the challenge is to find suitable e'_i for $i = 1, ..., p$ such that the questions identified in Section 3.2 may be answered. Finding adequate projections e'_i turns out to be quite challenging and strategies for choosing them are discussed below.

3.5.1 Uniform distribution

When projecting multivariate data to 2D, as outlined above, the e'_i corresponding to every dimension can be uniformly distributed in the plane (see Figure 3.3). This is similar to the approach taken with Kandogan's star coordinates [41]. Uniform distribution allows the user to determine the overall spread of the values of node attributes. In particular, if the distribution occurs in the order in which the corresponding histograms are arranged, it offers an interesting way to analyze the influence of different dimensions by reordering and selection.

Kandogan argues that uniform distribution leads to the discovery of clusters of objects that share characteristics. Since there is no guarantee, however, that distinct p-dimensional points map to distinct positions in 2D, it can be difficult to interpret results. Moreover, due to the fact that p dimensions are forced into 2D, it is hard to attach precise semantics to a particular position.

Figure 3.3 shows the result of a uniform distribution of nine dimensions of the ABP transition graph. Edges are rendered as curved line segments. The directions of the edges are encoded in the orientation of these arcs, which are interpreted in a clockwise fashion.

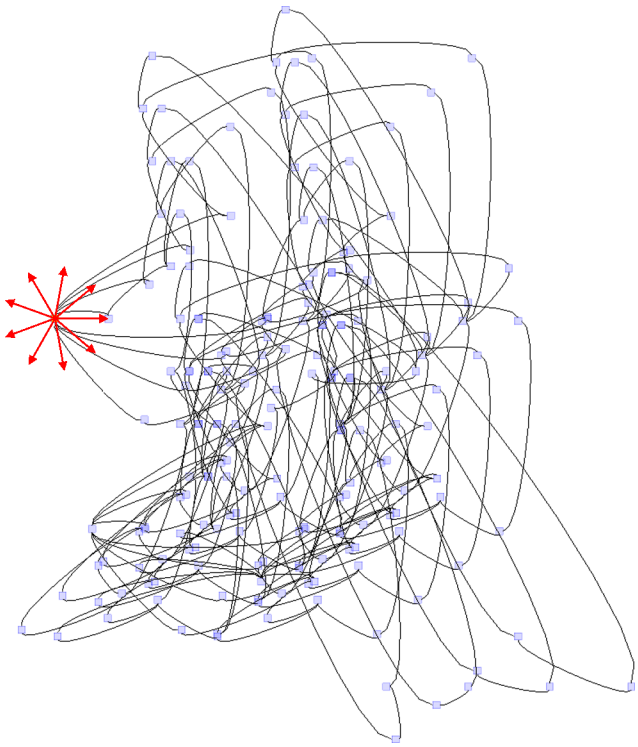


Figure 3.3: Uniform distribution (Alternating Bit Protocol).

Despite the fact that there are some suggestive regular patterns in the visualization, for the reasons cited above, it is difficult to interpret it accurately.

3.5.2 Manual distribution

One line of attack, which addresses the interpretation of the mapping of high-dimensional nodes to 2D, is to enable the user to manually alter the e'_i corresponding to the dimensions. In this way a clearer understanding of the influence of particular node attributes can be obtained. This is done by considering the changes in the projection in real time, as the e'_i are being adjusted by the user. It does not prevent several nodes from being mapped to the same position, however.

An interesting observation is that users often resorted to comparing two dimensions at a time by arranging the e'_i orthogonally in a fashion that resembles a 2D scatterplot. This is similar to the approach taken by Wattenberg [91], which was discussed in Section 2.5.2. Through manual interaction, Figure 3.3 was reduced to Figure 3.4 by deselecting all but two node attributes and by positioning the e'_i orthogonally.

Using this visualization, analysts were able to identify a number of duplicate states,

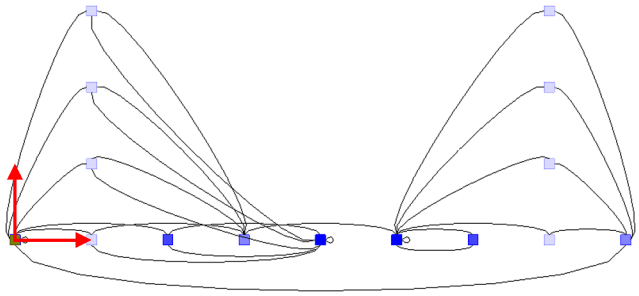


Figure 3.4: Manual distribution (Alternating Bit Protocol).

represented by nodes that are mapped to the center-most cluster. By considering the edges entering and leaving this cluster of nodes, they were able to determine that it completely mimics the behavior represented by the initial node on the far left. The reason for this was that the program that extracted the transition graph from the original specification erroneously duplicated certain system states. Although not related to system behavior, this was an important result.

3.5.3 Hypergrids

A hypergrid generalizes the notion of a 2D scatterplot to n dimensions. Although scatterplot matrices [13] achieve the same objective, they have the disadvantage that detecting relationships that span over more than two dimensions is difficult. To address this issue a hypergrid consists of recursively nested 2D grids: a 2D grid is positioned at every coordinate of a higher level grid. Care is taken to ensure that no two points overlap and that coordinates are uniformly distributed in the x- and y-axis in a fashion that guarantees a minimum interval size.

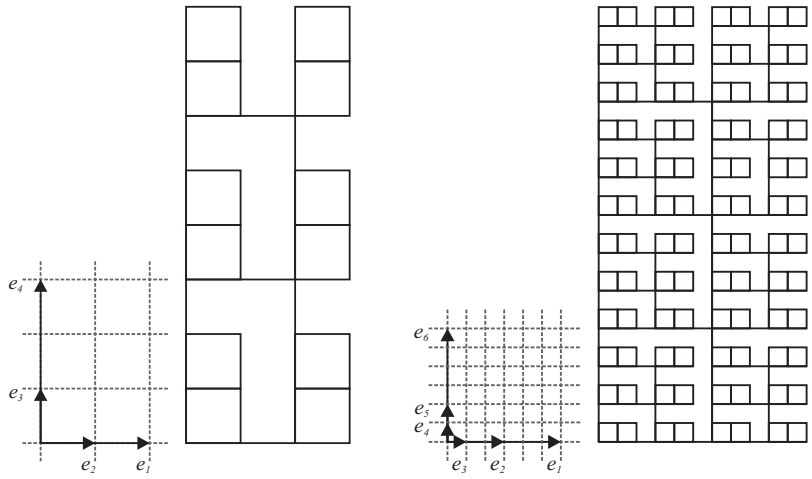
To see how a hypergrid is constructed, first consider the case where the projection vectors e'_i are restricted to being parallel to the x- or y-axis. Now, $e'_i = (x_i, y_i)$ with

$$x_i = \begin{cases} 0 & \text{if } i > \lceil p/2 \rceil \\ \delta_x & \text{if } i = \lceil p/2 \rceil \\ |D_{i+1}| \cdot x_{i+1} & \text{if } i < \lceil p/2 \rceil \end{cases}$$

and

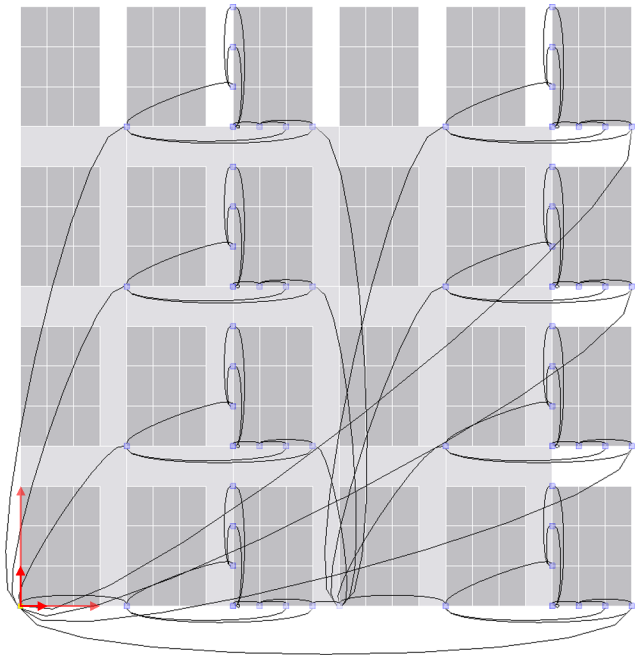
$$y_i = \begin{cases} 0 & \text{if } i < \lceil p/2 \rceil + 1 \\ \delta_y & \text{if } i = \lceil p/2 \rceil + 1 \\ |D_{i-1}| \cdot y_{i-1} & \text{if } i > \lceil p/2 \rceil + 1 \end{cases}$$

where δ_x and δ_y are the minimum permissible horizontal and vertical intervals and $|D_i|$ is the cardinality of the i^{th} dimension, or node attribute, in the subset of p dimensions selected by the user.



(a) Four dimensions.

(b) Six dimensions.



(c) Alternating Bit Protocol.

Figure 3.5: Orthogonal hypergrids.

With this method the coordinate system generated by e'_1 and e'_p takes into account the area needed for nesting the coordinate system that e'_2 and e'_{p-1} generate at each of its coordinates, and so forth. This is achieved by considering the cardinalities, $|D_2|$ and $|D_{p-1}|$, of D_2 and D_{p-1} as well as the area needed for grids that are nested at deeper levels (by considering the cardinalities of their generating dimensions). Also, it ensures a minimum distance in the x- and y-axis between any two successive coordinates. When an odd number of dimensions are visualized, the deepest grid is a one-dimensional line.

Figure 3.5(a) illustrates the hypergrid that gets constructed for $p = 4$ with $|D_1| = 2$, $|D_2| = 2, |D_3| = 3, |D_4| = 4$. Similarly, Figure 3.5(b) illustrates the hypergrid that gets constructed for $p = 6$ with $|D_1| = 2, |D_2| = 2, |D_3| = 3, |D_4| = 2, |D_5| = 3, |D_6| = 4$. Toward the left of the hypergrids are the vectors e'_i from which they are generated. As can be seen, e'_1 and e'_n have a more pronounced influence on the positions of coordinates than e'_2 and e'_{n-1}, and so on.

Figure 3.5(c) shows an orthogonal hypergrid generated from four dimensions of ABP. The outer grid represents the modes of the sender and the data channel on the x- and y-axis. The inner grids encode the receiver and its data values. In the figure the same pattern is repeated twice, once at the left and once at the right. Also, within this pattern there are sub patterns that repeat four times from top to bottom. This indicates that the protocol behaves in a predictable fashion irrespective of the internal modes of its composite parts.

Hypergrids constructed in the fashion discussed above are similar to dimensional stacking proposed by Ward [86]: dimensions all contribute to displacing data items in directions parallel to the x- and y-axis. Next, the idea is refined further to more explicitly represent the influence of different dimensions by rotating successive grids slightly from the previous. It is argued that the dissimilarity in terms of orientation make different dimensions easier to perceive [87]. To achieve this, let $e'_i = (x_i, y_i)$ with

$$x_i = \begin{cases} 0 & \text{if } i = p \\ s(i) \cdot \delta_x & \text{if } i = p-1 \\ s(i) \cdot |D_{i+1}| \cdot |x_{i+1}| & \text{if } i < p-1 \end{cases}$$

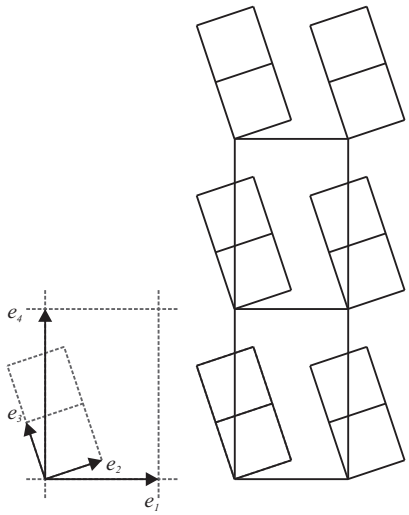
and

$$y_i = \begin{cases} 0 & \text{if } i = 1 \\ \delta_y & \text{if } i = 2 \\ |D_{i-1}| \cdot y_{i-1} & \text{if } i > 2 \end{cases}$$

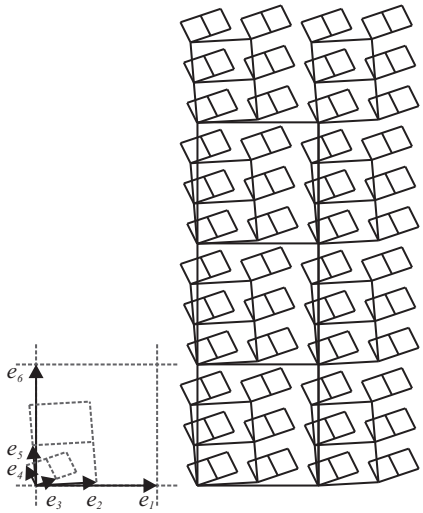
where

$$s(i) = \begin{cases} 1 & \text{if } i \leq \lceil p/2 \rceil \\ -1 & \text{if } i > \lceil p/2 \rceil. \end{cases}$$

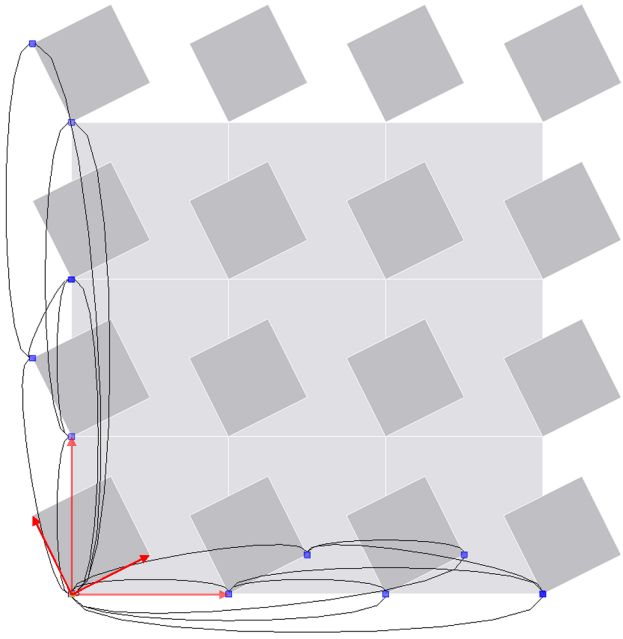
The function s ensures that alternating pairs of projection vectors are more orthogonal in the sense that projection vectors with end points above the line $x = y$ are reflected about the y-axis. Figure 3.6(a) and Figure 3.6(b) show the rotated hypergrids that get generated for the same data set as in Figure 3.5(a) and Figure 3.5(b). That is, for respectively $p = 4$ with $|D_1| = 2, |D_2| = 2, |D_3| = 3, |D_4| = 4$ and $p = 6$ with $|D_1| = 2, |D_2| = 2, |D_3| = 3, |D_4| = 2, |D_5| = 3, |D_6| = 4$.



(a) Four dimensions.



(b) Six dimensions.



(c) Alternating Bit Protocol.

Figure 3.6: Rotated hypergrids.

In Figure 3.6(c) the larger grid plots two attributes that describe the modes of the data and acknowledgment channels in ABP. For the nested grids, the axes correspond to the value of the alternating bit for these two channels. From this visualization, analysts were able to confirm that due to the sequential nature of ABP the two channels operate completely independently. For instance, it is easy to verify that no state exists in which the alternating bit for both channels simultaneously assumes a value other than 0, the default.

Because node attributes in state transition graphs generally have a small cardinality (see Section 3.1), the hypergrid approach may be applied here. In general though, hypergrids do not scale that well as the cardinality of dimensions or the number of dimensions to consider increases. A further drawback is that hypergrids are quite expensive with regard to screen real estate; the area needed to ensure no overlap increases dramatically with every dimension that is included. However, hypergrids enable users to attach precise meaning to the position that a node is projected to. As mentioned, this is not the case with the other techniques discussed here.

3.5.4 Principal component analysis

Principal component analysis (PCA) is one of the oldest and most widely used techniques for multivariate data analysis [39]. PCA reduces the dimensionality of a data set while retaining as much variation as possible.

In light of the current discussion, the aim is to map p dimensions to 2D by identifying the first two principal components. This entails finding the two largest eigenvalues of the covariance matrix of the dimensions and considering the two corresponding eigenvectors [39]. More concretely, $e'_i = (x_i, y_i)$ with $x_i = c_{1,i}$ and $y_i = c_{2,i}$ where $c_1 = (c_{1,1}, ..., c_{1,p})$ and $c_2 = (c_{2,1}, ..., c_{2,p})$ are the eigenvectors mentioned above.

Figure 3.7 illustrates the result of performing PCA on eight dimensions of the ABP state transition graph and calculating the e'_i as outlined above. PCA offers a way to identify those dimensions that have the most dominant influence on the variation of the data set. The cardinalities of node attributes influence the amount of variation, however. This can be addressed by normalizing every dimension before performing PCA. It may be argued, though, that a node attribute with a large degree of variation is not necessarily interesting. Also, as with uniform and manual distribution, no precise meaning can be attached to a position when $p > 2$. Nonetheless, from Figure 3.7 the two-phased behavior of ABP is clearly discernible.

3.6 Discussion

Existing techniques for visualizing multivariate data focus on assisting users in identifying dimensions that are similar in terms of predefined metrics (see Section 2.5.1). In this chapter it was shown that for system analysis other aspects of multivariate data are also important. Apart from indicating similarities, the techniques that were presented enable users to identify how dimensions differ from each other. They also show the presence of alternating patterns and they highlight patterns within patterns.

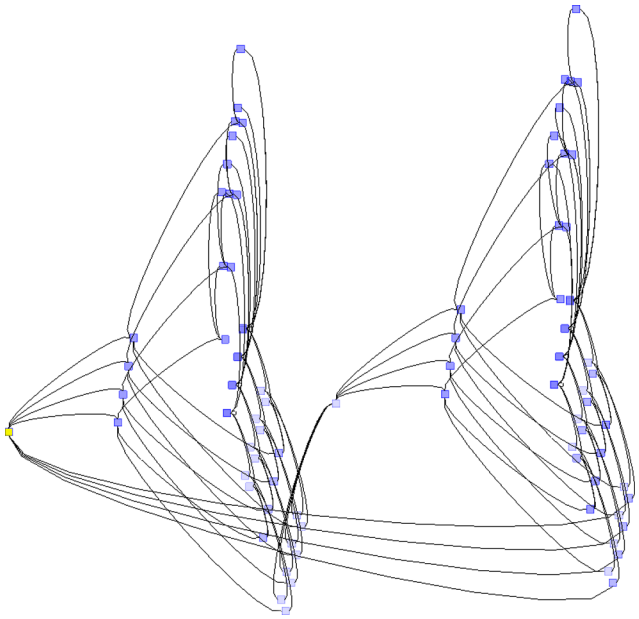


Figure 3.7: Principal component analysis (Alternating Bit Protocol).

The case given in this chapter shows that by presenting system analysts with their data in a visual way, they often learned that the behavior of the systems they study is not completely understood. It was also found that analysts are often unaware of their limited understanding until their data are presented to them in a way that causes them to raise questions about it.

The degree to which the techniques presented in this chapter meet the requirements of system analysts are considered below. Table 3.1 summarizes how well they assisted users to answer the questions introduced in Section 3.2.

Although simple, parallel histograms proved to be a useful visual aid for addressing many of the questions in Section 3.2. Out of all techniques considered, parallel histograms provide the most sufficient overview of state attributes. This includes the ability to study individual attributes and the ability to identify the presence or absence of correlations between multiple attributes.

Like many existing multivariate visualization techniques, the original aim of parallel histograms was to help the user identify interesting node attributes to study further. This led to an unexpected insight into the way that system analysts interact with their data. It was found that they usually have a very good understanding of what different attributes mean. It is also this knowledge that informs their choice of attributes to compare, first with the histograms and then using the various projection techniques. This discovery has an important impact on the visualization techniques that are discussed in the remainder of this dissertation.

Table 3.1: Support for answering the questions in Section 3.2.

	Histograms	*Uniform*	*Manual*	*Hypergrids*	*PCA*
Question 1	✓		✓[1]		✓[2]
Question 2	✓	✓[3]	✓[3]	✓	
Question 3	✓	✓[3]	✓[1]	✓	
Question 4					
Question 5	✓	✓[3]	✓[3]	✓	

[1] The movement of e_i' may be useful to some extent.
[2] If variation is considered interesting. Also for identifying phases of behavior.
[3] If $p \leq 2$.

Parallel histograms, like the other techniques considered in this chapter unfortunately do not support analysts in identifying interesting correlations between edge labels and node attributes. It is certainly the case that significant behavioral characteristics of a modeled system are captured in the edges and edge labels in the corresponding state transition graph. This is an important opportunity for future work and is treated in detail in Chapter 7.

In the prototype that was developed to investigate the ideas presented in this chapter, parallel histograms were combined with 2D projections in a single interface (see Figure 3.8). Besides providing insight into system behavior for subsets of node attributes, analysts found parallel histograms to be an intuitive steering device for 2D projections. Also, the combination of parallel histograms with projection methods often compensated for projection techniques that, when considered in isolation, do not scale that well beyond two dimensions; the ability to quickly select, deselect and order dimensions enable users to identify correlations spanning multiple dimensions. This is due to reflecting any interaction with the parallel histograms in the 2D projection in real time.

One disadvantage of projecting individual nodes to 2D, though, is that it limits the scalability of the approach. With datasets containing more than 10,000 nodes, real time interaction becomes a challenge. This could be addressed by first grouping similar nodes into clusters and then projecting only the clusters to 2D. Feedback from users suggests that the use of curved arcs works well to encode the directions of edges when graphs are projected to 2D. This convention is reapplied in subsequent chapters.

It is often important to determine whether nodes containing specific valuations of a subset of node attributes exist. This can be verified with parallel histograms, but hypergrids also proved useful. Due to the nested nature of the axes in hypergrids, users can locate positions where such nodes are or would have been projected to, had they existed. Hypergrids represent a high-dimensional space such that coordinates in it are still geometrically interpretable in 2D. This enables analysts to analyze multiple dimensions in a single 2D projection.

Note that the challenge of associating meaning to the position of nodes, apart from their relation to direct neighbors, is inherent to many graph visualization techniques, such as those discussed in Section 2.3. The ability to associate precise meaning with particular

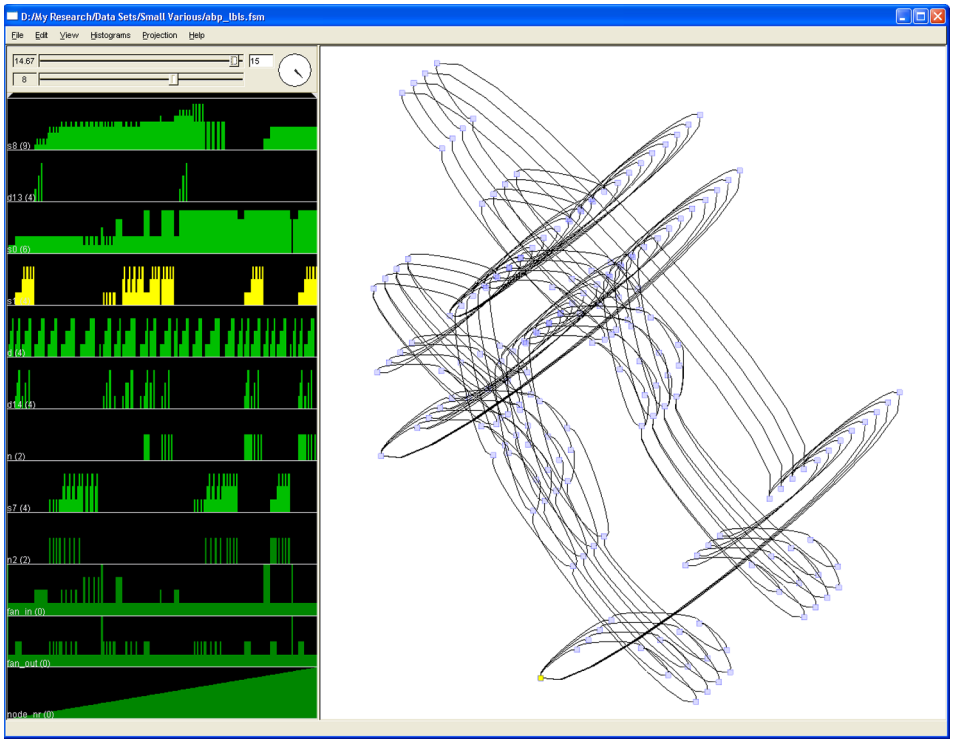


Figure 3.8: Combining parallel histograms with projection to 2D.

positions using hypergrids is a great advantage over the other techniques investigated in this chapter and is a principle that is returned to in the remainder of this dissertation.

A limitation of hypergrids, which has been mentioned, is that they are expensive with regard to display area as the number of attributes to consider increases. The same holds for attributes with large domains. For large or continuous domains, this can be addressed by partitioning the domain into a smaller number of discrete clusters; a practice known as binning.

As far as the other projection techniques are concerned, manual interaction was found to be useful for gaining insight into system behavior for selected node attributes. By interacting with the projection vectors, users were able to discover those attributes where there are large or small variances in the values that are assumed. This was done by considering the degree to which nodes were being displaced.

Users often positioned two attributes orthogonally when interacting with projection vectors. With the aid of parallel histograms they were able to cycle through many such projections quickly and efficiently, for example, by keeping one dimension constant while changing the other. A further extension of the approach discussed in this chapter, which has not been explored, is to enable the user to specify two subsets of attributes. The first

selection could be kept constant while the second is animated over.

Although 2D projections based on PCA do not provide precise answers to the questions in Section 3.2, system analysts agreed that this approach allowed them to get a better idea of different phases in system behavior. This helped them with formulating hypotheses, which could then be investigated with other projection techniques or even with non-visual analysis.

Chapter 4

Attribute-Based Clustering

As explained in previous chapters, state transition graphs have attributes associated with their nodes. In this chapter an approach for the visual analysis of state transition graphs, based on interactive attribute-based clustering of their nodes, is presented. Clustering results in metric, hierarchical and relational data, represented in a single visualization. To visualize hierarchically structured quantitative data, the bar tree, a novel technique, is introduced. This is combined with a node-link diagram to visualize the hierarchy and an arc diagram to visualize relational data. The effectiveness of the approach is illustrated by applying it to a real-world case. For this a transition graph that models the behavior of an industrial wafer stepper, containing 55,043 nodes and 289,443 edges, is considered.

4.1 Rationale

A crucial observation made in Chapter 3 is that system analysts associate precise meaning with the node attributes found in state transition graphs. They know which aspect of a modeled system an attribute describes and what it means when it assumes different values. This observation has been confirmed by system analysts at Eindhoven University of Technology and at CWI, the Dutch Institute for Mathematics and Computer Science.

As observed in Chapter 2 the vector of attributes associated with every node in a transition graph is usually the union of the variables introduced in the formal specification of the behavior of a system. Additional derived information can also be attached to nodes as attributes. Examples include graph-structural properties, such as fan-in and fan-out, and equivalence classes, a concept from operational semantics [18]. This further emphasizes the important role that node attributes can play in the analysis of state transition graphs.

With the two most successful visualization techniques presented in Chapter 3, parallel histograms and hypergrids, users can learn more about the behavior described by state transition graphs by analyzing their node attributes. These methods make it possible to determine the presence or absence of correlations between different attributes. Users can also check whether nodes with particular valuations for a subset of node attributes exist. In particular, the nested nature of hypergrids enables users to associate precise meaning

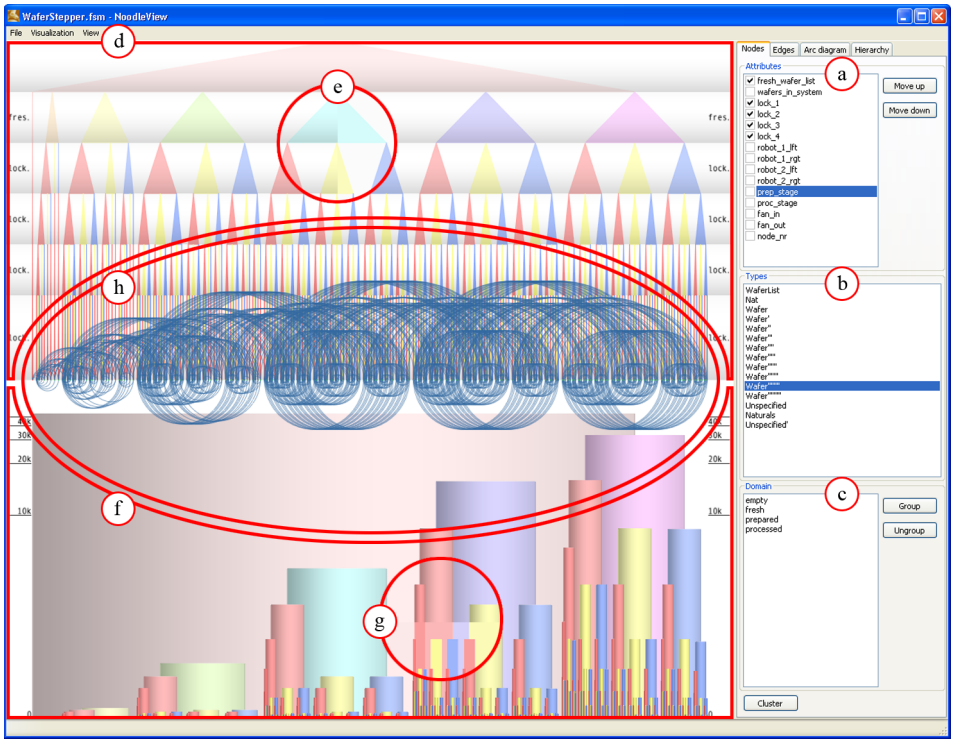


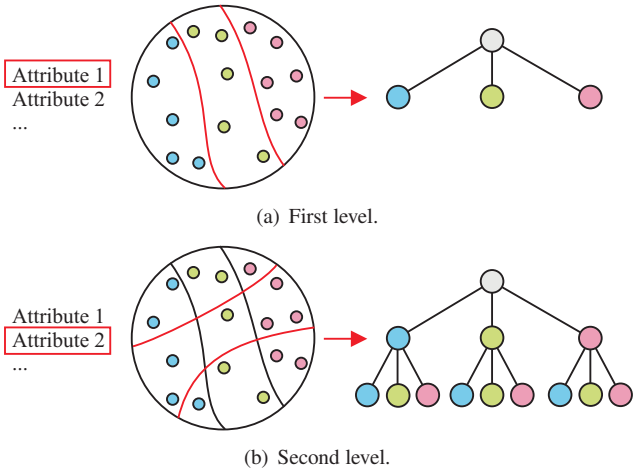
Figure 4.1: Visualization of multivariate state transition graphs.

with the locations to which nodes are projected.

In this chapter a technique is presented that builds on and extends the useful aspects of these designs. The approach presented here also addresses some of their shortcomings. For instance, by performing clustering before the transition graph is visualized, the technique scales better than by individually projecting nodes to 2D. More precisely, in this chapter the multivariate nature of state transition graphs is exploited in order to:

- Enable users to reduce the complexity of transition graphs in a semantically rich way.
- 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, the representations of three different types of data (hierarchical, metric and relational) are integrated in a single visualization. The objective is to enable the user to perceive and analyze correlations between these different elements. The following sections explain how this visualization is built by integrating different visualization techniques.



(a) First level.

(b) Second level.

Figure 4.2: Hierarchical attribute-based clustering.

4.2 Interactive hierarchical clustering

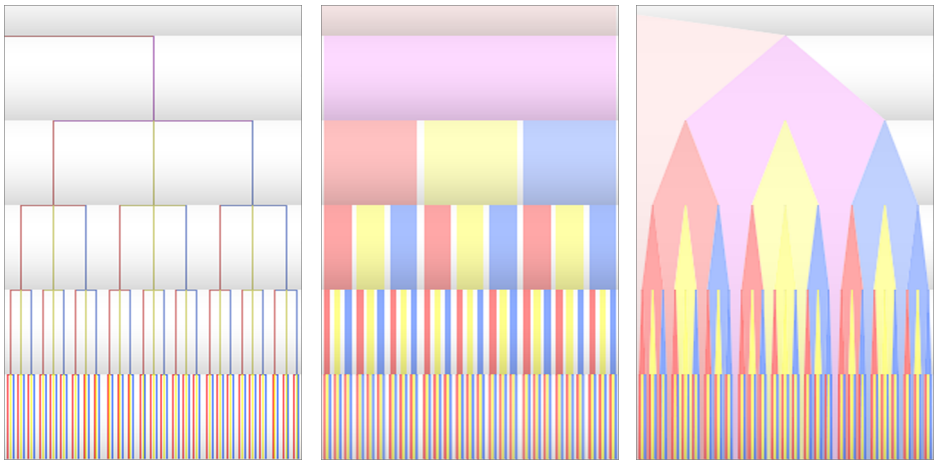
Like the visualization techniques introduced and discussed in Chapter 3, users can select a subset of node attributes they are interested in (see Figure 4.1(a)). The order of the attributes in this subset can be adjusted and the user is given an overview of the associated attribute types and their value domains (Figure 4.1(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 clusters branching from the root (see Figure 4.2(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 clusters in the clustering hierarchy (Figure 4.2(b)). Continuing in this fashion, a tree consisting of $p + 1$ levels is generated for a subset of p attributes.

Since the cardinalities of the attribute 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 an interaction mechanism, as shown in Section 4.5.

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 is discussed further in Section 4.4. The clustering hierarchy is used as the starting point of the visual representation of the clustering results and is shown as a node-link diagram in the top half of the visualization (see Figure 4.1(d)). In it parent-child relations are represented by triangles.

The visualization of the hierarchy uses subtle cushioning to differentiate levels better (see Figure 4.1(e)) [88]. Also, 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



(a) Dendogram. (b) Icicle plot. (c) Buttresses.

Figure 4.3: Alternative representations of the clustering hierarchy.

that a straightforward rainbow color mapping scheme suffices for this.

Three other graphical representations for the clustering hierarchy were also experimented with (see Figure 4.3). Users were unanimously in favor of triangles, citing that they strike 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 can cluster values of domains, thereby reducing their cardinality. This corresponds to the notion of data abstraction, as used by the formal methods community [14]. This results in reduced complexity and is a powerful analysis technique in itself (see Section 4.5).

4.3 Metric data

It is possible to compute quantitative values for every cluster in the clustering hierarchy. Such metrics are often cumulative; the values of non-leaf clusters can be obtained by summing the values of their children. An elementary and useful example is the number of nodes contained in the corresponding cluster (see Section 4.5).

To visualize such metrics a new technique was developed that uses nesting and layering to represent the hierarchy with which the metrics are associated. This visualization is called a bar tree.

Bar trees are constructed as follows (see Figure 4.4(a)–4.4(c)). Starting with the leaf clusters and working upward to the root, the metrics associated with clusters are encoded as a bar chart in every successive level. 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 clusters in level $i + 1$. This results in a layered nesting that reflects the clustering hierarchy.

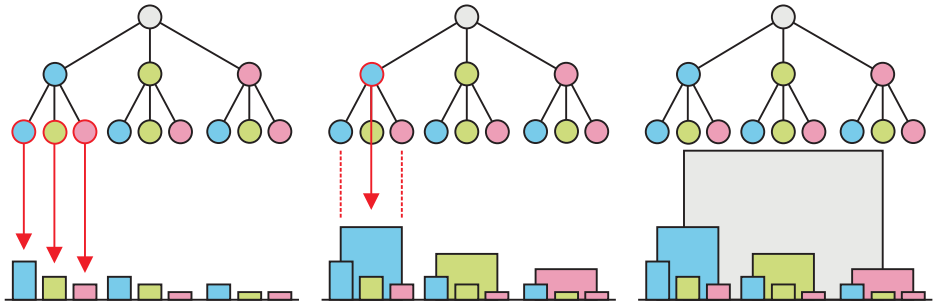


Figure 4.4: Bar trees.

As with many other visualization techniques for hierarchical metric data (for example, [73, 26]), 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. Exact values can be viewed by selecting the bars.

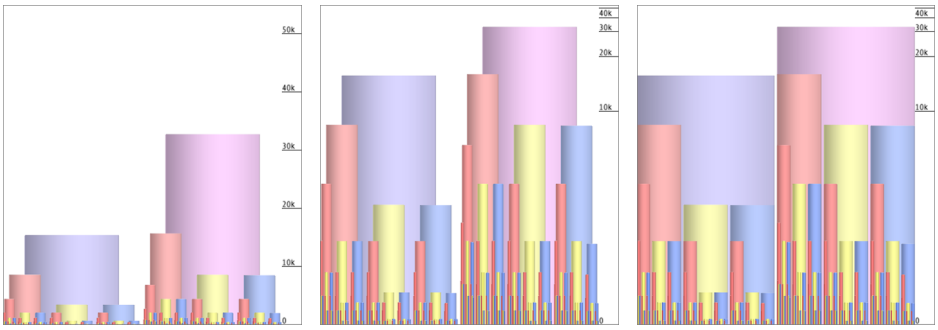
The bar tree is positioned directly below the clustering hierarchy in the bottom half of the visualization (see Figure 4.1(f)). This further emphasizes the relationship between the bar tree and the clustering hierarchy. Bars are coded with the same color as matching nodes in the hierarchy and users are able to show or hide layers corresponding to different levels of this hierarchy. To amplify the layered effect, cushioning is used (Figure 4.1(g)).

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 4.5(b), where this is done, much more detail of the lower levels of the bar tree is visible compared to Figure 4.5(a), where the distribution is linear. Experiments were also done with fully nested bar trees, but users indicated that the hierarchy is not as clearly discernible (compare Figure 4.5(b) and Figure 4.5(c)).

In the remainder of this chapter, the bar tree is used to encode 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.

Fully nested bar trees bear some resemblance to the "sum rendering method" proposed by Mihalisin et al. [50]. The two methods differ in terms of their appearance, application and aim, however.

In contrast to the visual representation used by Mihalisin et al., bar trees visually emphasize the layered nature of the clustering hierarchy using depth cues [88]. Mihalisin et al. use their technique to conduct a visual statistical analysis of multivariate data items. Users 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 is similar to the hierarchical clustering facility discussed above. Also, bar trees are used to represent summary information of an existing hierarchy and not the actual data dimensions themselves.



(a) Linear distribution. (b) Logarithmic distribution. (c) Fully nested.

Figure 4.5: Bar tree variations.

4.4 Abstract graph

The third and final element of the visualization introduced in this chapter represents the abstracted, less complex, transition graph that results from hierarchical clustering. It can be argued that this is the most important part of the 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. To capitalize on the fact that leaf clusters are positioned on a horizontal line, behavior is visualized using an arc diagram [90], positioned in the center of the visualization (see Figure 4.1(h)). It has been shown that arc diagrams are well suited for identifying repeated patterns in relational data [42].

To this end, the following steps are taken. First, co-directional edges are bundled. These bundles are represented by semi-circular arcs that span from the source to the target cluster (see Figure 4.6). The thickness of an arc is proportional to the number of bundled edges.

Additionally, the directions of the bundles are encoded in the orientation of the arcs. These are always interpreted in a clockwise fashion. User feedback from this, as well as from the results presented in Chapter 3, has been positive in this regard. Users seem to have little trouble adopting this convention. This makes visual pattern detection more meaningful since the directions of transitions play an important role in understanding the behavior that a transition graph describes. Also, it is not necessary 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 4.7). The visualization is magnified at these positions (Figure 4.7(a)) and compressed further away (Figure 4.7(b)) [67]. 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 discernible. Users did prefer a single focus band, though.

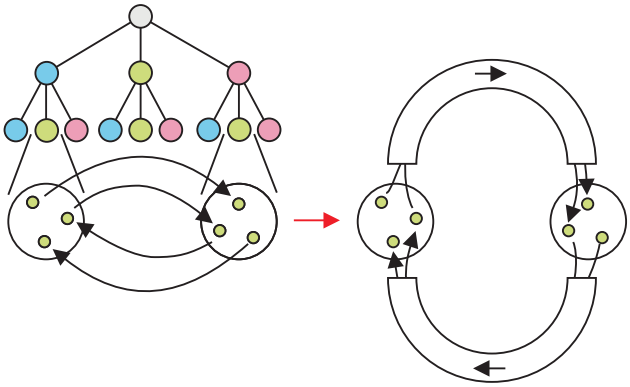


Figure 4.6: Arc diagram.

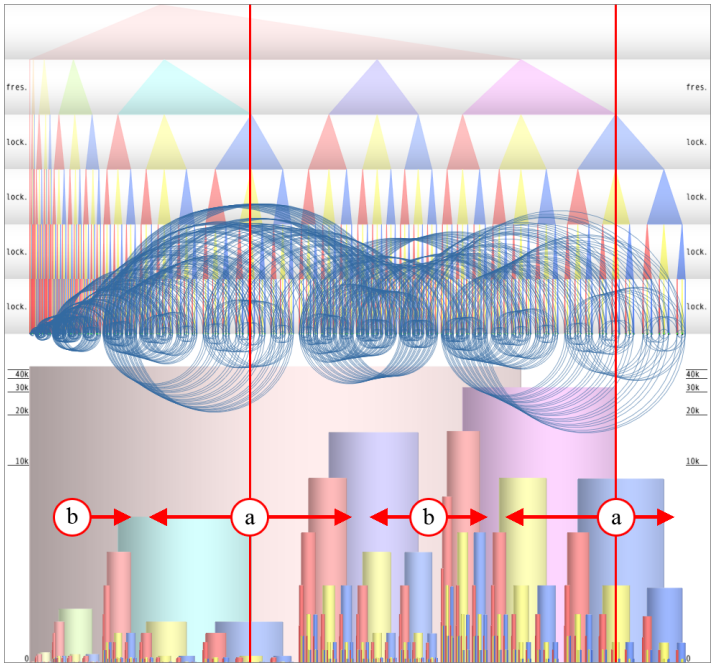


Figure 4.7: Focus bands.

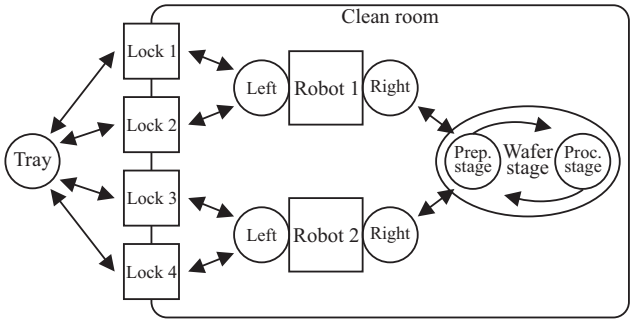


Figure 4.8: Wafer stepper.

4.5 Case - wafer stepper

In this section a real-world case study is considered. This is done to show how the approach introduced above can assist users in the visual analysis of state transition graphs. More concretely, the technique described in previous sections is used to analyze the transition graph of a generalization of the wafer handling of an industrial wafer stepper [30].

This system is used to manufacture integrated circuits by transporting unprocessed silicon wafers from the tray to the wafer stage, where they are processed, and back (see Figure 4.8). Due to the microscopic scale of integrated circuits, the process has to occur in an environment that is free of fine dust. This is called the clean room. To enter the clean room, wafers pass through vacuum locks. Once inside, wafers are handled by two robots.

More specifically, 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, left and right, which 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 edges. It has 15 associated attributes, of which the following are considered:

- *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–lock_4.* The contents of the four locks.

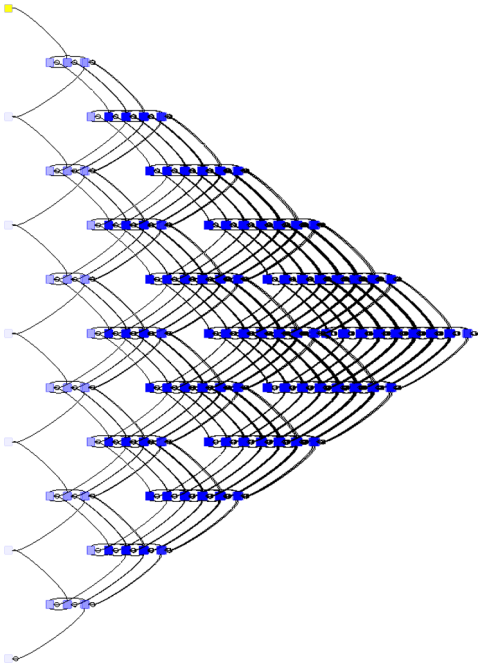


Figure 4.9: Wafer stepper transition graph projected to 2D (see Chapter 3).

- *robot_1_arm_left* and -*right.* The contents of the two arms of robot 1.
- *robot_2_arm_left* and -*right.* 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 traditional graph drawing techniques, the above transition graph resembles a tangled ball of yarn and consequently very little can be learned. Figure 2.6 in Chapter 2, which is in fact a visualization of this graph, shows that the approach by Van Ham et al. [25] also does not branch into symmetrical substructures. This means that the graph describing the behavior of the wafer stepper is an example of the problematic graphs referred to in Section 2.3. That is, the degree of interconnectivity between nodes is high. When viewed with the technique developed in the previous chapter, some suggestive behavioral patterns can be seen, but it is difficult to relate these back to concrete insights (see Figure 4.9).

In the following two sections, the visualization technique introduced in the current chapter is applied to analyze the behavior of the wafer stepper. Section 4.5.1 shows how the corresponding state transition graph can be analyzed in an explorative way. Section 4.5.2 illustrates how users can be assisted with in-depth, focused analysis.

4.5.1 Different perspectives

The technique described in this chapter makes it possible to take specific perspectives on a system being studied. This is achieved by clustering on a subset of attributes found in the data. Below, the transition graph describing the behavior of the wafer stepper system is viewed from three different perspectives. This allows for important observations to be made regarding a system's behavior and to check specific requirements.

Tray's perspective. To view the system from the tray's perspective the following attributes are selected in this order: *fresh_wafer_list* and *wafers_in_system*. The first describes the contents of the tray and the second indicates how many wafers are inside the clean room that have not yet been returned to the tray. In this fashion, it is possible to abstract from all other attributes and only consider the transition graph in terms of the two attributes related to the tray.

Clustering results in Figure 4.10(a). A first observation is that there are three types of transitions. First, those internal to *fresh_wafer_list* clusters and between *wafers_in_system* clusters (Figure 4.10(a)(i)). The arcs that represent these transitions span between clusters in the third level of the hierarchy, but do not cross the boundaries between clusters in the second level of the clustering hierarchy. Second, there are those transitions that span between *fresh_wafer_list* clusters (Figure 4.10(a)(ii)). The arcs that represent these transitions do cross the boundaries between clusters in the second level of the clustering hierarchy.

The third type of transitions are self-loops (Figure 4.10(a)(iii)). These transitions loop back to the cluster they originate from and can be thought of as internal actions. This is a notion borrowed from process algebra and refers to the hiding of certain types of edges with the aim of highlighting the influence of the remaining ones [18]. In this sense, the approach described here allows for dynamic action hiding.

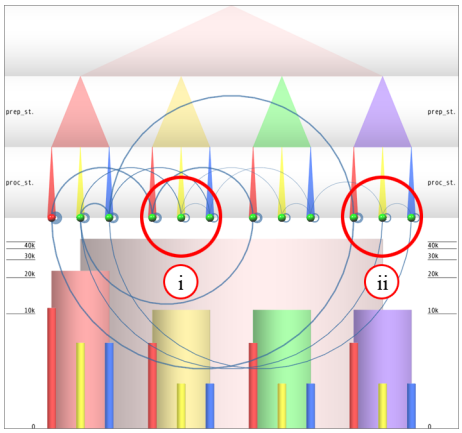
An important insight from Figure 4.10(a) is that from the tray's perspective, the system displays an explicit determinism. There is a clear forward progression from an initial cluster (far left) to a final cluster (far right), where all wafers have been processed. The bar tree visualization emphasizes this observation. Further, it is supported by noting that there are no back-pointing arcs in the arc diagram.

By selecting corresponding bars in the bar tree visualization, it can be confirmed 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. This makes it possible to say something about the liveness of the system. This term, from the model checking literature [14], expresses that a desirable state should eventually be reached.

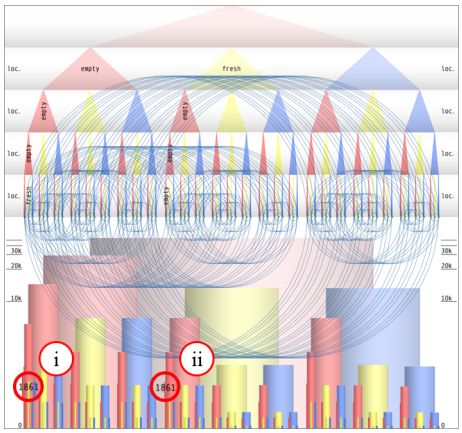
Wafer stage's perspective. Clustering on the attributes *prep_stage* and *proc_stage* enables analysts to consider the system from the perspective of the wafer stage (see Figure 4.10(b)). With this result, it is possible to validate important requirements.

First, it is impossible for an unprepared wafer to be processed. If the third level of the clustering hierarchy is considered (corresponding to *proc_stage*), it can be seen 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

(a) Tray's perspective



(b) Wafer stage's perspective



(c) Locks' perspective

Figure 4.10: Taking different perspectives.

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*.

The above requirements would call for significant effort to validate using other techniques. For instance, using formal model checking methods [14], they would have to be meticulously formulated and their validity proved.

Locks' perspective. The final perspective that is considered is that of the locks. To do so, the attributes *lock_1* through *lock_4* are selected. Figure 4.10(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, there are also corresponding regular patterns in the clustering hierarchy and the bar tree. This illustrates the claim that the technique introduced in this chapter makes it possible to match correlating patterns in the three types of data being visualized (metric, hierarchical and relational data).

Since the behavior of any one lock is independent of any other lock (the same behavioral pattern is present at all four corresponding levels of the clustering hierarchy), one can speak of behavioral symmetry. Furthermore, it is possible to check that for similar permutations of values assumed by the attributes (for example, (i) when locks 1–3 are empty and lock 4 contains a fresh wafer versus (ii) when lock 1 contains a fresh wafer and locks 2–4 are empty), the transition graph contains an equal number of nodes (by clicking on the bar tree visualization). From this it can be deduced that the four locks exhibit identical behavior. This is put to good use during the more detailed analysis in the next section.

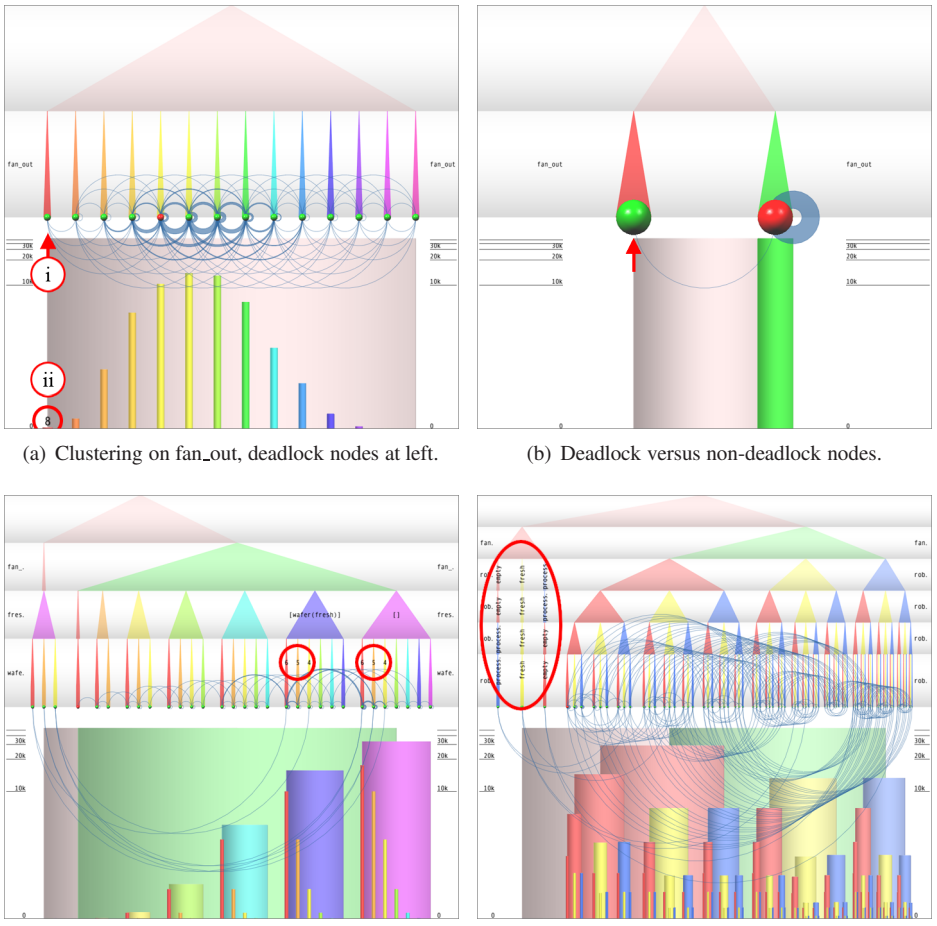
4.5.2 Focused analysis

A common objective of system analysis is to identify deadlock states [4, 18]. These are occasions when a computer-based system hangs or ceases to function. In state transition graphs such states are represented by nodes that do not have any outgoing edges.

Deadlock detection. The first step of deadlock analysis is to detect deadlock nodes. This can be done by clustering on the attribute *fan_out*. As a result the visualization in Figure 4.11(a) is generated. Immediately, the cluster of deadlock nodes can be identified on the far left (Figure 4.11(a)(i)). This is because the ordering of the clusters from left to right corresponds to an ascending ordering of the domain of *fan_out* with 0 at the left. Interaction with the bar tree corresponding to this cluster shows that there are exactly eight deadlock nodes in the system (Figure 4.11(a)(ii)).

Although the approach proposed by Van Ham et al. [25] 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 found. During a comparative study, only three deadlock nodes were identified.

There are also non-visual tools that offer ways to detect deadlock. However, after



(a) Clustering on fan_out, deadlock nodes at left.

(b) Deadlock versus non-deadlock nodes.

(c) Limited number of wafers can prevent deadlock.

(d) Specific robot configurations lead to deadlock.

Figure 4.11: Deadlock detection.

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 the aid of visualization, deadlock analysis is much easier, as shown below.

Starting with the results in Figure 4.11(a), the analysis can be greatly simplified by noting that the property being investigated is deadlock and by clustering the domain values of the *fan_out* attribute accordingly. When hierarchical clustering is reapplied this results in a binary partitioning of the nodes as illustrated in Figure 4.11(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, the aim is 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, the transition graph can be clustered on *fan_out, fresh_wafer_list* and *wafers_in_system*, in that order and with the domain of *fan_out* still clustered. This provides an interesting result.

In Figure 4.11(c) it can be seen that deadlock nodes can only occur from nodes where there is a single or no unprocessed wafer left on the tray (this corresponds to the orange and pink triangles in the second level of the hierarchy). This is observed 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, it is already possible to 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 4.11(d)). Since all deadlock nodes are clustered toward the left of the visualization, it is immediately clear that deadlock only occurs under very specific conditions. That is, when either:

- One of the robots has a processed wafer in each arm while both arms of the other robot are free.
- When both robots have fresh wafers in both arms.

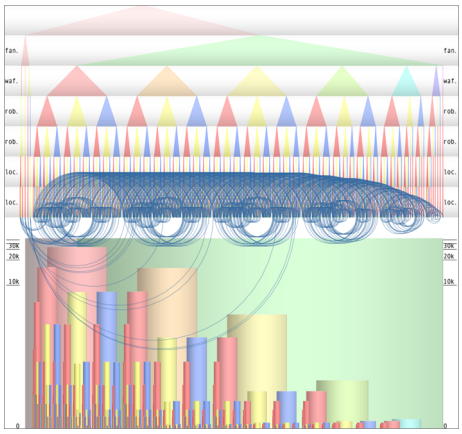
Deadlock analysis. Finally, a specific deadlock occurrence is considered. This is done under the assumption that it suffices to analyze a single wafer path. That is, the analysis is restricted to only one robot and the locks that it serves. This assumption is justified by the observation regarding the symmetric behavior of the locks in Section 4.5.1. In order to identify deadlock nodes, *fan_out* is again taken into account when clustering. Consequently, the result depicted in Figure 4.12(a) is generated.

Notice that a high-dimensional data set is being considered. Nonetheless, despite the large number of nodes and edges, 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.

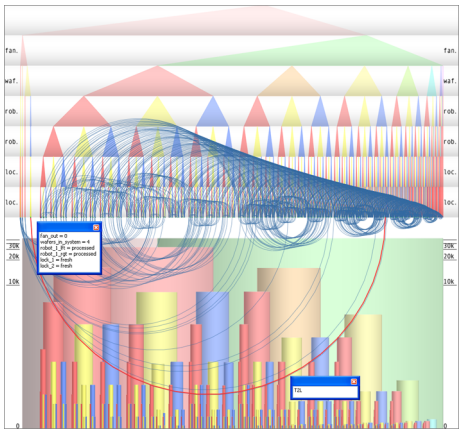
When the deadlock nodes toward the left are zoomed into and one of these is selected (see Figure 4.12(b)), this results in a pop-up window with an overview of all the selected attributes and the values they assume. When one of its incoming transitions is selected it lights up and an overview of all actions bundled in the arc is shown. It is now easy to visually trace this bundle of transitions and zoom in on its originating cluster (see Figure 4.12(c)).

Using the above approach, analysts were able to identify the exact conditions that lead to this occurrence of deadlock. This is illustrated in Figure 4.13 and is related to the earlier observation regarding the configurations of the robot arms when deadlock occurs.

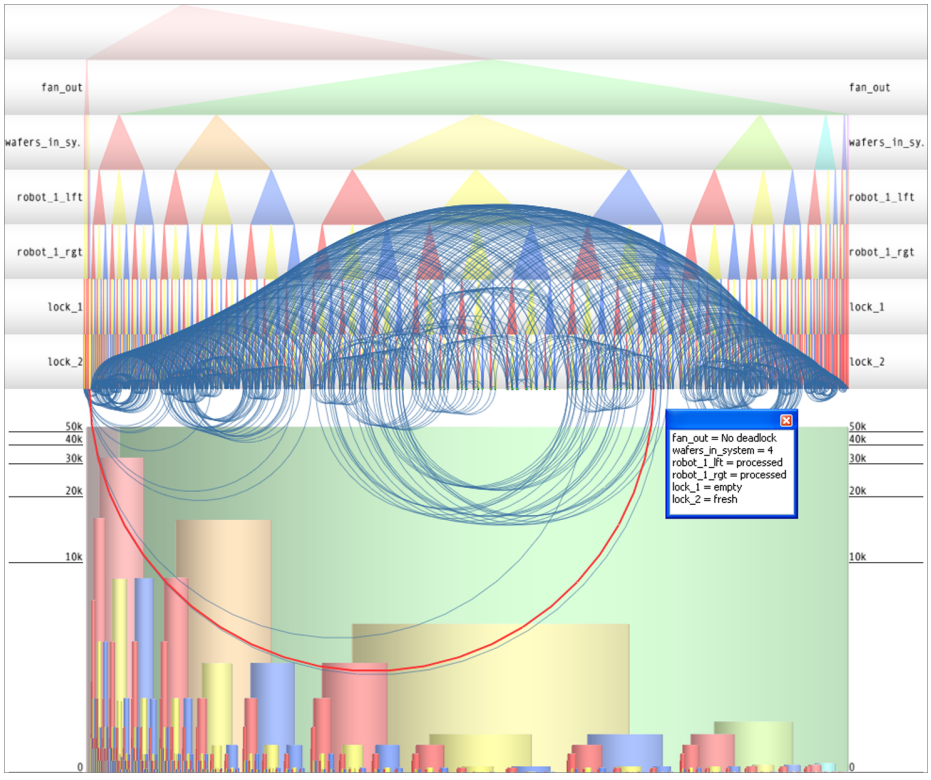
Right before deadlock occurs, robot 1 contains two processed wafers, lock 1 contains a fresh wafer and there is one fresh wafer on the tray. This is illustrated in Figure 4.13(a). When a *tray_to_lock* action is executed next, whereby this fresh wafer is transported to lock 2, deadlock occurs (see Figure 4.13(b)). The reason for this is that once wafers have



(a) Deadlock nodes at left.

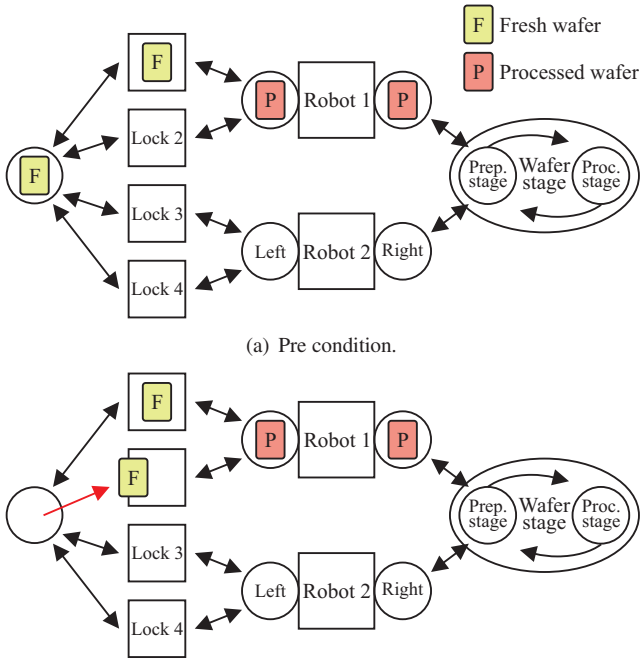


(b) Zoom in, select deadlock transitions.



(c) View configurations leading to deadlock.

Figure 4.12: Deadlock analysis.



(a) Pre condition.

(b) Deadlock.

Figure 4.13: Wafer stepper deadlock.

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.

4.6 Discussion

In the previous section, the advantages of the approach introduced in this chapter were demonstrated with a case from industry. It was shown how the approach enabled analysts to reduce the complexity of transition graphs in a meaningful way. Further, it was illustrated how the approach can serve as a useful mechanism with which to perform analysis.

Because users know what node attributes model, they are able to apply the technique to get a better intuition for the systems being studied by taking different perspectives. They are also enabled to conduct more focused analysis. In this regard, it has been shown that it is possible to check system requirements and to identify and clarify scenarios that lead to deadlock.

It was shown how, for a number of problems, the attribute-based approach advocated in this chapter works better than existing visualization techniques for state transition graphs. To do so, a novel visualization for encoding hierarchically structured quantitative data, referred to as a bar tree, was introduced. Existing visualization techniques have also been extended by combining arc diagrams with hierarchical information. The author

knows of only one other approach that combines relational and hierarchical data using arc diagrams [52]. This approach uses containment to encode hierarchical relations, however, and is limited in the number of hierarchical levels and relations that can be effectively visualized. Additionally, encoding direction in the orientation of arcs enhances the visual language of arc diagrams.

Users have been observed to talk about transition graphs and the behavior they describe in terms of the visual results discussed in this chapter. This includes the use of terms such as "symmetry, irregularity, third level" and "green cluster." Users agreed that this enhanced communication between themselves and they expect that this will also hold for other stakeholders.

One open question is whether the approach is useful for arbitrary multivariate graphs such as email traffic and citation networks. In state transition graphs, nodes have multivariate attributes that are discrete and of low cardinality. In more general cases, though, this cannot be guaranteed. To deal with larger, possibly continuous domains, these can be reduced in size or discritized with domain clustering or binning (see Section 4.1). For instance, an attribute that encodes the age of email users can be clustered into a smaller number of age categories. Based on the above, it is argued that the results presented here are also applicable in the wider context of multivariate graph visualization.

State transition graphs are generated from system specifications (see Section 2.1) and generally contain some degree of regularity in terms of their edges, which represent system behavior. To cater for more irregular cases the approach described in this chapter can be used to study higher level selections. For instance, to study the relations between gender and country in email traffic. Another interesting option would be to extend the approach with edge filtering. Using such a facility, it would be possible to analyze email traffic between specific geographic regions, for instance.

As far as the visual analysis of state transition graphs is concerned, the results presented in this chapter suggest a number of further opportunities. Details about so-called may- and must-transitions could provide analysts with important insight into their data. When edges between two clusters of nodes are bundled, a number of different actions are grouped together. Some of these actions are possible from all nodes in the originating cluster (must) while some are not (may). This issue receives closer attention in Chapter 7.

While interacting with the visualizations discussed in this chapter, users have been observed to make diagrammatic sketches corresponding to system configurations represented by particular clusters. Similar to how the diagrams in Figure 4.13 facilitated the discussion in this chapter, users' hand-drawn diagrams were used to reason about the systems they studied. This observation serves as motivation for the approach pursued in the following chapter.

Chapter 5

User-Defined Diagrams

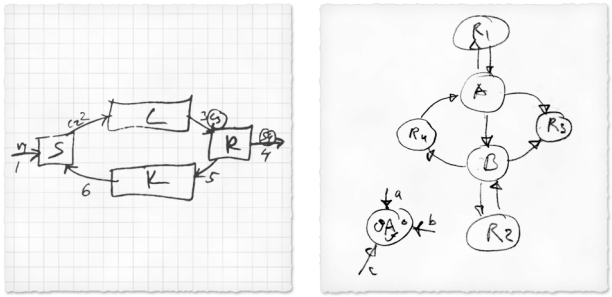
Users are often faced with large data sets where the underlying meaning of the data is clear, but where sense-making is difficult. State transition graphs fit this description well. As argued in previous chapters, system analysts associate precise semantics with multivariate attributes that describe the nodes of transition graphs, but most visualization techniques do not exploit this. Consequently, users have to invest significant effort to interpret and understand visualizations. In this chapter a novel technique that bridges this semantic gap is introduced. This is achieved by enabling users to define custom diagrams that closely reflect the associated semantics. User-defined diagrams are incorporated in a number of correlated visualizations that support different user needs. To show the merit of the approach, two cases of its application to real-world data sets are discussed.

5.1 Rationale

Throughout this dissertation it has been argued that despite being conceptually straightforward, state transition graphs are not easy to analyze. For example, it is often hard to tell if, when and why nodes that represent undesired states can be reached. In this sense, transition graphs are representative of many data sets where the organizational structure and meaning of the data is clear, but where gaining insight is difficult.

In Chapter 2 the difficulty of studying state transition graphs was attributed to two factors. First, there is the state explosion problem: transition graphs often contain large numbers of nodes and edges. As a result, it is difficult to make sense of the behavior they represent. Second, state transition graphs describe system behavior at a low abstraction level. Although analysts possess deep domain knowledge, it is a challenge to map this to nodes and edges in the data. This will be referred to as the semantic gap.

This chapter introduces an approach for bridging the semantic gap. Although both the above issues are addressed, the emphasis is on the latter. To achieve this, users are enabled to define custom diagrams that reflect their conceptualization of a problem. These user-defined diagrams are integrated in a number of correlated visualizations, which allow analysts to study and reason about system behavior in a familiar way.



(a) (b)

Figure 5.1: Analysts' sketches.

5.1.1 Conceptualization

As noted in Section 2.1, state transition graphs are generated from formal specifications of system behavior. To specify a system's behavior, analysts have to conceptualize it. For computer-based systems this is typically in terms of a number of interacting components.

For example, reconsider the communication protocol introduced in Chapter 2. It consists of a sender, a receiver and two communication channels. Analysts will specify each component as a different process at an appropriate level of detail. Automated tools then generate a transition graph by interleaving these specifications. Variables used for describing the different components are found back in the transition graph as node attributes. Every node of the transition graph will consist of a vector of attributes that respectively represent the status of the sender, the receiver and the different communication channels.

To get more insight into the needs of system analysts two steps were taken. First, analysts were observed as they reasoned about system behavior. Second, they were asked to explain how the systems they study work. In cases where errors had been identified, analysts were asked to elaborate on what they had found.

An interesting discovery was made: analysts regularly used visualization for the above tasks. In all cases they resorted to drawing diagrams where graphical objects, like circles and squares, depict particular system components (see Figure 5.1). Moreover, in analysts' conceptualizations of the systems, there were clear semantic mappings between every graphical object and one or more of the attributes that describe every system state.

For example, the hand drawn diagram shown in Figure 5.1(a) represents the communication protocol discussed above. The two squares at its left and right represent the sender and receiver. The two communication channels that connect the sender and receiver facilitate message passing. As mentioned, each of these components is described by a node attribute in the corresponding state transition graph.

In addition to the experiment described above, users have been observed to make sketches, such as those shown in Figure 5.1, while analyzing state transition graphs with the visualization technique introduced in the previous chapter. This serves as further motivation to pursue the current approach.

5.1.2 Simulation

System analysts often use guided simulation to study system behavior [72]. This is supported by tools called simulators. These enable analysts to traverse a transition graph incrementally starting from the initial node. As explained in Section 2.2, the user considers a current state and the simulator provides an overview of all transitions (directed edges) possible from it. When the user selects one of these transitions, the simulator updates the current state and a different set of transitions becomes possible.

To experience first-hand how users go about analyzing systems in this way, the author collaborated with an analyst to study an automated parking system [48]. The approach taken was to consider nodes that represented potentially dangerous system states. Next, all behavior possible from such nodes was considered to determine whether errors really did occur. It was soon realized, however, that interpreting the text-based output of the simulator tool was arduous, not intuitive and prone to human error.

Many simulators require users to interpret long text dumps, leading to the above problems. Visualization can relieve users of this burden. For example, to analyze the parking system, a simple custom visualization plug-in was developed for the simulator [48]. This visualization maps a single state of the modeled parking garage to a 2D floor plan. It uses visual cues, such as color and position, to encode various attribute values (representing parking spaces, automobiles, and so forth) that describe every system state.

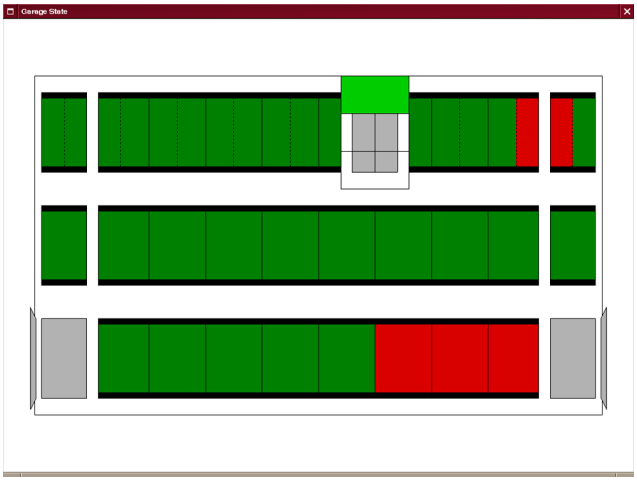
Despite its simplicity and limited features, this solution was extremely effective and a number of serious mistakes in the design of the parking system were identified. For example, a number of configurations were discovered where, had it been built, the mechanical hardware of the automated parking system would have been able to tear one or more cars in half.

One of these scenarios is illustrated in Figure 5.2(a) and Figure 5.2(b) where the two red rectangles at the top right of the diagram represent two halves of the same car before and after the system executes an erroneous transition. The checks that were performed by the system were designed on the premise that a car can only be in a single parking bay when, in fact, it was possible for a car to be positioned halfway across two parking bays as in Figure 5.2(a).

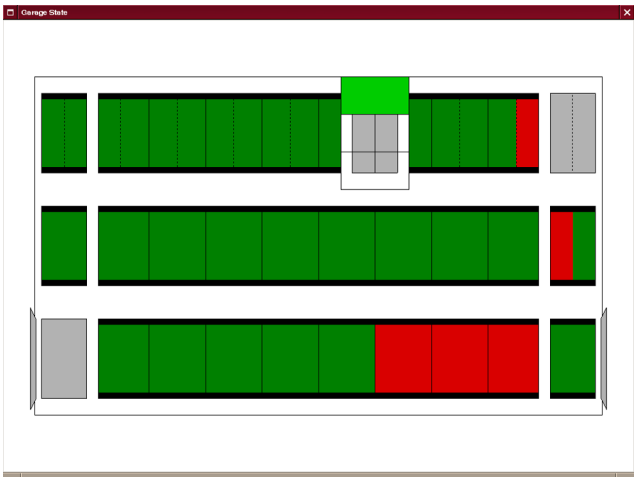
To communicate the severity of these findings to the client, there were two hurdles. The client was a businessman and not interested in technical details. Also, he did not speak English. Nonetheless, by pointing to a laptop and silently walking him through one problem using the visual representation of the garage he instantly understood. There was no need for a verbal explanation. This serves as further proof of the effectiveness of using diagrams for system analysis.

The success of the above solution is attributed to the fact that the visual representation was a close match to how analysts and other stakeholders reasoned about the parking system. From this, it was concluded that presenting a problem in terms of the user's own conceptualization of it can be very effective.

Despite being effective, the above method is a custom solution for a specific system. To avoid having to program new visualizations for every system studied, users have to be provided with visualization tools that are reconfigurable. Some results exist for scientific simulation. Ribarsky et al. [66] proposed user-defined glyphs of which the graphical



(a) Unsafe scenario.



(b) A car is torn in half.

Figure 5.2: Automated parking system [48].

properties can be interactively bound to data. As a particular data item assumes different values, the corresponding glyph's properties, such as position and size, change.

As suggested, the current aim is also to let users visually capture their conceptualization of a problem rapidly and to use this visual representation to gain further insights. Such an approach was taken with the work by Van Wijk et al. [95] on computational steering. Users define diagrams that realistically reflect their understanding of a problem. They then parameterize the diagrams by linking them to a scientific simulation engine, much like the approach of Ribarsky et al. The dynamic behavior of a simulated system is shown as an animation. Diagrams can also be used to steer simulations by acting as

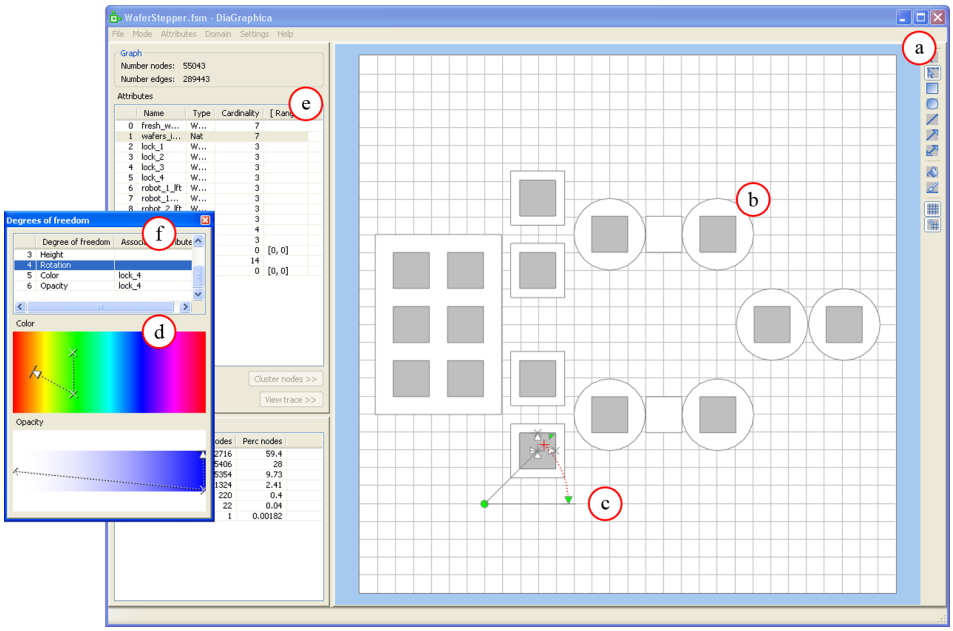


Figure 5.3: Editing custom diagrams.

input devices for changing variable parameters. This technique helped users to gain a better understanding of the influence of multiple data parameters on complex scientific simulations.

5.2 Custom diagrams

With the above as point of departure, this chapter introduces a three-part strategy for the visual analysis of state transition graphs. This approach lets the user:

1. Define custom diagrams that reflect the semantics associated with the data.
2. Explicitly link node attributes in the data with graphical properties of the diagrams.
3. Use these diagrams to bridge the semantic gap.

Below, these steps are discussed in turn. To illustrate the approach, it was implemented in a prototype. Using this system, users are able to work in either edit mode or analysis mode.

In edit mode users define custom diagrams with an integrated graphical editor (see Figure 5.3), which provides basic direct manipulation facilities like most vector-graphics suites (Figure 5.3(a) and Figure 5.3(b)). Users can draw graphical primitives such as ellipses, rectangles, lines and arrows and can rotate, resize and move them. They can also

edit properties such as fill and outline color. Other utilities such as a snap-to-grid feature and a clipboard are also provided. Many extra features could have been added, but it is argued that the ones that are provided are sufficient for a proof of concept and rich enough to cater for many applications of the approach.

5.3 Parameterization

It has been mentioned a number of times that users attach precise meaning to the node attributes found in state transition graphs. Also, in the user's conceptualization of a problem, as manifested in a custom diagram, these attributes often have clear semantic mappings to the graphical objects in the diagram (see Section 5.1.1).

To exploit this, after a diagram has been drawn, users can explicitly link node attributes with the graphical properties of the diagram. Inspired by the idea of user-defined glyphs and parameterized graphical objects (see Section 5.1.2), seven Degrees of Freedom (DoFs) are defined for all shapes in such diagrams: x-position, y-position, width, height, rotation (about a movable pivot), hue and opacity. DoFs represent the graphical properties of every shape. In edit mode, users can edit DoFs by direct manipulation. This involves defining a range for the DoF and linking a node attribute with it.

When editing a shape the user is presented with a graphical representation of its DoFs. For the geometric DoFs (x-position, y-position, width, height and rotation), the user selects and drags handles to define a linear range (see Figure 5.3(c)). Non-geometric DoFs, such as hue and opacity, are also defined visually (Figure 5.3(d)). Because the shortest path between two points in color-space is often unintuitive, users can define a number of control points in the hue- and opacity-channels. To specify an exact path, the user can add new control points, drag existing ones to new positions or delete them.

The user is provided with an overview of all node attributes of a data set loaded in the tool (see Figure 5.3(e)). When editing a diagram, a list of DoFs associated with the currently selected shape is also shown (Figure 5.3(f)). To link a node attribute and a DoF, the user drags and drops the attribute on the appropriate DoF. One attribute can be linked with many DoFs, but any DoF can only have a single attribute associated with it. In this way, users are able to link the node attributes found in transition graphs with their own custom diagrams.

The diagram representing a particular state is obtained by mapping attribute values to the ranges of the DoFs. This is done through interpolation based on the position of an attribute value in its domain. Consequently, the ordering of values in a node attribute domain is important and users can adjust this. In this fashion, for a linked attribute and DoF, the data values assumed by the attribute determine the graphical property represented by the DoF.

This is illustrated in Figure 5.4 where two attributes, both with a domain of $\{0, 1, 2\}$, are considered. The attributes have been linked with a diagram containing a single rectangle. *Attribute 1* has been linked with the DoF describing the rectangle's x-position and *Attribute 2* with the DoF describing its y-position. For a node where *Attribute 1* = 1 and *Attribute 2* = 2, the resulting diagram is shown on the right.

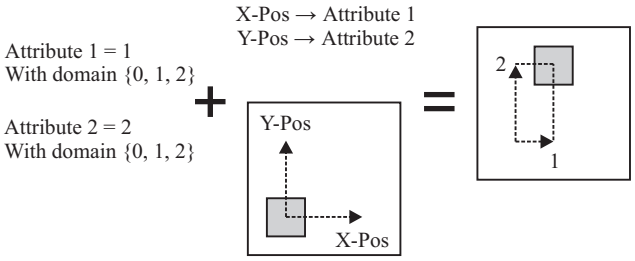


Figure 5.4: User-defined diagrams.

5.4 Bridging the semantic gap

To bridge the semantic gap (see Section 5.1), the goal is to incorporate user-defined diagrams in visualizations that enable the user to:

- Reduce complexity and identify points of interest while retaining an overview of the behavior represented by the data.
- Explore a local neighborhood around a point of interest.
- Combine the inspection of interesting aspects with the ability to temporarily store results.

To meet these requirements, three correlated visualizations were developed. These are discussed below.

5.4.1 Cluster view

Figure 5.5 shows the analysis mode interface. As noted in Section 5.1.1, when users were only presented with a clustering hierarchy and its associated aggregate data (see Figure 5.5(a)(i)–(iii)), they often made diagrammatic sketches to assist them in their analysis. With this as inspiration, the clustering hierarchy developed in Chapter 4 has been extended with the custom parameterized diagrams introduced above. When the user selects a cluster, it is annotated with a callout showing the user-defined diagram (Figure 5.5(a)(iv)). All parameterized DoFs are calculated on the values assumed by the node attributes with which they are linked.

When users parameterize diagrams with node attributes not considered for clustering, the different nodes in each leaf cluster in the clustering hierarchy map to different diagrams. Users are interested in these differences. A number of icons are provided to let them step through these diagrams or to view them in an animated sequence (see Figure 5.5(a)(v)). Thus, they can identify interesting configurations within the diagrams or parts of the diagram that remain constant. As users step through the different diagrams, arcs corresponding to the current diagram's incoming and outgoing edges are highlighted (Figure 5.5(a)(vi)).

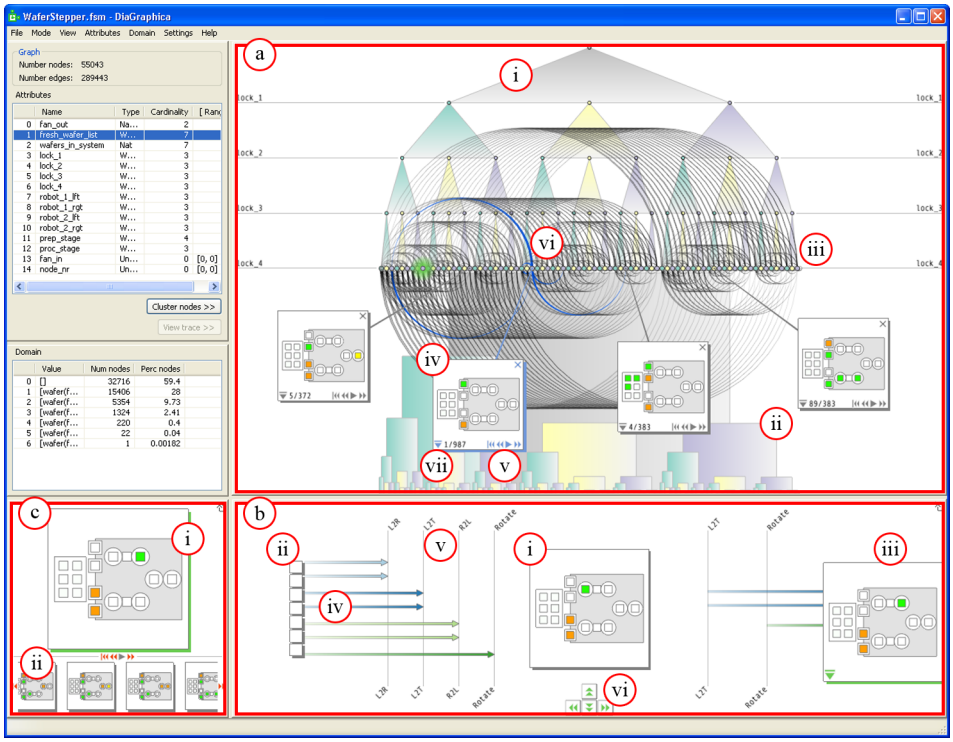


Figure 5.5: Analysis mode.

Users can define projections of the transition graph by choosing not to link node attributes with shapes in the diagram. For example, consider the communication protocol discussed in Section 5.1.1. Suppose the user does not parameterize any shapes with those node attributes that describe the communication channels. Also suppose that the attributes describing the sender and receiver are linked with shapes and can respectively assume two values. Then, there are only four possible mappings of nodes to diagrams. Such a collection of nodes that are mapped to the same diagram is called a projection.

A projection can serve as an abstraction and analysis mechanism, much like the selection of a subset of attributes to be considered for clustering (see Chapter 4). For simplicity, in the remainder of this chapter, it is assumed that all node attributes are linked with shapes in the diagram and that every distinct diagram represents a unique node.

5.4.2 Simulation view

To explore a local neighborhood around a point of interest the simulation metaphor is used (see Section 5.1.2). After identifying an interesting diagram in the cluster view, the user can load it into the simulation view by clicking on the icon in the lower left corner (see Figure 5.5(a)(vii)). In the simulation view (Figure 5.5(b)), the current node is visualized

as a diagram in the center (Figure 5.5(b)(i)).

Instead of presenting only the current node, as traditional simulators do, the user is provided with an overview of all nodes leading to and from it (see Figure 5.5(b)(ii) and (iii)). The simulation view shows these nodes as diagrams at its left and right. Users can view details by using mouse rollovers to magnify the diagrams.

Edges, representing transitions, are encoded with horizontal arrows, as shown in Figure 5.5(b)(iv)). It is likely that edges from various nodes involve the same edge label. There may also be numerous edges with different edge labels possible from or to a particular node. To avoid cluttering in such cases, arrows are not simply labeled. Instead, an overview is provided by showing a sorted list of incoming and outgoing edge labels at the top of the visualization (Figure 5.5(b)(v)). A number of vertical lines, corresponding to unique edge labels, are also defined. The length of an arrow is determined by taking the intersection with the vertical line representing its edge label. Arrows representing edges with the same edge label are identically colored.

These cues enable the user to identify repeated patterns in terms of edges. For example, Figure 5.5(b) shows three pairs of nodes at the left with the same outgoing edges. The user may choose to inspect these to determine whether they share any other similarities. Even when many transitions are possible from a particular diagram, the user is able to view individual ones by rolling over edge labels at the top of the visualization. All arrows corresponding to the currently selected label are then highlighted.

To navigate the transition graph the user selects any diagram leading to or from the current node. Consequently, its diagram slides toward the center and becomes the new current node. The user can also navigate using icons (Figure 5.5(b)(vi)) or the keyboard. The position of the currently active diagram is highlighted in the cluster view to maintain context.

5.4.3 Inspection view

Finally, users have to be able to inspect interesting nodes more closely and to temporarily store them. This is to facilitate communication with colleagues and other stakeholders. The inspection view was developed for this purpose (see Figure 5.5(c)).

First, it serves as a magnifying glass (Figure 5.5(c)(i)). When the user moves the mouse over a diagram in any view, the inspection view shows it enlarged. Second, the user can employ the inspection view as temporary storage (Figure 5.5(c)(ii)). For instance, the user may want to keep a history, store diagrams from various locations in the transition graph to compare them later, or keep diagrams as seeds for discussions with colleagues. Stored diagrams are represented as a scrollable list.

By clicking on an icon on any diagram it can be loaded into the inspection view or from the inspection view to any other view (Figure 5.5(a)(vii)). To maintain context the currently active diagram is highlighted in the cluster view.

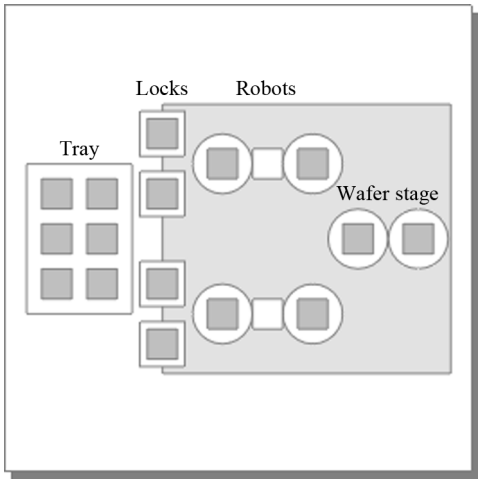


Figure 5.6: User-defined diagram of a wafer stepper.

5.5 Case - wafer stepper

To illustrate how the above approach can assist users with system analysis, two applications to real-world case studies are presented in this and the next section. First, the wafer stepper system introduced in Chapter 4 is reconsidered [30]. The state transition graph that describes its behavior contains 55,043 nodes, 289,443 edges and has 15 node attributes. Using the visualization technique introduced in the previous sections, a number of further discoveries were made about this system.

To analyze the wafer stepper, a diagram of the system was drawn and parameterized. Figure 5.6 shows this diagram. On the left there is a tray that can hold six wafers. Fresh, unprocessed wafers enter the system here and then move to one of four locks, represented in the diagram by a column of four rectangles. Next, one of the two robots, represented by barbell-shaped configurations, takes a wafer from the locks using its left arm and rotates $180°$. It then places the wafer on the processing stage, represented by two circles at the right. Here the wafer is processed. To exit the system, wafers follow a similar path back to the locks.

It had previously been observed that the robots that move wafers from the locks to the wafer stage occasionally behaved in unpredictable ways. Consequently, there was particular interest in studying this system from the perspective of the two robots. To do so, the data was clustered on the attributes that describe the contents of the four robot arms (two per robot). All four attributes have the domain $\{$*empty, fresh, processed*$\}$. The resulting clustering tree's leaves represent all possible combinations of these values (see Figure 5.7).

To get a general impression of the robots' behavior, leaf clusters were annotated with the custom diagram and a number of these diagrams were loaded into the simulation view (see Figure 5.8(a) and Figure 5.8(b)). An interesting discovery was made.

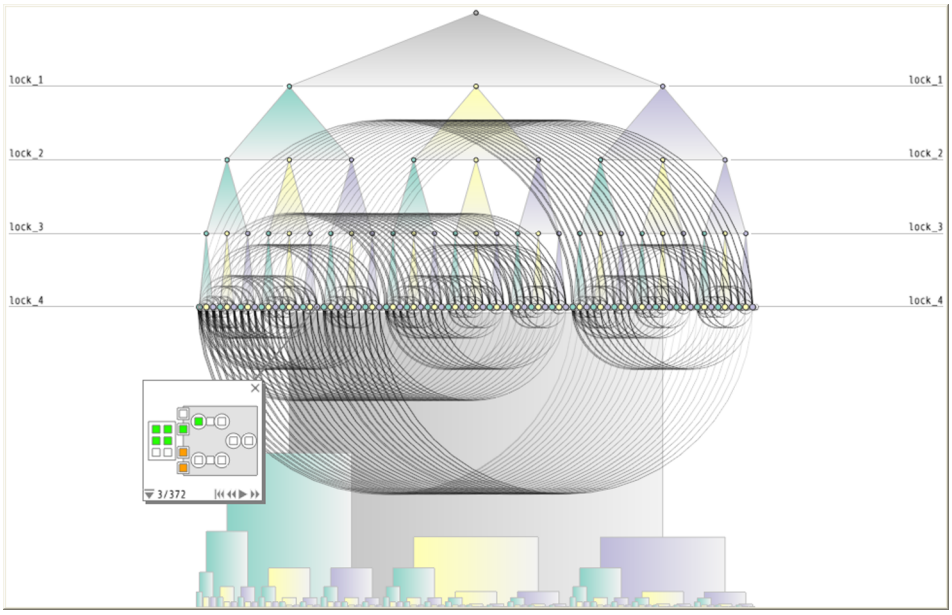


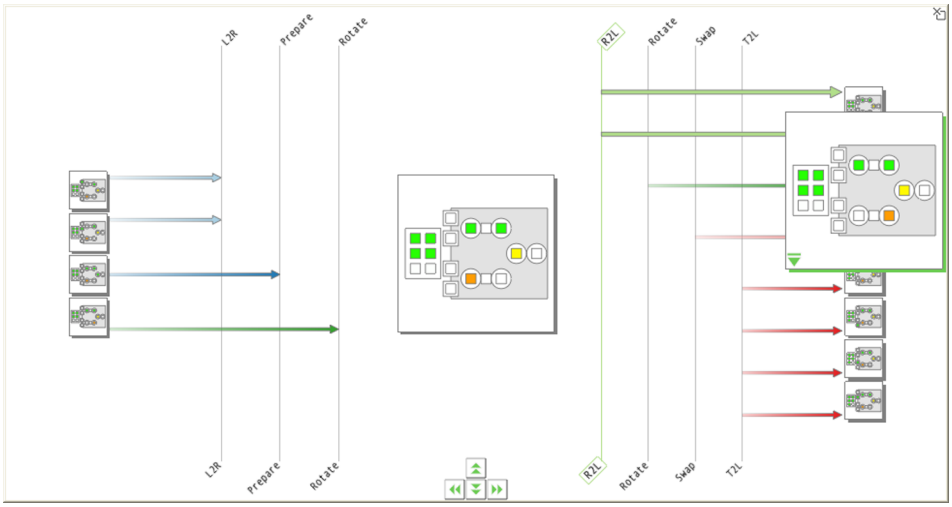
Figure 5.7: Wafer stepper from the locks' perspective.

In the diagram, the shapes representing the robot arms were parameterized such that their values map to transparent, green or orange rectangles. It was noticed that from many configurations, the wafer stepper system contained potential livelock. Livelock is a scenario where it is possible to cycle through a number of nodes infinitely. In the wafer stepper, livelock is possible when, for a particular robot, one of its arms is empty and the other occupied with a fresh or processed wafer. In Figure 5.8(a), for instance, executing a *rotate* action causes a transition from the central diagram to the diagram highlighted on the right. However, from this configuration it is possible to execute another rotate action back to the system's immediately preceding state (see Figure 5.8(b)).

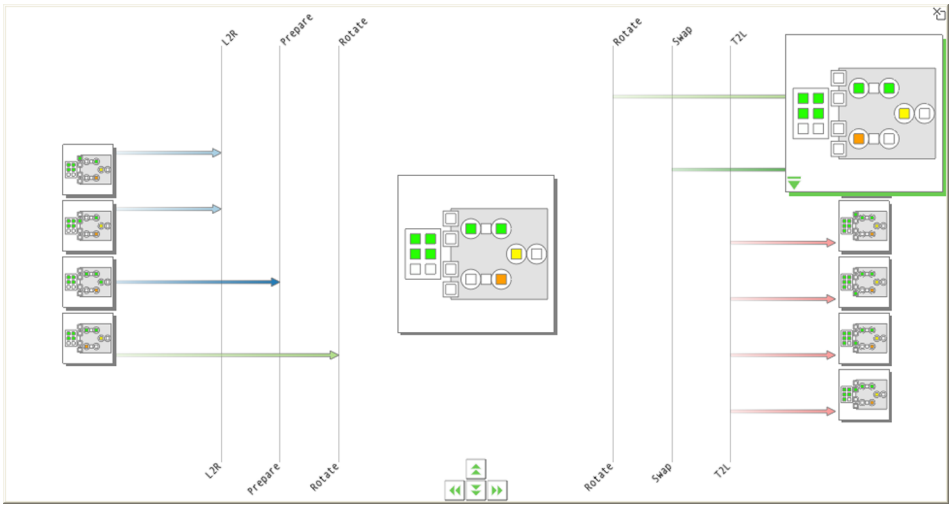
The author loaded a number of these scenarios into the inspection view and contacted the system analysts who had generated the transition graph of the wafer stepper system. Using these diagrams as starting point it was possible to quickly explain to them what had been found. The analysts were baffled. They had been studying this system for quite some time and were sure that it contained no errors. Yet, this behavior was unknown to them. By following up on the issue they were able to rectify it.

5.6 Case - paint factory

Analysts often study transition graphs to validate requirements (see Section 2.2). As part of his thesis work, an Eindhoven University of Technology graduate student in Mechanical Engineering studied the behavior of a paint factory system [27]. Figure 5.9 shows a laboratory setup of this system.



(a)



(b)

Figure 5.8: Wafer stepper livelock [48].

The system uses three reservoirs containing the colors red, yellow and blue. By mixing appropriate amounts of these primary colors the system produces various tints of paint. For this it uses a complex arrangement of valves, pumps and tanks. The student specified the behavior of these components and generated a transition graph consisting of 70,784 nodes, 294,625 edges and 33 node attributes. He also identified a number of requirements. One of these requirements is considered here; that only two of the three pumps that extract paint from the primary reservoirs can be in use at the same time.

Figure 5.9: Laboratory setup of a paint factory system.

Next, the student defined a diagram of the system. Using a prototype that implements the visualization technique described in this chapter, he was able to specify a layout that fits his conceptualization of the paint factory (see Figure 5.10). He positioned the three primary reservoirs at the top left of the diagram. He then parameterized the DoFs corresponding to their hue, opacity and rotation. Next, he linked these parameters to three attributes that describe the status of the three reservoirs' primary pumps. These attributes all have the domain $\{standby, in\_use\}$. The student also parameterized a number of other shapes to accurately capture his understanding of the system.

To check the requirement under consideration, the student clustered the transition graph by selecting the three attributes that describe the primary pumps. Figure 5.11 shows the clustering hierarchy. Here the entire transition graph has been aggregated to present it from the perspective of the three primary pumps. By applying some reasoning, the student could have verified requirements by interpreting the clustering hierarchy as a binary logical expression. However, the semantics of nodes inside the leaf clusters were immediately clear when they were appended with the user-defined diagram. Using this functionality, the student confirmed that no more than two of the primary pumps were colored and rotated in any of the diagrams in Figure 5.11.

The student made some additional discoveries. By stepping through the diagrams he learned that whenever a pump is engaged, at least one of its two valves are active (two squares beneath every reservoir pump). Furthermore, whenever the second (lower) valve is engaged, the valve leading into the mixing pump is also engaged (see the fourth, sixth and seventh diagram in Figure 5.11). In this way the student learned that there existed a strict dependence relation between these valves. As is often the case, he had specified the behavior of the different components separately (see Chapter 2). Only after considering the transition graph resulting from the interleaving of these specifications was this emergent behavior detectable. Nonetheless, despite the fact that this correlation is vital, it was not originally specified as a system requirement.

There are non-visual tools for formally verifying such requirements. However, they require users to formally specify the requirements first. The student was glad not to have to take this route because he had many requirements that he wanted to check before

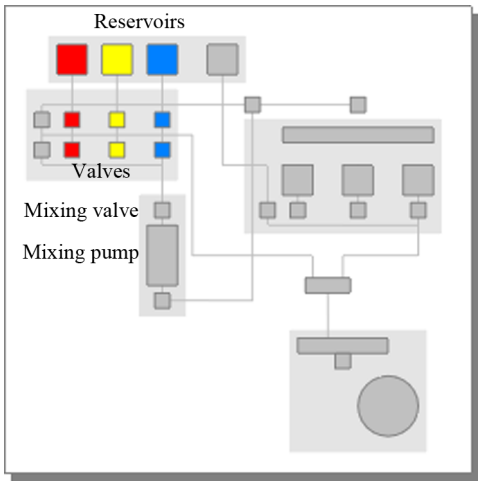


Figure 5.10: User-defined diagram of a paint factory system.

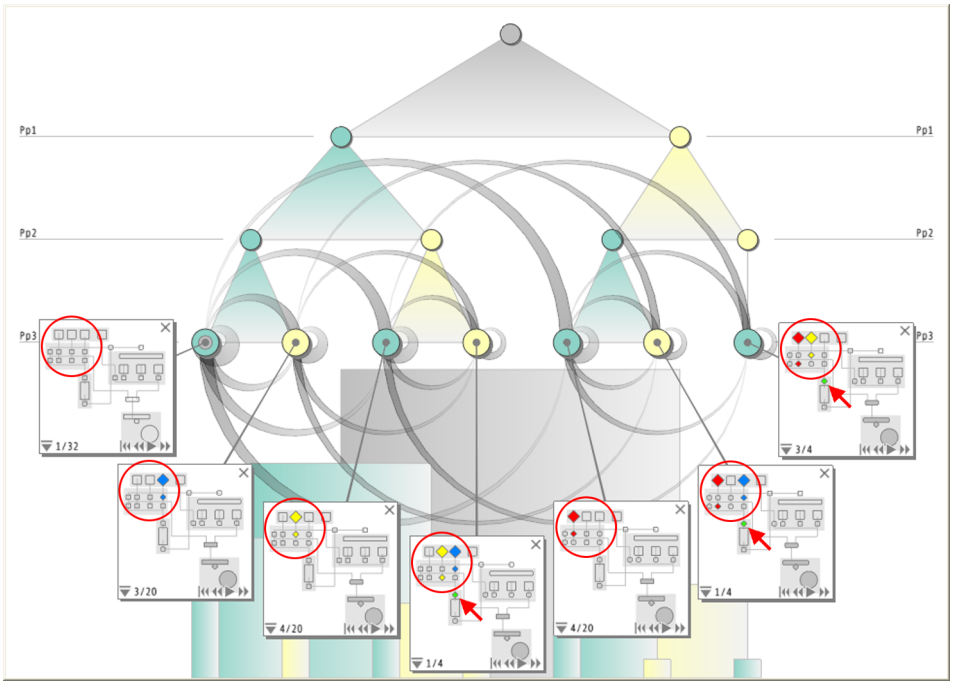


Figure 5.11: Paint factory verification.

proceeding with other tasks. Also, like the discovery he made, he did not know all the requirements in advance.

Finally, the student studied the edges that span between the different leaf clusters, such as those shown in Figure 5.11. By rolling over the corresponding arcs, he could consider the edge labels, which represent actions resulting in state transitions. These are displayed with tooltips. The student said that this allowed him to quickly gain a more accurate picture of the system's behavior. He also noted that communicating ideas about the system to his supervisors was much more effective using the visual representations described above.

5.7 Discussion

Like the other attribute-based techniques introduced in this dissertation, user-defined diagrams enable users to approach their problems in terms of knowledge they already have. In both the above case studies, interactive visual analysis made it possible to identify issues that may not have been noticed otherwise. As an aside, it was found that the act of defining diagrams served as a way to revisit and confirm what analysts already knew about the systems being studied. It also forced them to clarify their understanding of the configurations and constituent parts of these systems.

For state transition graphs, like the examples considered in the previous section, the domains of node attributes are usually discrete with a small cardinality. Because the parameterization of graphical objects is based on interpolation, however, real values can also be handled. This is shown in Chapter 6.

Although the small set of DoFs introduced in Section 5.3 are sufficient to illustrate the approach discussed in this chapter, more complex and expressive geometric DoFs are possible. On the other hand, non-geometric DoFs such as color and opacity were always the first to be utilized by users. In this regard, to enhance the efficiency of using custom diagrams, the user could be provided with a number of well-known preset color mappings. With such an approach, the hue and opacity control points would be automatically set when the user selects a color scheme. The user would not be required to edit the control points, although fine-tuning could be an option. This idea is investigated further in Chapter 6.

The simulation view has a fixed horizon of one time step. Although this is an improvement on current simulation tools, which only present a single node (current state) and a list of outgoing edges (transitions), there is room for further improvement. To enable users to get a better idea of the graph topology around an area of interest, it may be extended with a variable horizon. The idea of a variable horizon relative to a fixed point of interest is treated further in Chapter 7.

In the inspection view, user-defined text annotations could be useful. As shown by Shrinivasan and Van Wijk [77], such a feature assists users in documenting their discovery process. This may also assist in knowledge transfer to other stakeholders and in this way support collaborative visual analysis.

Another opportunity for improvement is to assist users to visually compare a large number of diagrams for similarities and differences. Although overlaying transparent di-

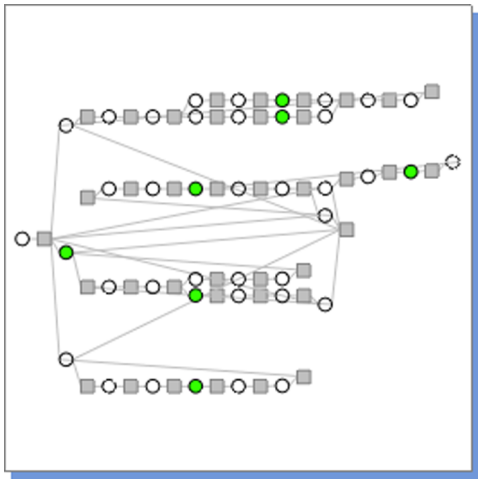


Figure 5.12: Petri net representation of a traffic controller [84].

agrams would suffice for comparing a small number of diagrams, finding a more scalable solution is not trivial.

The use of custom diagrams to study system behavior has greater scope than suggested by the two case studies presented in this chapter. For example, diagrams are not restricted to representing the physical layout of a system. Other established visual formalisms, such as Petri nets [57], can also be represented as parameterized diagrams.

To investigate this possibility, the author collaborated with analysts to derive Petri nets from state transition graphs in an automated fashion [84]. The places of these Petri nets were considered as derived node attributes that describe every node in the transition graph. Because a Petri net has an unambiguous visual representation, where places are represented by ellipses and events by rectangles, it is straightforward to generate a parameterized diagram to be used to investigate the corresponding state transition graph. In this way, a specific system state, represented by a node in the transition graph, corresponds to a particular marking of the Petri net.

This approach was used to check a number of requirements of a traffic light controller by using an automatically generated Petri net as a diagram. Figure 5.12 shows one marking of this Petri net. The circles in this diagram represent different node attributes. These attributes can assume one of two values and consequently the circles are either white or green. This illustrates the flexibility of parameterized diagrams and the prospect of annotating visualizations of state transition graphs with automatically generated diagrams. This suggests that the approach discussed in this chapter can be generalized by, for example, using the many variants of UML diagrams, which also have clearly defined semantics.

Finally, user-defined diagrams could be used to annotate other visualizations such as topology-based representations of state transition graphs. For instance, the approach by Van Ham et al. [25], discussed in Chapter 2, could be interactively annotated with custom

diagrams to assist users in relating interesting topological aspects to node attributes. In the next chapter, the flexibility of custom diagrams is illustrated by enabling users to interactively annotate system trace data.

User feedback for the visual analysis of state transition graphs with custom diagrams has been positive. One incident was particularly encouraging. Using a prototype developed for the technique described here, a problem that had been identified for a particular transition graph was presented to an audience that had no prior exposure to the data set. Before the author could explain why the problem occurred, one of the audience members exclaimed, "I have it!". He proceeded to explain exactly what the problem was and concluded by saying that, "diagrams make it obvious".

Chapter 6

Multiple Views on Traces

The type of data considered in this chapter differs from, but is closely related to, state transition graphs. A new method is presented for the visual analysis of multivariate system traces. This method combines three perspectives: a schematic diagram, time series plots and a state transition graph. After it is shown how these perspectives are related, their integration into a single visualization technique is discussed. The combination of the three perspectives provides analysts with a rich analysis interface. This enables users to gain significant insight into system behavior. The advantages of the approach are illustrated by providing a real-world case.

6.1 Rationale

In many engineering domains analysts study execution traces to understand the behavior of complex systems. A popular approach for such analysis is to generate traces by simulation [72]. As discussed in Section 5.1.2, the input for simulation is a mathematical behavioral model for the system being considered. Using this behavioral description as input, automated tools generate a finite sequence of consecutive system states. Such a system trace can be thought of as a single path in the system's transition graph and expresses its behavior over a finite interval of time.

This chapter introduces a technique for the interactive visual analysis of system traces. For this, three different perspectives that can be taken on trace data are identified. A different visualization is designed for each and these are integrated into a single correlated solution. The technique extends some of the methods introduced in previous chapters.

6.2 Data perspectives

System traces are closely related to state transition graphs. Consequently, the states in such traces are similar to the nodes that represent system states in transition graphs. A trace is defined as follows:

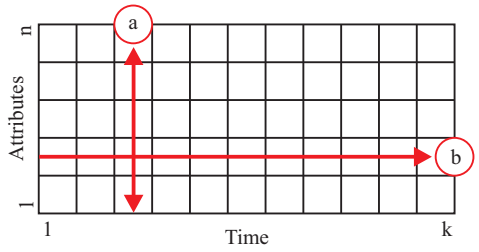


Figure 6.1: Common perspectives on multivariate system traces.

System trace. Let $D_1, ..., D_n$ be discrete or continuous domains. A system trace T is a finite sequence $s_1, ..., s_k$ where $s_i \in D_1 \times ... \times D_n$ is the state at time steps $i = 1, ..., k$.

Similar to state transition graphs, every state $s_i \in T$ consists of a vector of attribute values. That is, $s_i = (s_{i,1}, ..., s_{i,n})$ with $s_{i,j} \in D_j$ for $j = 1, ..., n$. As illustrated in Figure 6.1, the above definition suggests taking one of two perspectives on multivariate system traces. Although less evident, it also gives rise to a third useful perspective. These are discussed below.

6.2.1 Perspective 1 - schematic diagram

By focusing on the attributes axis in Figure 6.1(a) it is possible to get a snapshot of a system. That is, a point in time is considered as a function of the values that the state attributes assume.

As shown in Chapter 5, this perspective can be taken by mapping attribute values to a schematic diagram that represents the system being studied. The user first defines a diagram for the system and then interactively links attributes to the graphical properties of the shapes that make up this diagram. Based on the values assumed by a data attribute, properties such as color, position and size of corresponding shapes change. In Chapter 5 it was shown how this assists the user to understand the influence of attributes on system behavior. The work by Ribarsky et al. [66] and Van Wijk et al. [95], discussed in Section 5.1.2, also make a strong case for enabling users to analyze system behavior with diagrammatic representations.

6.2.2 Perspective 2 - time series plots

Diagram-based techniques, such as those mentioned in the previous section, resort to animation to show how state attributes change over time. However, since a sequence of frames is shown in rapid succession, it can be challenging to discover correlations between attributes (causality) and trends over the time axis [82]. The notion of overview and detail is also missing. By focusing on the time axis, as illustrated in Figure 6.1(b), it is possible to address these issues. Many examples of such an approach can be found in the time series visualization literature where the values assumed for a particular attribute are plotted against time [81].

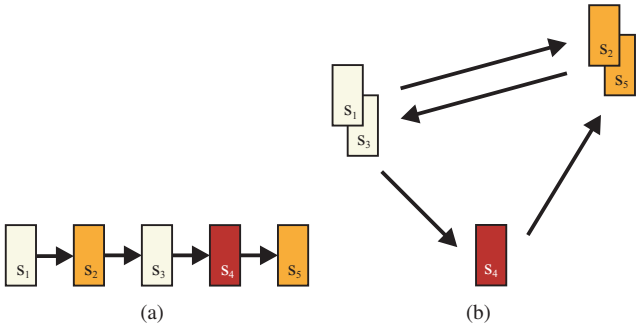


Figure 6.2: System trace as a graph.

Recent work on time series visualization has focused mainly on two issues: presentation and visual querying. Weber et al. [92] proposed circular layouts to deal with large data sets and to highlight periodic data patterns. Hao et al. [28] addressed large data sets with a hierarchical space-filling strategy. It assigns larger and more prominent display real estate to more important attributes. Hochheiser and Shneiderman [33] proposed query-by-example widgets for time series. These enable the user to visually specify queries based on which large sets of time series are filtered. Likewise, Wattenberg [89] developed a technique that allows the user to sketch a profile to retrieve similar time series.

The focus of the current chapter is on integrating time series visualization with other perspectives that can be taken on system traces. The presented approach is flexible enough, however, to be combined with other presentation and query techniques such as those discussed above.

6.2.3 Perspective 3 - state transition graph

The rationale behind studying system traces is often the following. If a finite sequence of states is representative for a system, then by understanding this (partial) behavioral description, it is possible to gain insight into the system as a whole.

It has been argued that schematic diagrams are good for showing the details of one or a few individual states. Also, time series plots are well suited for showing causality and for identifying patterns. However, none of these techniques present the user with a high-level representation of system behavior. To achieve this, a third perspective is proposed by considering trace data independently of time.

When time is discarded and an equivalence relation is defined on the states $s_i \in T$ a graph is generated (see Figure 6.2). More precisely, consider the relation $\equiv_G$ where $s_i \equiv_G s_l$ if and only if $s_{i,j} = s_{l,j}$ for $j = 1, ..., n$. Let V be the set of equivalence classes of $\equiv_G$ on T. Next, for all s_i let $e_i = ([s_i], [s_{i+1}])$ for $i = 1, ..., k-1$ where $[s_i]$ is the equivalence class of s_i. Now, let E be the set of all e_i and the graph $G = (V, E)$ is generated where V contains its vertices and E its directed edges. Note that this graph represents a partial multivariate state transition graph of the system being studied [4].

The above equivalence relation enables the user to cluster trace data based on the val-

ues assumed for the different state attributes. As shown in Section 6.3.3, this allows for an abstract representation of the data. Other researchers have suggested similar approaches. Lin et al. [45] used augmented suffix trees to summarize time series structure. By interacting with this tree, the user is able to identify and zoom into interesting parts of the time series. Van Wijk and Van Selow [96] proposed a multi scale technique that clusters time varying data hierarchically. This approach combines a calendar based representation with traditional line plots.

Users associate meaning with the state attributes found in system traces, just like they associate meaning with the node attributes of transition graphs. Consequently, clustering reduces the complexity of system traces in a semantically rich way. It leads to a multivariate graph, where every node represents a particular combination of attribute values and each edge a possible transition.

6.3 Multiple views

A visualization technique that integrates the three perspectives introduced in Section 6.2 was developed. To validate the approach a prototype was built to which reference is made in the remainder of this chapter. Figure 6.3 shows the three different views that were developed. To use these, the prototype provides the user with an overview of all attributes that describe the system states (see Figure 6.3(a)). For attributes with continuous domains, the user is provided with a range within which all values fall. When an attribute with a discrete domain is selected, the user is presented with a list of its domain values (Figure 6.3(b)).

Like the technique discussed in Chapter 5, a number of operations can be performed on the attributes and on their domains. The position of any attribute can be adjusted in the list and attributes can be duplicated. The ordering of attributes has an influence on the different visualizations, as shown in the following sections. Discrete domain values can also be reordered.

The approach presented in Chapter 5 is extended so that continuous domains can be discretized by performing classification, or binning. Classification based on three popular algorithms is provided: equal intervals, quantiles and mean-standard deviation [29]. This can be easily extended with other methods. For example, symbolic approximation techniques such as SAX [46] will also fit into the framework presented here.

6.3.1 User-defined diagram

To define diagrams that fit their conceptualization of a system, users switch to edit mode (see Figure 6.4). As with the technique introduced in Chapter 5, a graphical editor is provided for constructing diagrams from simple primitives such as ellipses, rectangles, lines and arrows (see Figure 6.4(a) and Figure 6.4(b)). Every shape also has seven Degrees of Freedom (DoFs): x-position, y-position, width, height, rotation (about a movable pivot), hue and opacity (Figure 6.4(c) and Figure 6.4(d)). To edit a DoF, a range is defined and an attribute is linked to it (Figure 6.4(e) and Figure 6.4(f), also see Section 5.3).

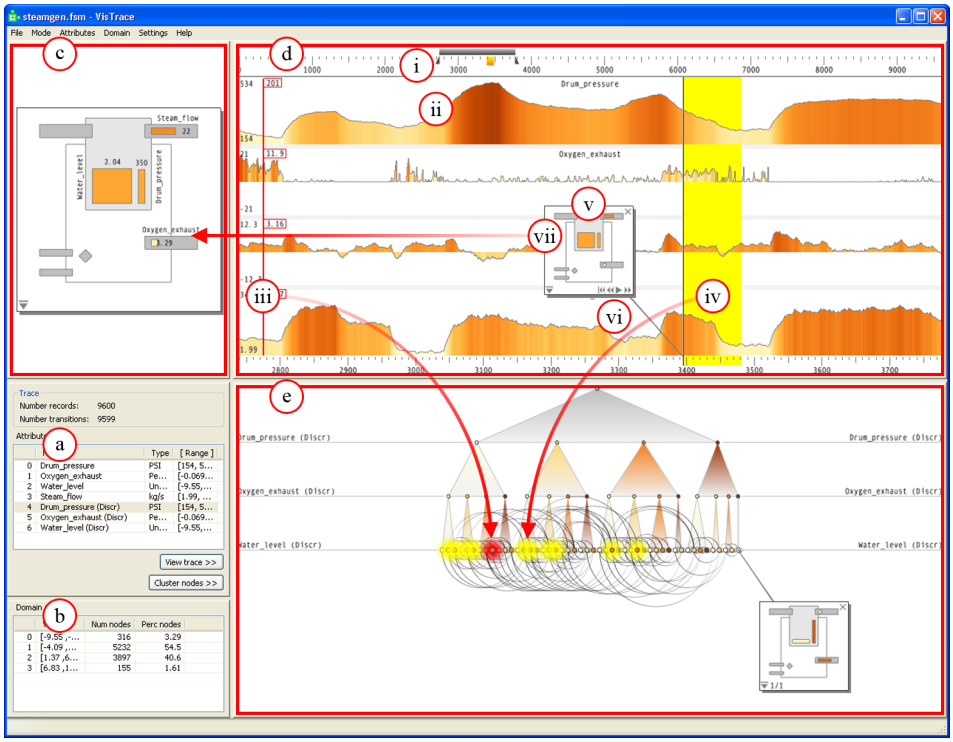


Figure 6.3: Visualization of multivariate system traces.

As mentioned, to broaden the scope of the technique presented in Chapter 5, attributes with continuous domains can also be handled. Because the diagrams representing particular states are obtained by mapping attribute values to the ranges of DoFs using interpolation, this required a straightforward extension of the initial approach discussed in Section 5.3.

The technique presented in Chapter 5 also did not support text. However, users expressed a requirement to annotate their custom diagrams with both static and dynamic text. To cater for this, another primitive called a text anchor is introduced (see Figure 6.4(g)). A text anchor behaves just like a normal shape. For instance, it can also be rotated and it has the same seven DoFs as any other shape. However, for every one of its DoFs that have been associated with a state attribute, the user can select whether to display the attribute name and value, the attribute name only, or the value only.

6.3.2 Time series plots

After switching back to analysis mode, the user can select and load a subset of state attributes into the time series view (see Figure 6.3(d)). To handle large data sets, two time scales are provided. At the top there is a global time scale representing the complete

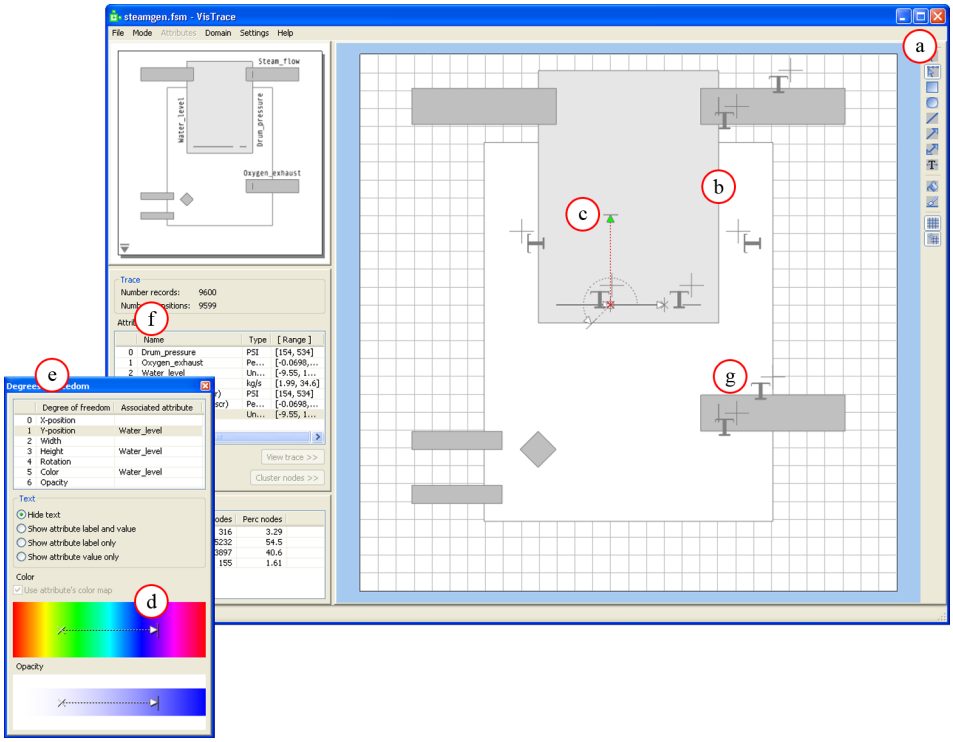


Figure 6.4: Integrated graphical editor.

system trace (Figure 6.3(d)(i)). The user has a current view within the global time scale that is represented by a beveled slider handle.

The current view is visualized at a magnified resolution on the local time scale (see Figure 6.3(d)(ii)). The attributes that have been loaded are visualized as a number of parallel time series plots. Time is mapped to the x-axis from left to right. The values that are assumed for the attributes are normalized and mapped to the y-axes of these plots.

By sharing the same time axis, the user is able to compare the behavior of the different attributes. By dragging the slider in the global scale the content of the local time scale can be adjusted. The user can also increase or decrease the length of the slider. This results in respectively zooming out or zooming in on the data in the local time scale. The vertical order of the plots corresponds to the order of the attributes in the selection (Figure 6.4(a)). As mentioned before, the user can adjust this ordering.

Basic interactive features are also offered to the user. For example, when the mouse is rolled over the local time scale, the current position is highlighted and the values of all selected attributes are shown with a tool tip (see Figure 6.3(d)(iii)). As mentioned in Section 6.2.2, extending the time series view with other interaction techniques is possible and will not interfere with the framework presented in this chapter.

6.3.3 State transition graph

In Section 6.2.3 an equivalence relation was introduced that allows for considering a multivariate trace as a directed graph. To make this more flexible, the user is enabled to select and consider a subset of the attributes that describe every state. That is, for any $s_i, s_l \in T$ it holds that $s_i \equiv_G s_l$ if and only if $s_{i,j} = s_{l,j}$ for $j \in I$ where $I \subseteq \{1, ..., n\}$.

The above enables the user to perform attribute-based clustering as discussed in Section 4.2. The user first selects a subset of state attributes and then clicks on the cluster button (see Figure 6.3(a)). The entire set of states $s_i \in T$ is taken as the root of a clustering hierarchy. The set is then partitioned based on the different values assumed for the first attribute. This results in a number of child clusters branching from the root. Every resulting cluster now contains a disjoint subset of the states in the root cluster. Next, each of these clusters is sub-partitioned based on the second attribute, resulting in a third level in the hierarchy (Figure 6.3(e)).

Clustering in the above fashion requires attributes with discrete domains, as noted in Chapter 4. This is one of the reasons for providing the classification algorithms discussed at the start of this section in order to deal with continuous domains.

As outlined in Section 6.2.3, clustering generates a graph. The nodes of this graph are the leaf nodes of the clustering hierarchy and contain sets of individual states. The edges are bundles of transitions that connect adjacent states in the original trace. Taking a similar approach to the one introduced in Chapter 4, the transition graph is visualized as an arc diagram [90]: edges are represented as semi-circular arcs that span from the source to the target cluster. The thickness of an arc is proportional to the number of bundled transitions. The arc diagram is positioned on the leaf clusters of the clustering hierarchy (see Figure 6.3(e)). Since clusters are positioned at equal horizontal intervals, arcs of equal height suggest repeated behavioral patterns. Arc orientation encodes transition direction and is interpreted clockwise.

6.3.4 Integration

In Sections 6.3.1–6.3.3 it was explained how the three perspectives on multivariate trace data, introduced in Section 6.2, were implemented. This results in three different views on the same trace data: a schematic diagram, a time series plot and a transition graph (see Figure 6.3(c)–(e)). For the different views to be complementary they were tightly integrated with each other, as discussed below.

The user can select individual states or ranges of states in the time series view. Consequently, the clusters in the graph view that contain these states are highlighted (see Figure 6.3(d)(iv)). Similarly, when a cluster is selected in the graph view, all states in it are highlighted in the time series view.

By double clicking a state in the time series view, or a cluster in the graph view, these visualizations can be extended with diagrams. They are annotated with callouts showing the user-defined diagram (see Figure 6.3(d)(v)). All parameterized DoFs are calculated as outlined in Section 6.3.1 and Section 5.3.

Using icons on the diagrams (see Figure 6.3(d)(vi)), the user can animate or step over a selected range in the time series view. To maintain context, the time series and cluster

views are updated to highlight the current state while animating. When the user moves the mouse over a diagram in the time series or graph view, it is shown at a larger magnification in the inspection view (see Figure 6.3(d)(vii)).

In terms of integrating the different views, a number of lessons have been learned. These should also be considered in light of the results presented in Chapter 5, where incorporating user-defined diagrams with other visualizations was first introduced.

To create a sense of coherence and to more clearly correlate views, the design has evolved to using color consistently. For instance, in initial experiments color coding was not used in the time series view. However, users repeatedly asked whether it was possible to map values to the same colors used in the diagram and graph views. Consequently, the prototype was adapted to default to using the same color mapping in all three views. In the current chapter, a perceptually ordered white to brown scheme is used. In cases where the user deems it necessary, it is possible to override this by specifying a different color map or a path to interpolate over (see Section 6.3.1).

Another insight deals with the layout of the views. Initially, it was reasoned that users would want a high level overview of system behavior as the most prominent visualization with all other views in the periphery. This was translated into positioning the graph view centrally with diagrams and time series views playing a supportive role and shown on user demand.

However, users much preferred having the time series view as their main point of reference. Next, they wanted to inspect parts of this view with diagrams. This is supported with the capability to show diagrams overlaid on the time series view. Users also wanted to inspect diagrams in detail, though. Since there is a trade-off between obscuring visualizations with diagrams and the effort needed to switch to a separate view, overlaid diagrams are kept small and detail is shown in a larger linked view. By positioning the graph view below the time series view, it plays the more supportive role preferred by users.

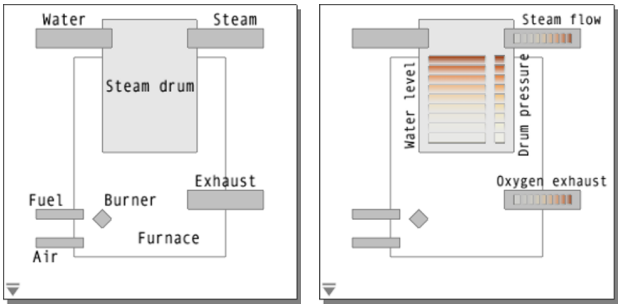
6.4 Case - industrial steam generator

A case is now presented to illustrate how the technique introduced in this chapter can assist users with the analysis of multivariate system traces. An industrial steam generator is considered. The data set was generated by simulation from a mathematical model of a boiler at Abbot Power Plant, Illinois, USA [56].

The steam generator system consists of two parts. First, there is the furnace (see Figure 6.5(a)). Fuel and air are injected into the furnace and are consumed by a burner. Combustion generates heat and flue gases. The latter are sucked out via an exhaust.

The second part of the installation is a steam drum that sits at the top of the furnace, above the burner. When heat is generated the drum gets hot. The water that is fed into it also heats up, reaches boiling point and starts to evaporate. The steam produced in this fashion increases the pressure inside the drum and consequently gushes out at the top right. This pressurized steam is used for power generation and heating.

The trace that describes the behavior of the above system consists of 9,600 states that are described by four attributes. These are:



(a) Different components. (b) Parameterized components.

Figure 6.5: Industrial steam generator.

- *Drum pressure.* The pressure inside the steam drum.
- *Oxygen exhaust.* Excess oxygen in the exhaust gases.
- *Steam flow.* The rate of steam flow from the steam drum.
- *Water level.* The water level inside the steam drum.

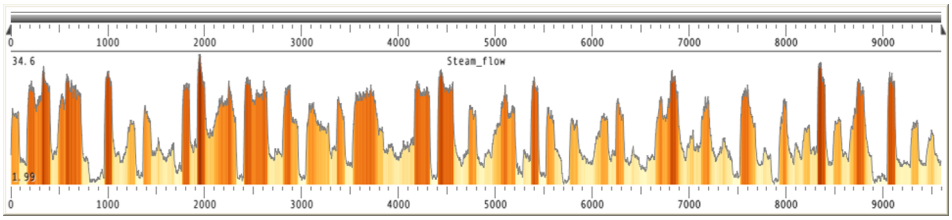
Using the prototype's editor, a diagram that represents the steam generator was composed (see Figure 6.5(b)). The furnace is represented by the larger white rectangle and the drum by the smaller gray rectangle. The pressure inside the drum is encoded with the height and color of a bar toward its right. The water level inside the drum is linked to a bar toward its left. Similar to drum pressure, it increases in height proportional to the water level. The elongated rectangle at the top right of the diagram represents the steam output valve. The color and horizontal length of a bar inside it are parameterized to represent steam flow. Finally, the rectangle toward the lower right is the exhaust. Similar to steam flow, it contains a bar that increases in length and changes color proportional to the percentage of excess oxygen in the exhaust.

6.4.1 Steam flow

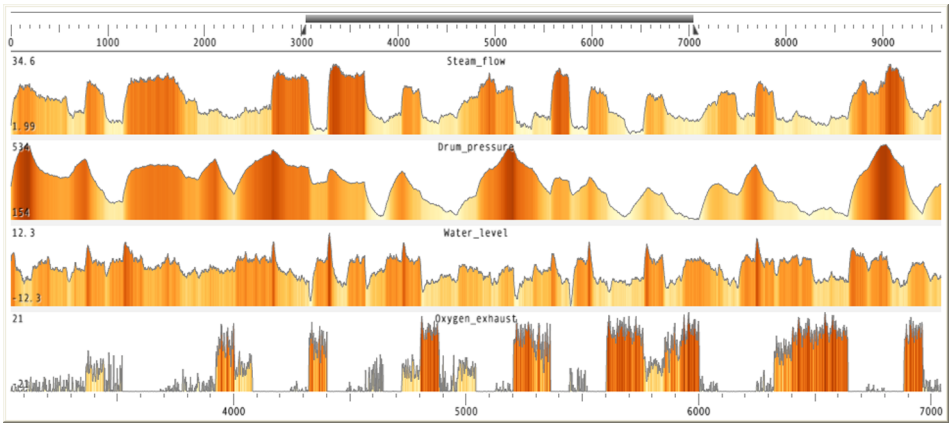
Suppose users are interested in those states where steam flow is high. To identify these states using the diagram alone, they would have to step through 9,600 frames. However, using the time series view, this task is much easier. To start, the *steam flow* attribute is loaded into the time series view.

Zooming out generates the result shown in Figure 6.6(a). It can be seen that the steam flow rate fluctuates quite a bit over time. It may even be argued that there are some repeated patterns visible in the fluctuation. It is possible to zoom in on the data and investigate these further, but perhaps more interestingly, the user may want to determine whether there are correlations with the other attributes that describe the data.

The result of loading all four attributes into the time series view is depicted in Figure 6.6(b). The attributes were first reordered: *steam flow*, *drum pressure*, *water level*,



(a) Steam flow in isolation.



(b) Steam flow in combination with other attributes.

Figure 6.6: Steam generator, time series view.

oxygen exhaust. Again the user can zoom in, zoom out and move to different parts of the time series to look for local and global patterns.

It is possible to identify correlations in this fashion. That is, by considering the parallel time series plots, the user can identify dependencies between them. For instance, it appears as if *steam flow* is high whenever the drum pressure and water level are high and when there is little excess oxygen in the exhaust gases. By double clicking on a few of these states, diagrams such as the one in Figure 6.7 are shown. These seem to support this observation.

To confirm the above hypothesis, it would be useful to identify and select all states where *steam flow* is high. The user can zoom in on individual states where this holds, but it is also possible to identify such states using attribute-based clustering (see Section 6.2.3).

6.4.2 Drum pressure

To perform clustering, the continuous domains found in the data can be classified to get a number of discrete intervals (see Section 6.3). The previous observation suggests that a high steam output is always coupled with high drum pressure and the aim is to find out whether this hypothesis really holds. For instance, do states exist where steam flow is at a maximum, but where drum pressure is not too high?

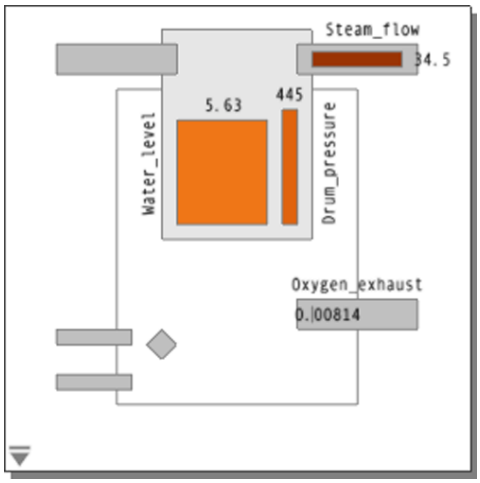


Figure 6.7: High steam flow.

It may be argued that these are ideal states for the system to be in since steam output is high but there is no risk of generating too much pressure in the drum. First, both these attributes' domains are classified into four classes using the equal intervals method [29]. This results in partitioning the domains into four equal intervals, all representing 25% of the total range.

Consequently, after clustering on *steam flow* and *drum pressure* the user is presented with the results in Figure 6.8. From the figure it can be seen that the earlier hypothesis does not hold: the two clusters that have been annotated with diagrams show that even when steam flow is in the top 25% of its range, there are states where drum pressure is in the range 25–50% and 50–75%. This is shown in two ways. First, the two leaf clusters third and second from the right (representing drum pressure of 25–50% and 50–75%, respectively) would not exist if this were not the case (see Figure 6.8(a)); the second level cluster corresponding to steam flow in the range 75–100% (right-most) would only have a single child node. Second, consider the height and color of the bar encoding *drum pressure* in the two diagrams (see Figure 6.8(b)). In both, this bar does not reach its maximum height and does not assume a dark brown color (compare to Figure 6.5(b)).

6.4.3 Water level

To produce steam the drum has to contain water. When the domain of *water level* is classified in the same way as above (four equal intervals representing 25% of its range) and attribute-based clustering is then performed (*steam flow*, *drum pressure*, *water level*), this generates the result shown in Figure 6.9.

Note, however, that the aim is actually only to distinguish between (1) those states where there is too little water and (2) those states where there is enough. This allows for the complexity of the visual analysis to be reduced even further. To do so, the sec-

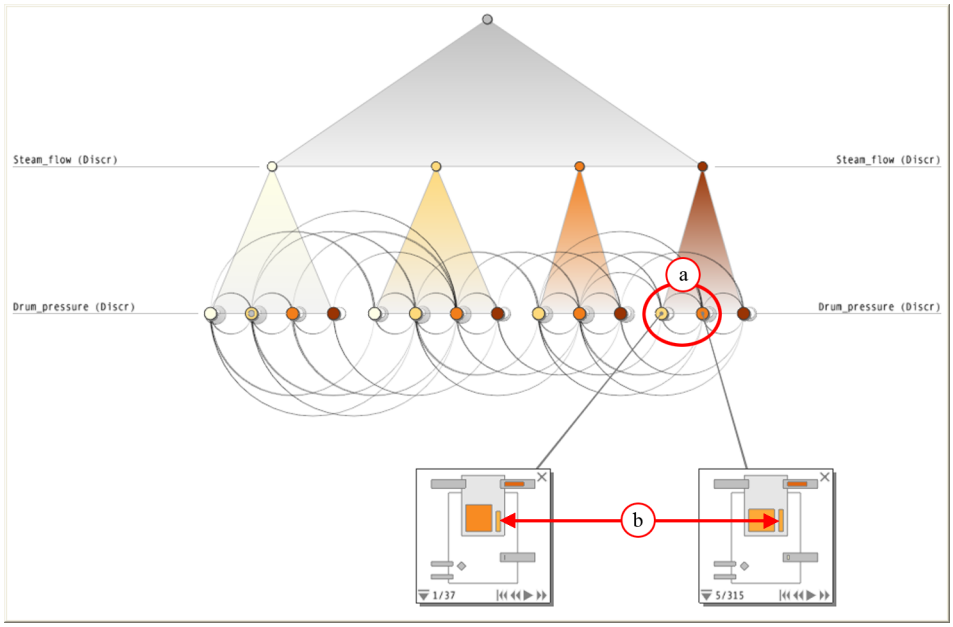


Figure 6.8: Drum pressure versus steam flow.

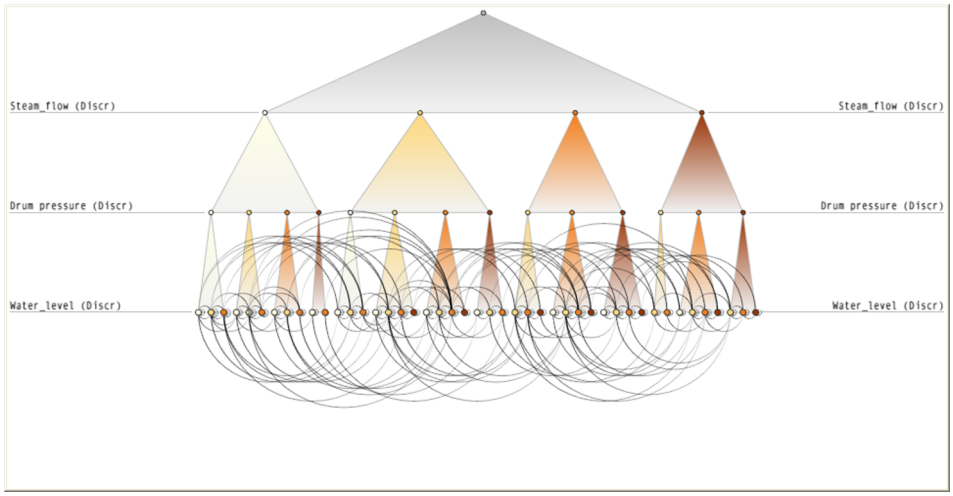


Figure 6.9: Steam flow, drum pressure and water level.

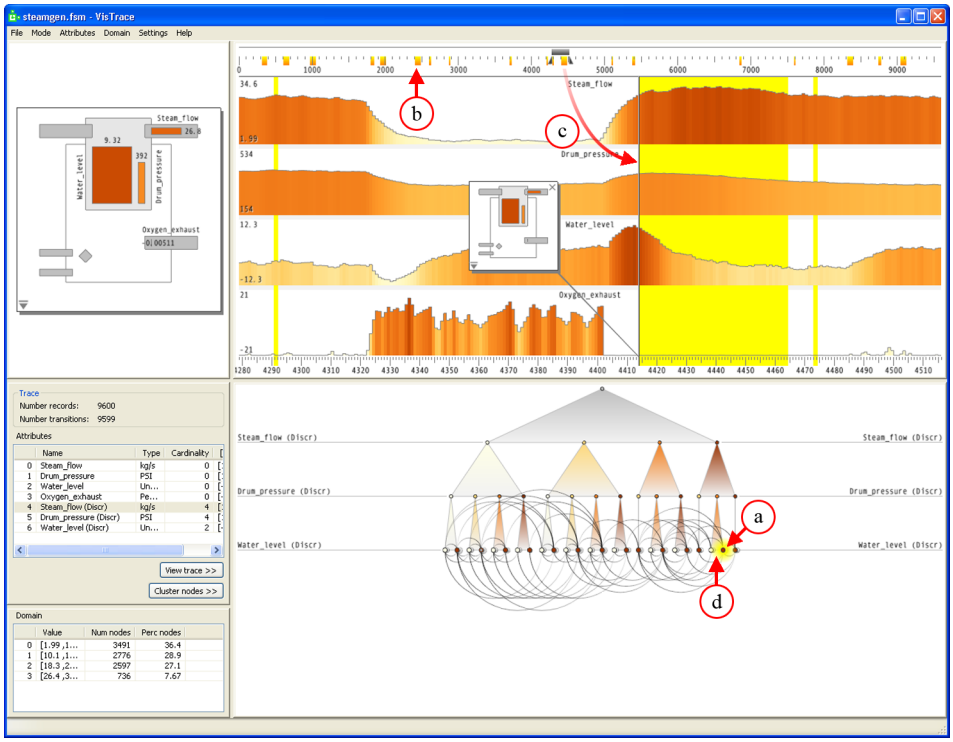


Figure 6.10: Multiple views on an industrial steam generator.

ond, third and fourth intervals of *water level* are grouped together (see Section 6.3). That is, the following mapping is defined: 0–25% → *too low*, 25–100% → *ok*. When the same attribute-based clustering as above is reapplied, the user is presented with the results shown in Figure 6.10 (compare the number of arcs and leaf clusters with those in Figure 6.9).

Using the clustering results, all states can be identified where the following conditions hold: steam flow is in the top 25% of its range, drum pressure is in the range 50–75% of its range, and the water level is not in the bottom 25% of its range. These states are all clustered in the leaf second from right in the clustering hierarchy (see Figure 6.10(a)). When this cluster is selected, all individual states in it are highlighted in the time series view. This holds for both the global (see Figure 6.10(b)) and the local time scales (see Figure 6.10(c)).

Zooming in to see more detail, it is possible to learn more about those states where there is a high rate of steam flow, an acceptable drum pressure and sufficient water. For example, the states preceding the highlighted section of Figure 6.10(c) show that drum pressure has been reasonably constant for a stretch of time while the excess oxygen in the exhaust recently dropped to its minimum.

This observation is not that obvious to interpret. The following reasoning may be

applied. There will be little excess oxygen in the exhaust gases when (1) little air enters the furnace, or (2) the burner efficiently converts the air and fuel mixture into thermal energy by combustion. The first hypothesis can be ruled out since this would also result in a drop in drum pressure and steam flow, which is not the case (see Figure 6.10). The second explanation seems more feasible. This conclusion can be verified by navigating to other highlighted states where the steam flow is high, and where there is sufficient pressure and water.

A final observation regarding Figure 6.10 can be made. The transitions, represented by arcs and interpreted clockwise, show that there are no transitions from states where there is a high rate of steam flow and sufficient pressure and water to states where there is very little water, and vice versa. This is seen by observing that there are no arcs between the highlighted cluster and the cluster immediately to its left (see Figure 6.10(d)). The latter cluster represents states where steam output is high, there is sufficient pressure, but very little water.

6.5 Discussion

By providing different views on multivariate system traces, the user is enabled to take different perspectives on the data. This implies switching between different conceptualizations of a problem. The point of departure is that this assists the user in making cognitive switches, resulting in a deeper understanding of the data being studied.

Currently, a different linked view is offered for every perspective introduced in Section 6.2. A promising opportunity for future work is to investigate how these views could be integrated more seamlessly. This could imply combining views and could also lead to new user interaction methods. The ability of users to show diagrams on demand in the time series and diagram views is a first step in this direction.

The results presented in the current and the previous chapter suggest possible extensions for custom diagrams. For example, users can only define a linear range for the geometric DoFs (position, size and rotation). However, there may be problems that require more complex paths. Also, the ability to specify a hierarchy of shapes and DoF dependencies could allow for very intricate diagram behavior to be defined. Alternatively, the user could be provided with a number of predefined widgets, such as various common visualizations. By only linking one or more attributes with such a widget the user could be spared the effort of assembling them from various shapes. Investigating the trade-off between simplicity and expressivity in user-defined diagrams is an interesting theme for future work.

The discussed approach may be used earlier in the system design cycle. For example, it could be applied during the development of the mathematical models that are used as input for simulation (see Section 6.1 and Section 5.1.2). In this way, in a fashion analogous to debugging, many problems could be identified before the model is finalized. This is opposed to only inspecting the results after the model has been completed.

The technique introduced in this chapter may also be applied to compare simulated and actual traces (performance measurements from implemented systems) of the same system. This could be achieved by considering the measured and simulated attributes in

parallel.

In this chapter a visualization approach based on different views on the same data set was presented in the context of system analysis. Furthermore, the examples provided in Section 6.4 have natural physical representations. This is an important class of problems that is frequently encountered.

However, it is argued that the approach presented here is flexible enough to deal with other application domains or time series data, in general. More general applications of the technique could be supported by classification algorithms that take the distribution of attribute values along the time axis into account, as remarked in Section 6.3. For example, because the discretization algorithms currently offered only consider attribute domains, a high spike and a high plateau would be grouped into the same class. However, techniques such as SAX [45] distinguish between such patterns.

The technique could be easily extended to offer additional classification methods and to support user interaction with classification results. For example, when classifying a numeric domain the user may want to employ statistical methods first and then manually update intervals to fall on the nearest multiple of 10.

Related to the above, are the linearity assumptions that are currently made. For example, linear interpolation is used to calculate the graphical appearance of shapes in the diagram view. For problems such as those presented in Section 6.4 this is reasonable. However, to deal with problems where the distribution of values is skewed, more flexibility may be an advantage. Logarithmic scales could be a first step.

It may also be an advantage to treat different types of attributes (nominal, ordinal, divergent, and so forth) in different ways. The framework of three conceptual perspectives on the same data set is flexible enough to allow for such extensions. For example, the user could be provided with alternative presentations to the traditional time series plots discussed here (see Section 6.2.2). This also holds for the graph view. Although there has been opted for combining a clustering hierarchy with an arc diagram, such that the positions of the nodes have meaning for the user, other presentation techniques could be used.

Chapter 7

Querying Nodes and Edges

In previous chapters a number of state transition graph visualization methods, which focus on node attributes, were presented. To enable users to also detect relations and patterns in terms of data that describe edges, this chapter presents a technique where, in addition to node attributes, data associated with edges play a more central role. Nodes and edges are clustered based on associated data. Via direct manipulation users can interactively inspect and query the graph. Questions that can be answered include, "which edge types are activated by specific node attributes?" and, "how and from where can I reach specific types of nodes?" To validate this approach it is contrasted with current system analysis practice. Several examples of where the method was used to study transition graphs that model real-world systems are also provided.

7.1 Rationale

The meaning associated with node attributes is a useful point of departure for the visual analysis of state transition graphs, as previous chapters have shown. However, users often seek to also understand such graphs in terms of edge labels. To further validate these claims, five domain experts were interviewed. These were system analysts who regularly analyze state transition graphs (see Section 2.6.1). The aim was to find out three things: how they reason about their data, how important they regard questions related to node attributes and edge labels, and how they currently approach such questions.

7.1.1 Reasoning

Analysts brought their own data to the interview. They were then asked what these graphs model, what they find interesting about the data and what else they would like to learn from it. In all cases, node attributes and edge labels were central in discussions that followed.

For example, one analyst was studying the behavior of a traffic regulation system consisting of a number of traffic lights at a complex street intersection. Every light can be in one of three phases (red, amber or green). These phases are modeled by node

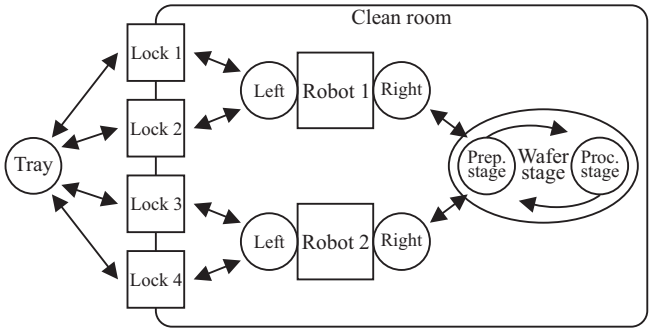


Figure 7.1: Wafer stepper.

attributes. There are also messages that cause phase changes, modeled by labeled edges. The analyst was interested to see whether certain dangerous scenarios are possible. For instance, certain lights should never be green at the same time. When this occurs it is important to identify the responsible messages.

7.1.2 Questions

Analysts were also introduced to a graph that describes a generalization of the behavior of an industrial wafer stepper [30], first introduced in Chapter 4. This system produces integrated circuits from silicon wafers (see Figure 7.1). On the left is a tray that can hold six wafers. Unprocessed wafers enter the system here and then move to one of four locks. Next, one of two robots takes a wafer from a lock using its left arm, rotates 180° and places the wafer on the wafer stage. Here it is first prepared on the preparation stage and then processed on the processing stage. To exit, wafers follow a similar path in reverse.

As noted in Chapter 4, the transition graph $G = (V, E)$ that describes the wafer stepper consists of 55,043 nodes and 289,443 edges. It has 15 node attributes, including:

- *prep_stage.* Contents of the preparation stage.
- *proc_stage.* Contents of the processing stage.
- *robot_1_rgt.* Contents of the right arm of robot 1.

The graph also has a set L of 26 edge labels, including:

- *Prepare.* Prepare the wafer on the preparation stage.
- *Process.* Process the wafer on the processing stage.

To get users' opinions on questions related to state transition graphs, they were presented with three questions about the wafer stepper. They did not have to answer these. Instead, they were asked to give an indication of how relevant they thought the following questions were:

- *Question 1.* Which outgoing edges are possible from nodes where robot 1's right arm is empty and both the preparation and processing stage contain prepared wafers? Which target nodes do these edges lead to?

- *Question 2.* Which nodes can be reached via edges labeled *Process* from nodes where robot 1's right arm is not empty and the preparation and processing stage both contain prepared wafers?

- *Question 3.* Which source and target nodes share edges with the label *Prepare*? For the source nodes, which different combinations of the attributes *prep_stage*, *proc_stage* and *robot_1_rgt* are possible? Do all nodes where the attributes take these values have outgoing edges labeled *Prepare*?

Feedback was unanimous: questions such as the above are important for understanding system behavior. First, being able to answer these and similar questions would lead to a better intuition about the data and the behavior it models. Second, specific system requirements can be checked this way. Furthermore, answers to more complicated questions could be derived if it were possible to answer many such questions quickly and effectively. All analysts, like the one studying the traffic regulation system, had similar questions about their own data.

7.1.3 Strategy

To learn how analysts would approach the above questions, they were asked to propose strategies to find the answers. Except for one analyst (who would convert to a different but semantically equivalent formal representation, not guaranteed to be found) all others proposed to query the graph using a combination of set theory and logic. That is, find those subsets of nodes and edges for which certain predicates hold. This implies formalizing the three questions as follows:

Question 1. Let L_{Q_1} and V_{Q_1} be the edges and target nodes of interest. Then

$$L_{Q_1} := \{l \in L | \exists v \in V_1 . \exists v' \in V . (v, l, v') \in E\}$$

and

$$V_{Q_1} := \{v' \in V | \exists v \in V_1 . \exists l \in L . (v, l, v') \in E\}$$

where

$$\begin{aligned} V_1 := \{v \in V | & v.prep\_stage = prepared \wedge \\ & v.proc\_stage = prepared \wedge \\ & v.robot\_1\_rgt = empty\}. \end{aligned}$$

Question 2. Let V_{Q_2} be the target nodes of interest. Then

$$V_{Q_2} := \{v' \in V | \exists v \in V_2 . (v, Process, v') \in E\}$$

where

$$\begin{aligned} V_2 := \{v \in V | & v.prep\_stage = prepared \wedge \\ & v.proc\_stage = prepared \wedge \\ & v.robot\_1\_rgt \neq empty\}. \end{aligned}$$

Question 3. Let V_{Q_3} and V'_{Q_3} be the source and target nodes of interest. Then

$$V_{Q_3} := \{v \in V | \exists v' \in V . (v, Prepare, v') \in E\}$$

and

$$V'_{Q_3} := \{v' \in V | \exists v \in V . (v, Prepare, v') \in E\}.$$

Now, let D_{Q_3} be the different combinations of values assumed by *prep_stage*, *proc_stage* and *robot_1_rgt*. Then

$$\begin{aligned} D_{Q_3} := \{(x, y, z) | & \exists v \in V_{Q_3} . \\ & x = v.prep\_stage \wedge \\ & y = v.proc\_stage \wedge \\ & z = v.robot\_1\_rgt\}. \end{aligned}$$

To answer the last part of the query, it has to be determined if the following equality holds.

$$V_{Q_3} = V''_{Q_3}$$

where

$$\begin{aligned} V''_{Q_3} := \{v \in V | & \exists (x, y, z) \in D_{Q_3} . \\ & x = v.prep\_stage \wedge \\ & y = v.proc\_stage \wedge \\ & z = v.robot\_1\_rgt\}. \end{aligned}$$

7.2 Nodes and edges

From the results presented in Sections 7.1.1–7.1.3, it is concluded that there is a need for graph analysis techniques where node attributes and edge labels play a central role. To the author's surprise, analysts who were interviewed did not know of any tools, apart from model checkers [18], that enable this type of analysis (see Section 2.2). This is remarkable, especially considering the overwhelming agreement among analysts on the relevance of the questions introduced in Section 7.1.2.

For model checking, questions have to meticulously formalized, like the queries above. This requires a significant investment of time and effort and all users suggested alternative approaches. These included custom scripts or even adapting the formal specifications from which their data were generated. This implies hard-coding queries plus additional debugging, which is not an efficient approach.

All the techniques presented in this dissertation so far and all related work discussed in Section 2.5.2 enable users to derive abstractions of graphs by considering subsets of node attributes. Users can answer questions formulated in terms of node attributes, but questions with references to edge labels, such as those introduced in Section 7.1.2, are not supported.

An obvious approach would be to add text labels to the edges shown in these visualizations. In general, optimal edge label placement in graph drawings is intractable [40]. However, in the graph drawing research community, approximation methods are often used to optimize label positioning [85]. Other approaches include substituting line segments that represent edges with text labels that span from source to target nodes [97]. Still, the optimized placement of labels in the limited drawing space does not address the questions mentioned above.

7.3 Associated data

To answer questions related to node attributes and edge labels, the latter also have to become first class citizens of the visual representation. As Figure 7.2(a) shows, this can be achieved by showing a list of edge labels at the center of the visualization. Every unique edge label is represented by a rectangular region. The set of edges is partitioned by letting every edge pass through that region that represents its edge label.

The directed edges found in state transition graphs impose an ordering on the two nodes they connect (see Section 2.1). For an edge $(v, l, v') \in E$, v is called the source and v' the target. The convention of visualizing ordinal data from left to right [81] is adhered to: all source nodes are represented by a region at the left of the visualization while all target nodes are represented by a region at the right (see Figure 7.2(a)). This implies that the collection of nodes is represented twice, once in its capacity as source and once as target.

The effectiveness of the visualization techniques discussed in previous chapters results from being able to consider clusters of nodes that share properties expressed in terms of a few attributes (see Section 4.2, for example). In this chapter, the same approach is taken by enabling the user to select a subset of node attributes.

The entire set of nodes is consequently partitioned based on the different values assumed for the first attribute in this selection (see Figure 7.2(b)). Every resulting cluster now contains a disjoint subset of the original set of nodes. Next, each of these clusters is sub-partitioned based on the second variable (Figure 7.2(c)), resulting in another level in the hierarchy. By recursive partitioning, a new layer of clusters is computed for every attribute selected by the user. Every cluster, apart from the root, has a child-of relationship with one higher level cluster.

A line that connects a leaf cluster on the left with an edge label X in the center im-

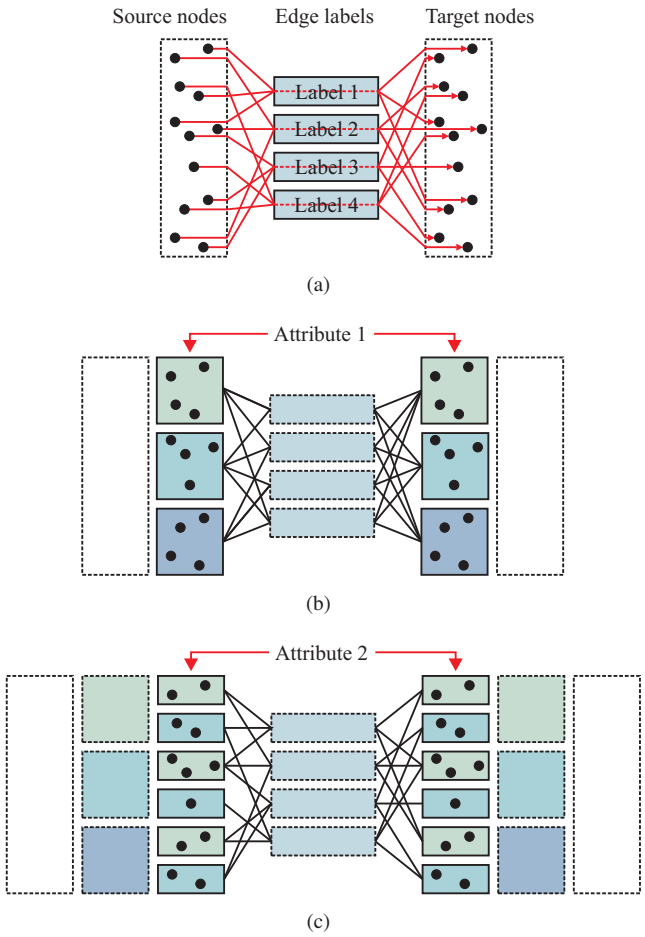


Figure 7.2: Approach.

plies that the cluster contains at least one (source) node with an outgoing edge of type X. Similarly, a line that connects edge label X with a leaf cluster on the right means that it contains at least one (target) node with an incoming edge of type X. In this way, the visualization intuitively reads from left to right. This, combined with real-time interaction and visual feedback, supports user queries in a natural way, as shown in following sections.

In the final visualization (see Figure 7.3), the root cluster is not shown since it simply represents the set of all nodes. In early versions of a prototype that implements this approach the clustering hierarchy was represented with a node link diagram (see, for instance, Chapter 4). Since the edges found in the data are already represented with line segments, however, the node link diagram was replaced with an icicle plot. Subtle cushioning is used to differentiate regions better [88]. The different attribute values assumed in a particular level are shown with labels and encoded with distinct colors to enable users

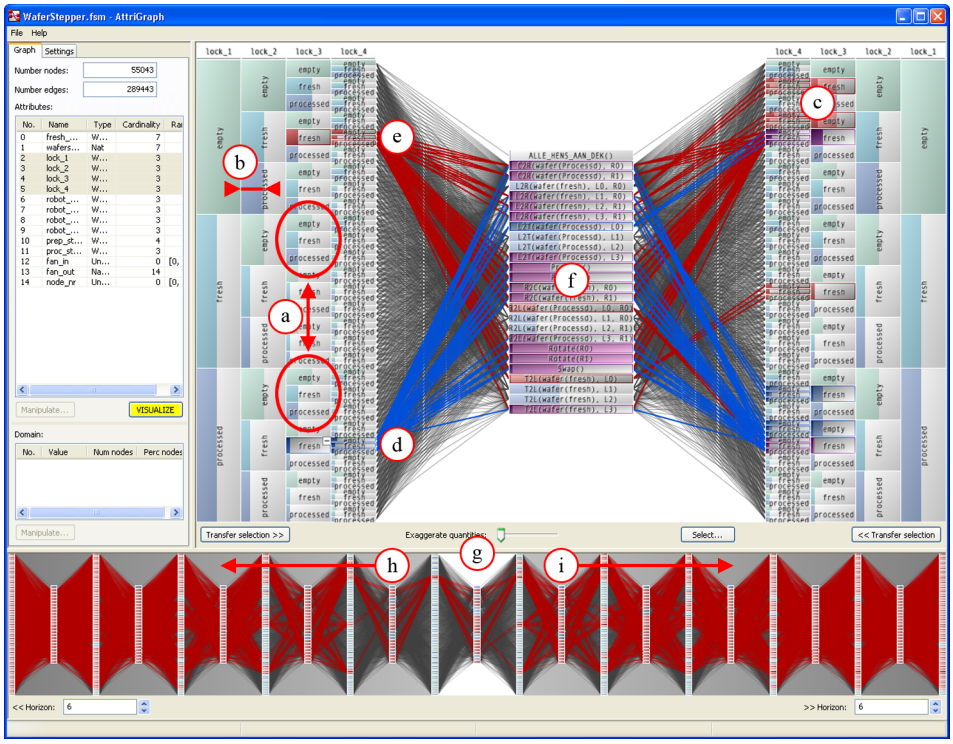


Figure 7.3: Visualizing node attributes and edge labels.

to identify repeated patterns (Figure 7.3(a)).

The number of nodes in every cluster and the number of edges with a particular edge label are encoded with the length of the colored bar inside the region (see Figure 7.3(b)). A logarithmic scale can be used to amplify differences for small quantities. This improves on earlier work, for example that discussed in Chapter 4, by combining hierarchical and quantitative data in a single representation. It also avoids issues encountered when encoding quantitative information with region size.

7.4 Interaction

Interaction plays an important role in the technique presented here. As described below, the user is enabled to interactively inspect and query state transition graphs based on the data associated with their nodes and edges. This is illustrated by showing how the questions introduced in Section 7.1.2 can be answered with this technique in a straightforward fashion. In the discussion that follows, consider how the user is able to rapidly find the answers with no more than three mouse clicks and contrast this with the effort needed to formulate and evaluate the formal queries discussed in Section 7.1.3.

When the user selects a source cluster, the cluster and all nodes contained in it are

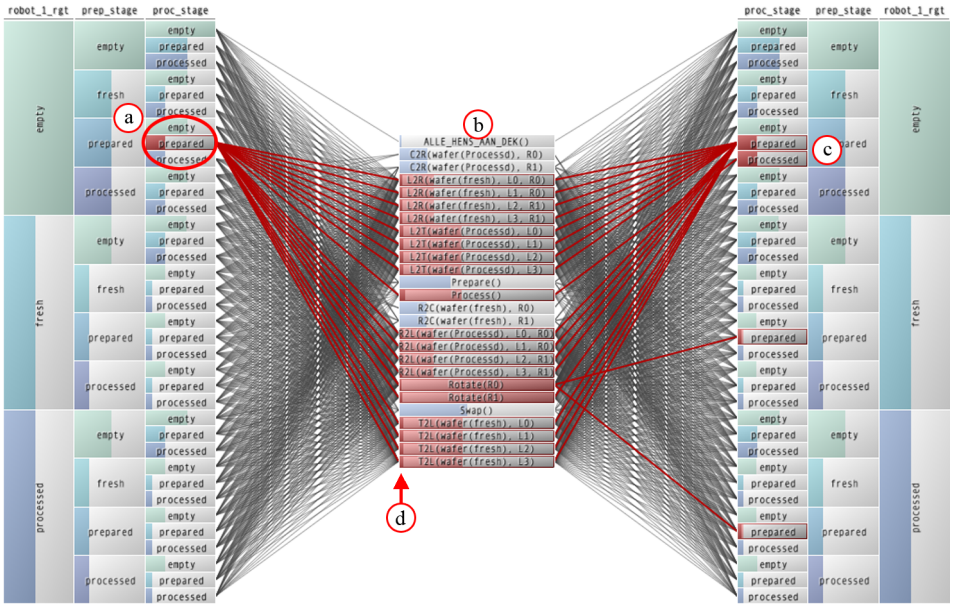


Figure 7.4: Question 1.

selected. The same holds for all outgoing edges and their edge labels, as well as target nodes and their parent clusters. Selected clusters and edge labels are highlighted in red.

Note that for a target cluster selected in this way, it is possible that only some of the contained nodes are selected. The same holds for edge labels (representing collections of edges). Knowing this may be important when interpreting the results. The fraction of selected edges and nodes is shown with a red bar overlaid on a lighter bar that encodes the total (see Figure 7.3(c)). When an edge label or target cluster is selected, the same reasoning is applied. With the above knowledge, it is now possible to answer the first question posed to users.

Question 1. Cluster on *robot_1_rgt*, *prep_stage* and *proc_stage*. Select the source cluster containing all nodes where *robot_1_rgt* = *empty*, *prep_stage* = *prepared* and *proc_stage* = *prepared* (see Figure 7.4(a)). Consequently, all labels of outgoing edges are highlighted in the center of the visualization (Figure 7.4(b)). Also, all target nodes are contained in the highlighted target clusters toward the right (Figure 7.4(c)). For the selected edge labels, note the small portion of individual edges that have been selected (Figure 7.4(d)).

Users can refine their selection by selecting a cluster or edge that is already selected a second time. This results in any part of the current selection that is not reachable from this last selection to be deselected. Put differently, only the intersection of the original selection and the selection that would have resulted from the last mouse click remains selected. Knowing this, the second and third question put to analysts can be answered.

Question 2. Keep the previous clustering and select all clusters where *robot_1_rgt* has the values *fresh* or *processed* and *prep_stage* and *proc_stage* assume the value *prepared* (see Figure 7.5(a)(i)). Now select the edge label *Process* to refine the selection (Figure 7.5(b)(i)). Consequently, all target clusters that can be reached from the initial selection via edges labeled with *Process* are highlighted toward the right (Figure 7.5(b)(ii)).

Note that the source and target clusters differ only in terms of the value assumed for *proc_stage*. In fact, as would be expected from the description of the wafer stepper system (see Section 7.1.2), the action *Process* changes the status of the wafer on the processing stage from *prepared* to *processed*.

Question 3. Keep the previous clustering and select the edge label *Prepare* in the center of the visualization (see Figure 7.6(a)). All clusters containing source nodes that have outgoing edges of this type are highlighted toward the left (Figure 7.6(b)). Also, all clusters containing target nodes that have incoming edges of this type are highlighted toward the right (Figure 7.6(c)).

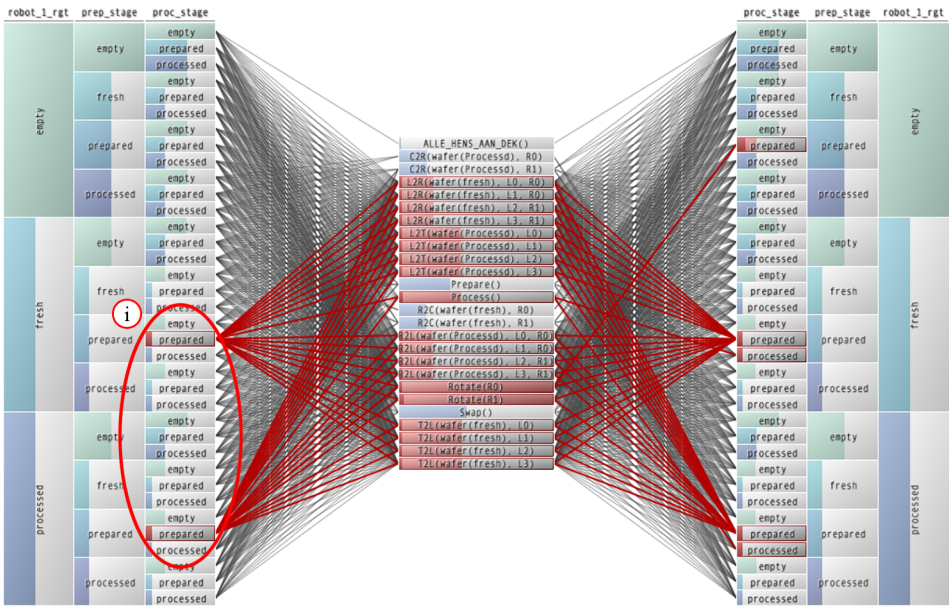
To answer the second part of the question, observe that *prep_stage* always assumes the value *fresh* while *robot_1_rgt* and *proc_stage* can assume any values for all highlighted clusters at the left. Finally, note that all horizontal bars of the highlighted clusters toward the left are completely filled with red. The conclusion is that all states where *prep_stage* contains a fresh wafer have outgoing *Prepare* actions, whatever the contents of *robot_1_rgt* and *proc_stage*. This is an important property of the wafer stepper system which implies that the preparation for processing of fresh wafers can proceed independently of robot 1 and the processing stage.

Users are offered two additional selection modes to add or subtract from the current selection. In add-mode all clusters or edge labels clicked on are added to the selection. For instance, in order to select the two source clusters to answer question 2 above, it is necessary to temporarily switch to add-mode. Similarly, in subtract-mode anything in the current selection is removed when it is clicked.

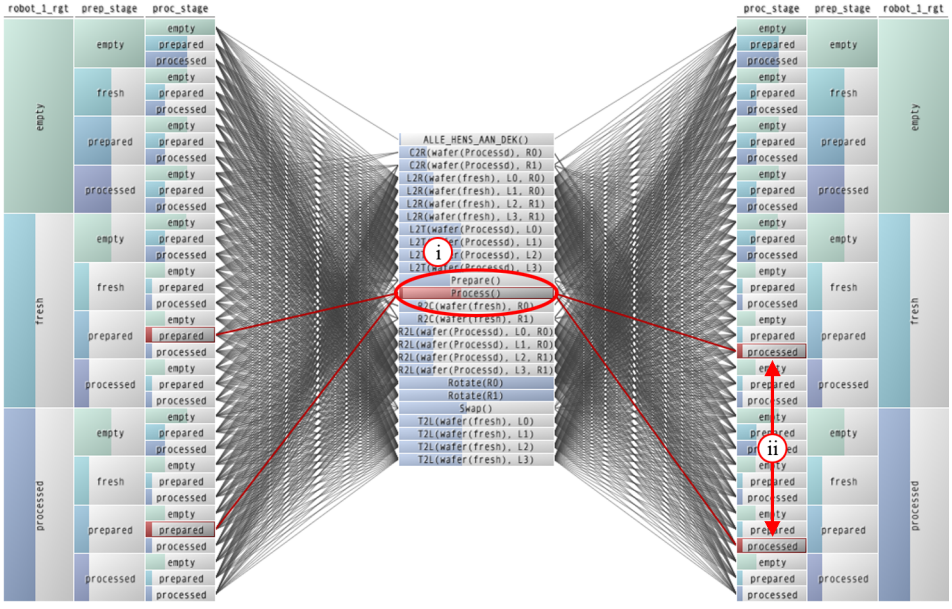
To guide the user, previews of those clusters or edge labels reachable from the current cursor position are provided. This is highlighted in blue (see Figure 7.3(d)). As outlined above, the current selection is shown in red (Figure 7.3(e)). The intersection of the preview and current selection is shown in purple (Figure 7.3(f)). This enables the user to answer questions such as, "which outgoing edge types are possible from my current position that are also possible from my previous selection?"

As shown in the next section, the visualization technique described above has been received enthusiastically. When an initial prototype was shown to users, they were pleased with the ease with which they could answer many questions about their data. These include questions such as, "which edge types are activated by specific node attributes?" and, "how and from where are specific edge types possible?" However, once these were answered they started asking questions like, "from which clusters four steps back is my current selection reachable?" and, "where can I go to from here in six steps?"

To answer such questions the original prototype was extended with a context view. The current view, discussed above, is shown with a white background (see Figure 7.3(g)).



(a)



(b)

Figure 7.5: Question 2.

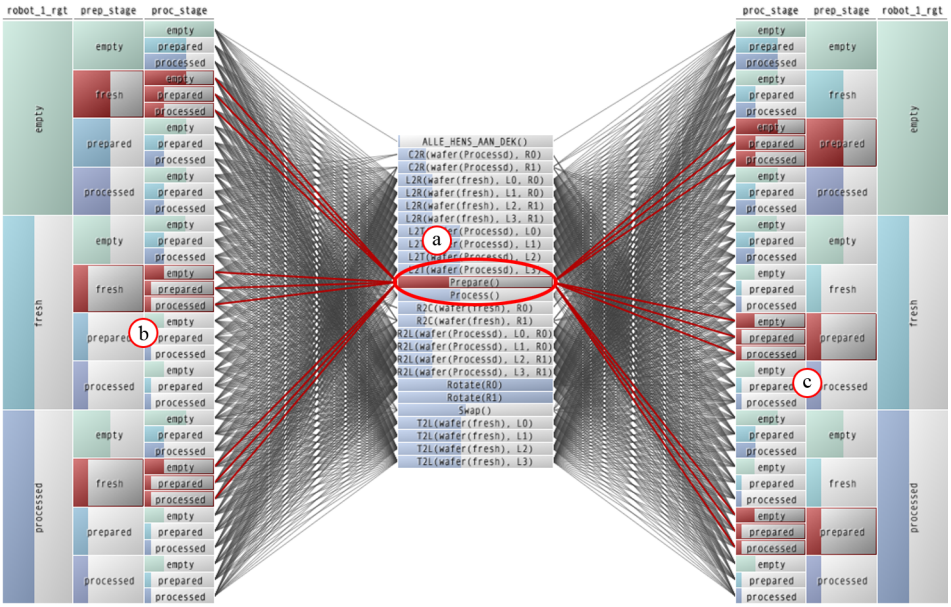


Figure 7.6: Question 3.

Users can interactively specify how many steps to show before and after the current view (Figure 7.3(h) and (i)). For every step, the edge labels and leaf clusters are repeated.

The current and context views are linked: when users make selections in the main view, the context view shows how this selection can be reached from n steps back and which clusters can be reached m steps forward. The context view can also be used to refine a selection. To support walking through the graph, users can transfer the selection on the left or right of the main view to the opposite side, after which selections in both views are updated.

To follow up on the interviews with domain experts, they were asked to use the techniques presented here to answer the questions that had been posed to them (see Section 7.1.2). Following steps similar to those outlined above, they were all able to find answers to these questions.

Next, the analysts were invited to inspect their own data with the prototype. Two sessions, spread over two weeks, were organized with every analyst. Semi-structured interviews were conducted, where the prototype that had been developed was first introduced and its main features highlighted. Analysts were then asked to talk aloud as they interacted with their data. After the sessions they all indicated that they had learned something new about their data. Also, they were all interested in using the prototype again. Below, three of these observation sessions are reported on.

7.5 Case - traffic regulator

In Section 7.1 it was mentioned that one analyst was studying a transition graph that models a traffic regulation system. This graph contains 40,826 nodes described by 41 attributes and 221,618 edges with 38 labels. Using the technique described above, he was able to verify that the dangerous scenarios mentioned before cannot occur.

Another task he set himself was to identify which edge labels have an influence on which node attributes. It was assumed that every edge type had an influence on a single traffic light. However, by first clustering on attributes that describe different lights' phases and then sweeping the cursor over the edge labels in the visualization, he quickly discovered that there are some edge types that result in more than one light changing its phase. By scrutinizing the formal specification of the system the analyst confirmed that such "conflicts" resulted from the way the behavior is modeled. The analyst was pleased with the ease with which he could identify these properties.

7.6 Case - communication protocol

Another analyst was studying a communication protocol for message passing over an unreliable channel, developed for consumer electronics. The corresponding transition graph has 10,548 nodes, 12,168 edges, 40 node attributes and 119 action labels. The analyst had not created the model himself. In fact, he was trying to familiarize himself with the protocol by studying the behavior exhibited by this model.

The analyst had read all available documentation but still felt he did not fully grasp the very complex behavior of this system. First, he wanted to confirm that specific edge types lead out from nodes with specific attribute values. By clustering on these attributes and selecting the configurations he was interested in, it was straightforward to check that certain edge labels did not light up. He also wanted to check which values are assumed for specific node attributes in target nodes that can be reached via specific types of edges. Again, by clustering and selection of edge labels this was simple.

The analyst proceeded to formulate and prove or disprove many hypotheses quickly and efficiently. He found that this gave him a much better intuition about the protocol. In his own words, "seeing that certain combinations of attribute values, which I thought were impossible, do occur, really got me thinking."

7.7 Case - understanding games

Games are often studied as examples of how a set of simple rules give rise to very complex systems (see, for example, Section 1.1). One of the interviewed analysts had specified the behavior of a simple board game and generated the corresponding state transition graph. This graph consists of 76,002 nodes with 8 attributes and 95,335 edges with 168 labels. Using his current tools, it was difficult to learn behavioral properties about states (nodes) with specific attribute configurations.

In the game there are two opponents, which will be called *A* and *B*. The analyst had expected that the two opponents could only win from specific positions on the board. A

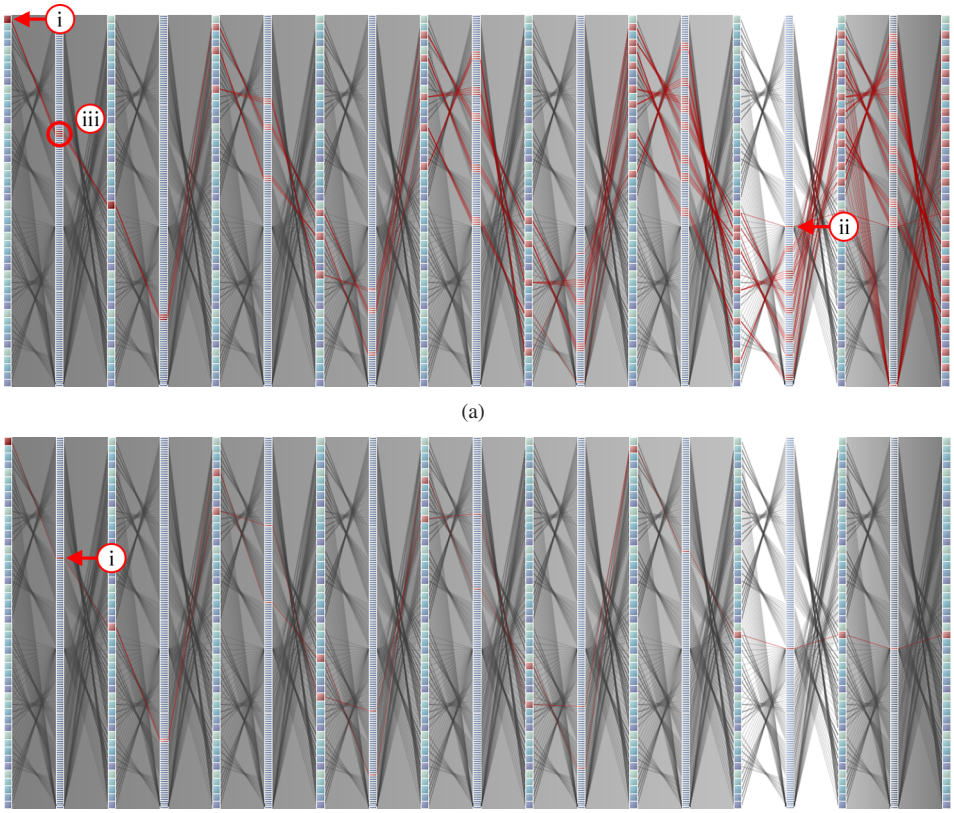


Figure 7.7: Game strategies.

win is expressed by an edge labeled *Win*. By clustering on the attributes that encode the positions of *A* and *B*, and by selecting the edge label Win, he immediately saw that he was wrong. Both opponents could win in any position on the board. He explained that he had just found that, contrary to what he had expected, there exists "no invariant of the type" of the opponent that can win in any position.

The analyst was also keen to use the context view to traverse the graph. That is, he selected a specific cluster of nodes and then looked forward to identify where the first edge labeled with *Win* occurs. He then analyzed the intermediate steps to identify game strategies.

For example, in Figure 7.7(a)(i) the cluster containing the initial node has been selected. Also, Figure 7.7(a)(ii) shows the first occurrence of a winning move. From Figure 7.7(a) the analyst could see that, starting from the initial node, a minimum of eight edges have to be traversed before an edge labeled *Win* is encountered (Figure 7.7(a)(i) and Figure 7.7(a)(ii) are eight steps apart).

The next question the analyst asked himself was which of the three actions possible

from the initial state would lead to this win (Figure 7.7(a)(iii)). By filtering the results on the wining action (Figure 7.7(a)(ii)), only one of the initial three actions remained selected, as shown in Figure 7.7(b)(i). In this way, the analyst was able to confirm that the first move does have a significant impact on chances of winning the game.

7.8 Discussion

Apart from being non-trivial, the graphs discussed in the current chapter are also large, all containing tens of thousands of nodes and tens to hundreds of thousands of edges. From this it is deduced that the technique presented here scales well. Similar to the methods presented in previous chapters, this is attributed to the ability to take different perspectives on the data. Different subsets of attributes generate different abstract, and much simplified, visual representations of the original graph. These can be interpreted in terms of knowledge the user already has: the meaning of the data associated with nodes and edges.

User observations show that the ability to rapidly query these representations enables users to effectively and efficiently analyze state transition graphs. The analysts who were interviewed were also enthusiastic about using the technique to explain aspects of their data to non-experts. Although it was not the intent to design a communication aid, this is an encouraging result.

Clustering is only possible when attributes and edge labels have discrete domains. These domains usually have a relatively small cardinality (typically, $2 \leq |D_i| \leq 30$ and $2 \leq |L| \leq 120$). For many real-world data sets, and transition graphs in particular, this assumption is valid. However, cases may arise where domains are larger or continuous. To handle these, discretization algorithms such as those presented in Chapter 6 can be used to partition domains into a number of discrete categories.

Related to this, the approach of listing all edge labels down the center of the visualization, as described in this chapter, worked well for most data sets. However, the 168 edge labels discussed in Section 7.7 nearly proved too many. To address this limitation, a possible solution is to enable users to group edge labels. Similar to the clustering approach taken with node attributes, they would then be able to focus on only those labels relevant to their current inquiry. This will further reduce the complexity of the data being investigated.

Analysts were able to use the approach outlined in this chapter to answer predefined questions before the end of the first observation session. Moreover, they arrived at the second session with questions devised of their own accord and then spontaneously started formulating and answering queries as they interacted with their data. As suggested by the results in Sections 7.1.1–7.1.3 these questions were all phrased in terms of node attributes and edge labels.

Apart from answering specific questions, users also used the technique for explorative analysis by, for example, sweeping over the edge labels to identify interesting patterns in terms of highlighted source and target clusters. Users found the approach easy to use, although understanding the bars that encode size information initially required some training.

As users got more familiar with the approach they all developed an interest for studying paths that lead to or from specific node clusters. Using the context view (see Figure 7.3), they often selected a cluster to "look into the future" to identify clusters that can be reached and sequences of edge labels leading to these (similar to the case illustrated in Figure 7.7).

This offers opportunities for future improvements. For example, using the context view, it is currently not possible to analyze individual paths leaving a particular cluster without selecting only one at a time. This is because when the cluster is selected, all paths leading from it are highlighted in red, making it impossible to disentangle them. A solution for comparing different paths, although not trivial, would be a very valuable addition to the technique discussed in this chapter. Related to this, the ability to accurately distinguishing paths that intersect at various steps would greatly benefit users.

Finally, a few observations with regard to the methodological approach taken in the current chapter are in order. To understand the role that node and edge labels play, semi-structured interviews were conducted with domain experts. Next, the insights gained from this study served as input to design a solution where node and edge attributes play a central role. This was found to be a much more constructive approach than trying to identify concrete user requirements in advance. As noted in Chapter 1, due to a number of reasons, the identification of such requirements has proved to be particularly tricky. These challenges are discussed in greater detail in Chapter 8.

While interviewing users, special attention was paid to understanding their current way of working. Furthermore, to validate the approach presented here, it was contrasted with current practice. This also served as an especially useful way to persuade users of the merit of the visual approach toward analyzing state transition graphs presented in this chapter.

Chapter 8

Reflections

In this chapter a retrospective view is taken on the techniques for interactive visualization of state transition graphs introduced in this dissertation. For this, the close collaboration with system analysts to design and build several prototypes is reconsidered. This process is discussed and emphasis is placed on how an understanding of the application domain evolved as a result. It is shown how this insight helped to overcome challenges and to design better visualizations. General principles derived from the experience of iterative design are also considered. Based on these, a model for information visualization design is presented.

8.1 Visualization for system analysis

In a recent report, which identifies a number of visualization research challenges, Johnson et al. [38] write, "In order to benefit both current and new application domains, we must engage in the systematic exploration of the design space of visualization techniques." During the past two decades the information visualization research community has developed and published many presentation techniques. Traditionally, issues such as the novelty of a method or measuring insight have been emphasized. However, there are few results where a retrospective analysis is made of the exploration of the design space.

Over a period of three-and-a-half years the author collaborated extensively with system analysts. As stated in Chapter 1, the goal was to design and implement information visualization techniques to help analysts better understand state transition graphs. This was not trivial. Transition graphs model the behavior of complex computer-based systems [4] and although their organizational structure and meaning are clear, gaining a thorough understanding is difficult (see Section 2.1 and Section 2.2).

By cooperating with analysts, the author had the opportunity to work with domain experts to design visualizations within a specific application context. Because this was not a one-off episode, it was possible to revisit the problem. This allowed for refining an understanding of the problem space and for repeatedly exploring the solution space.

From a visualization research perspective, developing visualizations for specific prob-

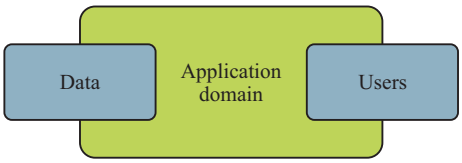


Figure 8.1: Visualization design problem space.

lems may initially appear narrow in scope. However, as suggested by the above mentioned NIH-NSF report [38], such cases do offer design lessons which can be reapplied.

In following sections, the experience of iteratively designing interactive visualizations for state transition graphs is reported on and a number of insights related to this process are discussed. The aim is twofold. First, to contribute to the knowledge embodied in context specific experience reports. This is relevant in light of a recent emphasis on close collaboration with domain experts to solve real-world problems [38, 93]. The work presented in this dissertation also serves as a good example of the challenges encountered and the design choices that have to be made when designing visualizations for concrete application domains. By taking a wider view, a second goal is to shed light on typical challenges that information visualization designers are confronted with and how these can be dealt with.

8.2 Problem space

General graph drawing tools have historically been used to visualize state transition graphs, as discussed in Section 2.3. Also, some custom visualization techniques for transition graph visualization have been developed. In particular, results by Van Ham et al. [25] led to a promising new technique that was welcomed by target users.

The aim of the research presented in this dissertation was to further investigate interactive methods for the visualization of state transition graphs. To achieve this, it was important to understand the problem being addressed. Without adequate knowledge of the problem space it would not have been possible to come up with appropriate visualizations. Developing an understanding is a gradual process, though, and the model illustrated in Figure 8.1 summarizes the three aspects of the problem space which were repeatedly revisited: the data, the users and the application domain. As shown below, all three components influenced the decisions made while the solution space was explored.

8.2.1 Data

The aim of information visualization is to assist people to explore and explain data [12]. Indeed, there is a reference to data in the research question addressed in this dissertation (see Section 1.4). As noted in Chapters 1 and 2, state transition graphs are relatively easy to understand at a conceptual level. Nodes represent system states and directed edges represent transitions between states [4].

Despite being straightforward in terms of what they represent, however, state transition graphs are not easy to study (see Section 2.1 and Section 2.2). First, they often contain tens of thousands of nodes, or more, and tens to hundreds of thousands of edges (the state explosion problem). This hinders analysis and insight. Second, transition graphs describe system behavior at a low abstraction level, making it difficult for analysts to map their domain knowledge to nodes and edges in the data.

8.2.2 Users

Users want to understand their data. Typical for the field of information visualization, the approach taken in this dissertation was user-centered [47]. To enable users to get a better intuition about the systems being studied and to allow them to investigate particular features and answer specific questions about their data (see Section 2.2), it was necessary to define and understand the typical user.

Typical users were identified as computer scientists and engineers who use transition graphs to model and study the behavior of complex computer-based systems. At Eindhoven University of Technology there is a strong tradition of system analysis and the author was fortunate to have close collaboration with a number of researchers at the departments of Computer Science and Mechanical Engineering. Although the size of the user group was not large, varying between seven and ten, this meant that there was continuous access to users and that the author was already embedded in their work environment.

8.2.3 Application domain

Data and users do not exist in isolation; they are part of the application domain. Many analysis techniques for computer-based systems involve the translation of behavioral specifications into state transition graphs (see Chapter 2). In general, transition graphs are analyzed to identify and correct behavioral problems of an existing system or to design the behavior of new systems before they are implemented.

To do so, as outlined in Section 2.2, analysts have devised a number of approaches for addressing the complexity of state transition graphs. These include: deriving and analyzing a much smaller variant of the graph, formulating and checking requirements with the aid of mathematical formalisms, and analyzing the data by using simulation tools.

8.3 Solution space

When users were first approached and informed of the goals of the investigation described in this dissertation they readily accepted, and in some cases enthusiastically proclaimed that visualization could play an important role during their analysis of complex systems. However, it proved very difficult for them to formulate what this role should be. This implied that right away an assumption had to be dropped. It had been expected that analysts would have a number of questions or a set of routine tasks to address with visualization. This was not the case.

To deal with unclear requirements, the following course of action was taken. First, it was necessary to infiltrate the users' world to closely collaborate with them. Second, an experimental and exploratory approach was adopted. Prototypes were developed, refined and extended in several rounds, finishing each with an evaluation.

In Chapters 3–7, five techniques for the interactive visualization of state transition graphs, which were developed in this fashion, were presented. Below, these techniques are reconsidered in chronological order to illustrate how solutions evolved from previous work by addressing shortcomings, by focusing on successes and by investigating design alternatives. To facilitate comparison, every technique is considered in terms of four aspects: goal, approach, execution and results.

8.3.1 Selection and projection - Chapter 3

Goal. Project the multivariate nodes of state transition graphs to the 2D plane by considering a subset of user-selected node attributes.

Approach. Previous techniques for the visualization of state transition graphs had focused on graph topology [21, 37, 44, 25] (see also Section 2.3). However, the nodes found in state transition graphs are multivariate and every node has associated data (see Section 2.1). These are typically the values of the variables introduced in the original formal specification (see Section 2.1). Because this has been largely ignored by the aforementioned visualization techniques, a visualization method was developed where node attributes play a more significant role.

Execution. For a first experiment, a number of simple questions, which were thought to be important for a better understanding of transition graphs, were defined in terms of node attributes (see Chapter 3). For example, "are there subsets of node attributes that exhibit interesting behavior?" To develop a visualization technique that would enable users to answer these questions, existing multivariate data visualization techniques were considered (see Section 2.5.1).

A common approach for visualizing multivariate data is to show low-dimensional projections of high dimensional data [69]. To reduce the complexity of the data set, the user is often able to select a subset of dimensions to visualize the data only with respect to these [98]. In Chapter 3 a method that was developed based on these principles was discussed. The user is presented with two correlated visualizations. The first is an overview of the different attributes shown as a series of parallel histograms, or bar charts. Every row represents an attribute and a column represents a node. Interesting attributes can be identified by visually discovering patterns, correlations, gaps and outliers. This representation is also used to select a subset of attributes that the user finds interesting.

The transition graph is then visualized as a node-link diagram, in a second view, by projecting the nodes to 2D based on the selection. A number of different projection techniques were investigated. These include techniques based on user-defined projection vectors, high-dimensional nested grids and statistical methods. To assess the combination of the above techniques, the ability of the approach to assist users in answering the questions that had been identified was considered.

Results. State transition graphs are complex structures in n-space. Since system behavior is reflected by the values that node attributes assume, approaching transition graphs from a multivariate perspective can lead to interesting new insights. Results from a prototype, in which the above techniques were implemented, were promising and showed clear relationships between values assumed by node attributes and the behavior expressed by directed edges. For the first time, users were able to quickly identify such relationships visually without the need for rigorous formal analysis.

Nonetheless, because the meaning of the 2D locations being projected to were not clear, the resulting visualizations were not always easy to interpret, despite the presence of striking patterns. Also, analysts did not use the visual overview of dimensions to select a subset of attributes as frequently as had been anticipated. Instead, they relied on their knowledge of the semantics associated with the attributes.

By interviewing users, a fundamental fact about the relationship between them and their data was confirmed: analysts associate precise meaning with node attributes. They know what aspect of the modeled system an attribute describes and what it means when it assumes different values. This suggested that this knowledge could be used as a stepping stone for interpreting visualizations. This was an important lesson. Analysts understand the organizational structure of their data. What they find challenging is using this knowledge to better understand the behavior that their data sets model.

8.3.2 Attribute-based clustering - Chapter 4

Goal. Develop a visualization technique where the positions of collections of nodes can be interpreted by taking into account the meaning of associated node attributes.

Approach. To address the shortcomings discussed in the previous section, a new visualization technique was developed. The guiding principle, which would prove important for all subsequent work, was that location should have meaning. That is, it must be possible for the position of a cluster of nodes to be interpretable in terms of the values assumed by a subset of node attributes. This is also increasingly being stressed by other researchers developing techniques to visualize general multivariate graphs [75, 91, 3] (see also Section 2.5.2).

Execution. Because users associate precise meaning with node attributes, a method that uses an interactive attribute-based clustering facility was developed. This technique was presented in Chapter 4.

Like the previous experiment (see Section 8.3.1), a subset of node attributes can be selected by users, this time by relying on their domain knowledge. Consequently a much simplified representation of the state transition graph is generated. In this abstraction, nodes that have equal values for the subset of user-defined attributes are grouped together in clusters. Further, a hierarchy that represents the partitioning of the original set of nodes into successively finer partitions is maintained. All edges that span between a source and a target cluster are bundled together.

The abstract graph and additional data that is generated in this fashion are shown in a single integrated visualization (see Sections 4.2–4.4). The resulting clusters and the

clustering hierarchy are visualized such that analysts can relate the position of a cluster to the values assumed by the selected node attributes. Bundled edges are shown with an arc diagram, which highlights the presence or absence of behavioral patterns. Finally, the number of nodes in every cluster is shown with a series of hierarchically nested bar charts.

Results. Clustering enabled users to reduce the complexity of transition graphs in a semantically rich way. The ability to relate meaning to the positions of visual elements, by means of the clustering hierarchy and meaning associated with node attributes, resulted in an improved visualization compared to the previous experiment.

By showing hierarchical, relational, and metric data in an integrated fashion, users could identify and analyze correlations between them. Furthermore, by grouping subsets of nodes together and representing such a cluster with a single visual entity, the performance of the visualization was dramatically improved compared to rendering every individual node at the position it had been projected to.

Because users understand what attributes mean, the reduction technique itself served as a powerful tool. First, by clustering on different subsets of attributes, it was possible to take different perspectives on and explore different abstractions of the transition graph. Second, it was possible to perform more focused analysis by answering specific questions related to node attributes.

The above claims were validated by collaborating with users to develop a case study of a large real-world data set. Not only were interesting new facts about their data discovered, it was also possible to explain why unwanted scenarios occurred. This collaboration also suggested that a better understanding of the system analysis application domain would lead to further opportunities for adding value with visualization techniques.

8.3.3 User-defined diagrams - Chapter 5

Goal. Bridge the gap between the way users reason about systems, as schematic diagrams, and state transition graphs, which are abstract representations of system behavior.

Approach. In terms of the model of the problem space introduced in Section 8.2 (see Figure 8.1), the techniques discussed so far were the result of carefully considering the users and their data. Next, the focus shifted to the application domain.

To experience first-hand what users do, the author became an apprentice system analyst [35] and assisted an experienced analyst in studying the behavior of an automated parking system [48] (see also Section 5.1.2). First, a number of requirements were identified. Second, the behavior of different components of the parking system was rigorously defined in process algebra [18]. The combined system behavior was generated by automated tools interleaving these specifications. This was studied in an attempt to validate that all requirements were met.

Simulation was regularly used to quickly and incrementally check behavior resulting from the specification [72]. While moving back and forth between specification and simulation, an opportunity for improving this way of working was identified. Interpreting the text-based output of the simulator was arduous, not intuitive, and prone to human error. To resolve this, a simple visualization plug-in, which maps a single state of the parking

garage to a 2D floor plan, was developed for the simulator [48]. Despite its simplicity and limited features, the visualization was extremely effective and played a significant role in finding a number of serious problems.

The success of the above solution was attributed to the fact that the visual representation was a close match to how analysts reasoned about the parking system. From this, it was concluded that presenting a problem in terms of the user's own conceptualization of it is very effective. This was confirmed when it was discovered that analysts regularly drew rough diagrams of the systems they studied as they were going about their work (see Section 5.1.1).

Execution. Although the visual approach taken with the parking garage was very effective, it is a custom solution for a specific system. To avoid having to program new visualizations for different systems, an opportunity was identified for providing users with a visualization tool that is reconfigurable.

In Chapter 5 a new technique that meets this requirement was described. This technique enables analysts to define custom diagrams that fit their conceptualizations of systems and to use these to gain further insight. By providing a simple graphical editor, this can be done rapidly and intuitively. Next, graphical properties of the diagram can be linked to attributes found in their data. In this way the data values assumed by the attributes in a particular node determine the graphical properties of the diagram that represents that system state.

To enable users to identify groups of nodes that interest them, user-defined diagrams were integrated with attribute-base clustering. The principle of simulation, often encountered in system analysts' application domain, was also reused by showing diagrammatic representations of a current system state and all its direct neighbors.

Results. User-defined diagrams enable users to bridge the gap between their existing knowledge of a data set and interactive visualizations of it. This is achieved by allowing users to map their conceptualization of a problem to a diagrammatic representation of it.

Analysts were enthusiastic about analyzing systems with custom diagrams. Two case studies involving real-world systems were conducted with the approach. In both cases important requirements were verified and previously unknown discoveries were made. The case studies showed that if users are able to capture their current understanding of a problem, this can be effectively used to learn even more. Another unexpected discovery was made; although the technique had been designed for visually analyzing data, users often used it as a communication aid to talk about systems amongst each other and with non-experts.

The reuse of the simulation metaphor was particularly popular with users. It enabled them to take a local view on the complex network of possible system behavior represented by state transition graphs. In this view, the possible transitions between states are represented in a left to right fashion: all states from which the current state can be reached are shown toward its left and all states reachable from it are shown toward its right (see Section 5.4.2). Users indicated that they found this representation intuitive and easy to interpret. Finally, there were many requests for visualizing system traces, or sequences of consecutive system states, with the aid of custom diagrams.

8.3.4 Multiple views on traces - Chapter 6

Goal. Develop a visualization technique for system traces that takes into account the lessons learned from visualizing state transition graphs.

Approach. For systems that have a very large number, or even infinitely many states, analysts have to find alternatives to studying state transition graphs. A common approach is to generate traces by simulation [72] (see also Section 5.1.2). A mathematical model of a system's behavior is first constructed. Using this as input, software tools generate a finite number of consecutive system states. These states express a system's behavior over a finite interval of time. Because system traces are closely related to transition graphs and often encountered in practice, a technique was developed to visualize them.

Execution. The states in a trace are similar to the nodes representing system states in transition graphs. They are also described as a tuple of attribute values (see Section 6.2). This similarity, and various user requests, led to investigating whether the visualization methods discussed in Chapters 3–5 could be reused for this different, but related data. To this end, some of the techniques were combined with time series visualization to offer three perspectives on system traces: a schematic diagram, time series plots, and a graph view. This method was discussed in Chapter 6.

Diagrams are defined as discussed in the previous section and their graphical properties can be linked with data attributes. The data values assumed by the attributes determine the graphical properties of the diagram and in this way the diagram that represents a particular state is obtained. Diagrams can be used to see how the system changes by animating over time. Furthermore, they are useful for studying individual states.

The second view, time series plots, maps the values assumed by attributes against time. This enables the user to study correlations between attributes and trends over time. The time series view shows context and detail, supports zooming, and offers other interactive features.

For the final view, the graph view, users are enabled to interactively define an equivalence relation on the states. This is similar to attribute-based clustering and the linear sequence of states is grouped into different clusters accordingly. These are considered as nodes of a graph that generalizes the behavior found in the original trace. In it, transitions between the original states now form directed edges. This graph is visualized using the approach outlined in Section 8.3.2. Time series plots and the graph view can be annotated with user-defined diagrams and all views are correlated.

Results. By providing different views on system traces, users were able to take different perspectives on their data. In this way they could switch between different representations of a problem, assisting them in making cognitive switches and leading to deeper insight. This was illustrated by developing a case for a data set that describes a real-world industrial system.

The attributes associated with states in system traces often have continuous domains. The graphical properties of diagrams that represent different states are calculated by interpolating over a range. As a result, user-defined diagrams could be effortlessly applied to

visualize trace data. This proved that diagram-based visualizations are portable and could potentially be integrated with many existing visualization techniques.

Attribute-based clustering presupposes the existence of discrete domains, however. In order to cluster system traces, a number of classification algorithms were therefor provided to discretize the attribute domains. This also illustrates that the cluster-based visualization techniques that had been developed up to this point could be extended to visualize multivariate data with continuous attribute domains.

8.3.5 Querying nodes and edges - Chapter 7

Goal. Make edge labels first class citizens of a visual representation that enables users to inspect state transition graphs in terms of data associated with both nodes and edges.

Approach. Node attributes play a central role in all the techniques discussed above. However, state transition graphs also have labels associated with their edges. These distinguish different actions that result in transitions between system states. The techniques that had been developed up to this point do not support users in considering such associated data during their analysis. Feedback from user interviews supported the hypothesis that the ability to also consider edge labels during visual inspection would be advantageous.

Execution. Based on the above reasoning, as discussed in Chapter 7, a new method where edge labels became first class citizens was developed. A list of edge labels is displayed at the center of the visualization. The set of edges is partitioned by letting every one pass through the region that represents its label. A directed edge imposes an ordering on the two nodes that it connects, to form a source-action-target triple. With this in mind, the convention of representing direction from left to right was reapplied, as had been successfully done with the simulation view discussed in Section 8.3.2.

By enabling users to perform attribute-based clustering (see Section 8.3.2), a clustering hierarchy is shown at the left and at the right of the visualization. Now, a source-action-target triple is visualized as two line segments. The first connects the cluster containing the source node (right) with the action label and the second connects the action label with the cluster containing the target node (left).

Interaction plays a crucial role in this technique. Users can select a source cluster to see which edge types are possible from it and to which target clusters they lead. When the user selects an edge label or target cluster, the same reasoning is applied. Users can refine their selection by adding to it, subtracting from it, or by filtering. This enables them to rapidly and iteratively form queries with only a few mouse clicks. To further support this, a context view is provided in addition to the main view. The context view enables users to consider longer paths in the transition graph.

Results. Users were pleased with the ease with which they could query their data by considering data associated with both nodes and edges. To inform the design and to validate the method, semi-structured interviews were conducted. These showed that, without the interactive visualization that had been developed, users would have resorted to formal queries and model checking [18] to answer questions related to edge labels.

The effort required to use these methods was contrasted with the simplicity of using the technique that had been developed and users were confronted with this. They all agreed that the interactive visual approach is a welcome alternative to their current way of working. Finally, users were observed as they applied the prototype that had been developed to analyze their own data sets. All users were able to validate important requirements or discovered previously unknown aspects about their data.

With the assistance of the context view, users were able to investigate paths in the transition graph. For example, by selecting a particular configuration of attribute values and identifying paths leading to or leaving from it they were able to deduce interesting facts about their data. In this way, the importance of analyzing paths, first encountered in the visualization of system traces (see Section 8.3.4), had an influence on the design of this technique.

Apart from the visualization technique itself, the design methodology that was used worked well. The informal semi-structured interviews with domain experts revealed a lot about how they approach their work and this informed the design. It also served as an effective way of validating the approach by contrasting the ease of using a visual prototype with the far more arduous formal specification of queries.

8.4 Detours

Section 8.3 may suggest that it was possible to move effortlessly and intuitively from one experimental prototype to the next. This is not true; while previous results did inspire and inform new methods, a number of detours were taken along the way. Some approaches, which initially seemed full of potential, turned out to be unsuccessful.

Although these did not result in useful visualization techniques, they played an important role in exploring the solution space. Also, to gain a better understanding of the problem space, these prototypes were just as important as those discussed in Section 8.3. To illustrate this, two failed attempts are briefly discussed below. To compare these approaches with those of the previous section they are also considered in terms of goal, approach, execution and results.

8.4.1 Shortest path from the initial state

Goal. Design a visualization technique for state transition graphs that enables users to analyze nodes in terms of their distance from the initial node.

Approach. All transition graphs have a special node that represents the initial system state in which the modeled system starts [4]. This can be considered as a point of reference for all other nodes in the graph. After work on selection and projection (see Section 8.3.1), a system that clusters nodes based on their shortest path distance from the initial node was developed.

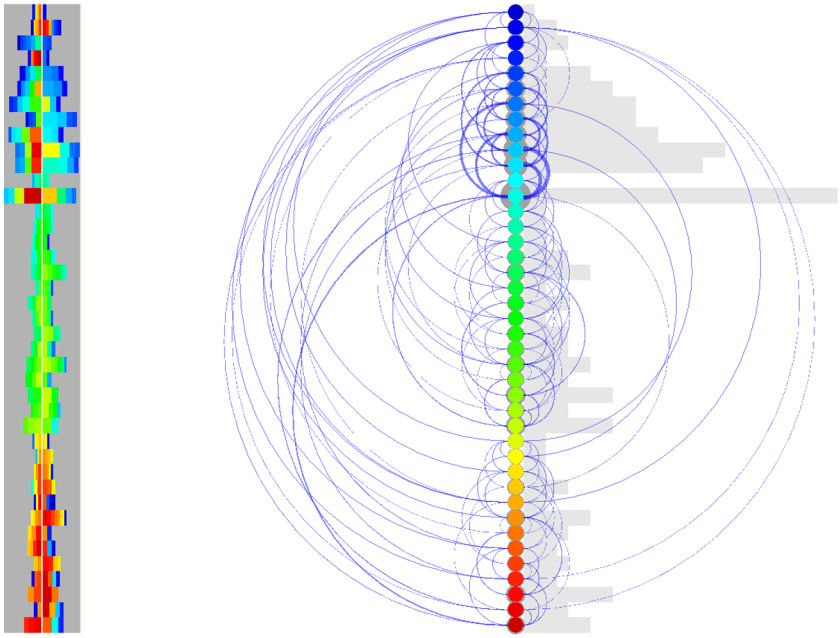


Figure 8.2: Visualization based on shortest path from the initial state.

Execution. By considering the smallest number of edges that can be traversed from every node back to the initial node, nodes that have equal path lengths are grouped together. These clusters are then visualized in a linear fashion, as shown toward the right of Figure 8.2. The topmost cluster contains the initial state. The i^{th} cluster from the top contains all nodes that can be reached from this state by traversing a minimum of i edges. The number of nodes in every cluster is shown as a horizontal bar toward its right.

Similar to attribute-based clustering, edges are bundled and visualized with an arc diagram. Every cluster is color coded. Toward the far left, strips containing the colors assigned to every cluster's direct neighbors (in terms of bundled edges) are shown.

Results. The assumption that the shortest path from the initial state would tell analysts interesting facts about their data was wrong. Despite this, the visual representation did not necessarily fail. In fact, it was easy for users to interpret the meaning of a particular cluster. However, they were not interested in studying their data in terms of path lengths.

This was an important lesson and suggested that position could be related to other more important aspects of the data. Furthermore, as shown in Figure 8.2, arc diagrams worked well to show the presence or absence of patterns in terms of bundled edges. Both these insights, together with the results on selection and projection (see Section 8.3.1), had a direct influence on the work on attribute-based clustering discussed in Section 8.3.2.

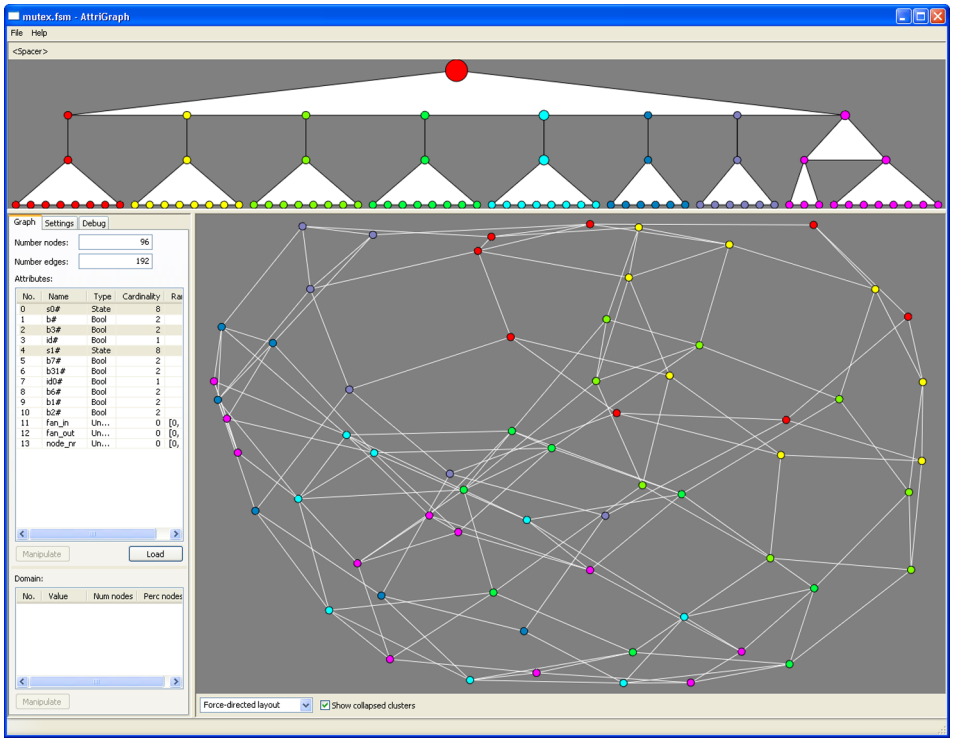


Figure 8.3: Visualization where edges play a more important role.

8.4.2 The role of edges

Goal. Enable users to consider the relationships between edges and node attributes in state transition graphs.

Approach. Before starting work on supporting queries based on node attributes and edge labels (see Section 8.3.5), it was suspected that edges could play a more important role in state transition graph visualization. An initial idea was to place more emphasis on the interconnecting bundles of edges between clusters that result from attribute-based clustering.

Execution. The prototype that was consequently developed positions such clusters with a force-directed layout (see bottom of Figure 8.3). The aim was to also enable analysts to relate these clusters to the values assumed by the subset of node-attributes on which had been clustered. For this purpose the clustering hierarchy is also shown (top of Figure 8.3). The two views are correlated and the selection of a cluster in the one results in highlighting it in the other.

Results. Clusters that are positioned close to each other in the force-directed layout may

be positioned widely apart in the clustering hierarchy. Although the system had been designed to investigate precisely such occurrences, users were confused by the results.

With this approach, as with those leading up to it, edges with different labels are bundled and represented by a single line. Analysts asked whether it would be possible to differentiate between the different types of edges in every bundle, using colors or text annotations, for instance. This problem was solved, in a more scalable fashion, with the technique discussed in Section 8.3.5.

8.5 Problem space revisited

To facilitate the discussion in this chapter, it is useful to consider the information visualization design space from two perspectives: the problem space and the solution space. In Section 8.2, a simple model for describing the problem space was introduced. The author described his understanding of the parts of this model and claimed that this knowledge was essential to investigate the solution space. That is, to design information visualization techniques for system analysis.

At the start of Section 8.3 it was noted that due to unclear requirements, the approach that was taken toward designing visualization techniques for state transition graphs was to develop experimental prototypes and to collaborate closely with users over an extended period of time. These two issues are mutually reinforcing. By building prototypes, a lot was learned about the problem space. This in turn allowed for designing better visualizations.

Below, this iterative design process is investigated in more general terms. Although the views discussed here are drawn from the specific experience of the work presented in this dissertation, it is argued that many insights are applicable to other contexts where information visualization may be applied.

8.5.1 Data

Although the importance of data is accepted in the visualization community, it is worth stressing that in order to design visualization methods, designers have to completely understand their users' data. This seems trivial, but as was shown, the author's understanding of state transition graphs evolved quite a bit during the course of his work. The traditional approach toward visualizing transition graphs is to consider them as being flat: nodes represent states, edges represent transitions and to understand the behavior described by them, users have to analyze their topological structure.

As shown in Section 2.3, it is true that a lot can be learned about systems by approaching the data in this way. However, system analysts are typically involved with the generation of their data. This involves specifying the behavior of a system in a formal language such as process algebra. Since variables introduced in such a specification are found back in the transition graph as node attributes, users understand what they mean (see Section 2.5.2). This characteristic of system analysis led to the development of new visualization techniques. Here the challenge was to assist users in applying knowledge

they already have to learn more about the data. By emphasizing the importance of edge labels a similar evolution was followed.

Reconsidering the data is an effective way of dealing with unclear requirements. By continually looking for aspects of system analysts' data that current visualization techniques do not highlight, and by questioning whether such emphasis would be beneficial, it was possible to develop a number of visualization techniques to assist users in considering their data in new manner.

It is also useful to try to identify different perspectives that can be taken on the data. With such an approach, for example, it was realized that system states can also be visualized as diagrams (see Section 8.3.3 and Chapter 5). Different views guide users in taking different conceptual perspectives on the data which support new types of analysis.

Sometimes there are related data types that are also relevant. For example, users were also interested in analyzing system traces (see Section 8.3.4). Considering related data helps to take a fresh look at the problem space and leads to new solutions. This can be seen in the way that the visualization of system traces led to more attention being paid to paths in the work described in Section 8.3.5, which was based on querying nodes and edges. It is also rewarding to investigate whether existing techniques can be reapplied to such data.

8.5.2 Users

It was surprising to discover that despite having a deep domain knowledge, which translates into a good understanding of node attributes and edge labels, users could not provide a list of typical questions they sought to answer about their data. Nor could they describe recurring tasks for which they wanted support.

The users were not being intentionally difficult or incompetent in verbalizing what they need to perform their work. To the contrary, they are highly motivated professionals. They work in an experimental domain where, on the one hand, they develop and refine formal system evaluation techniques rooted in computational theory. On the other hand, they apply these techniques to analyze systems.

Although it had been envisioned for visualization to assist analysts in this second part of their work, it soon became clear, however, that the burden of identifying opportunities for adding value with visualization would be shared. From this it is concluded that the role of a visualization designer entails much more than thinking up graphical presentations for neatly compartmentalized problems supplied by their users. Perhaps their role can best be compared to that of an architect who has to be a draftsman, designer, facilitator, engineer and other things, depending on the circumstances [20].

As time progressed, involvement with users got easier and more effective. The reason for this is that time is needed to build a relationship with users and to understand the problem space. For example, the author made an effort to read and become familiar with the application domain's literature. This enabled him to approach users with context related questions. Apart from valuable insight, the users' respect was also gained. They took recommendations more seriously and started approaching the author with suggestions and requests. Based on this voluntary feedback it was possible to dramatically improve the visualization techniques that were developed.

8.5.3 Application domain

Analysts' inability to express what visual support they needed to analyze state transition graphs also resulted from the context they work in. There are a number of issues that played a role:

- The systems modeled by transition graphs are extremely complex and require significant time and effort to understand. This often includes non-visual analysis.
- Typically only one or two analysts study a particular data set. This and the other factors discussed here make it more difficult for analysts to generalize their experience.
- The nature of the particular system being studied has a profound influence on the type of analysis being done. The exact insight being sought is not consistent.
- Often analysts are not sure what they are looking for. Their brief may be as vague as, "we think there is something wrong, but we are not sure what or why, please check."

The author expects that these aspects are not unique to system analysis. It was found, however, that users always have some knowledge about their data or the application domain they work in. The challenge was in finding out what this was and to use this as a point of access to users' data. In the context of this dissertation, by leveraging node attributes and edge labels, analysts were able to gain insight in terms of knowledge they already had.

Related to this, presenting data in terms of users' conceptual model of a problem can be very effective. Based on the author's experience there are two advantages of enabling users to capture and reuse this knowledge. First, they reaffirm what they already know. Second, they can use this as a familiar starting point to analyze their data. Although this was explored in a very literal way, with user-defined diagrams, the implications are more general.

8.6 Solution space revisited

To design interactive systems the following process is usually advocated [71]: (1) identify requirements, (2) generate alternative designs, and (3) evaluate these designs. It is usually emphasized that a number of iterations of this cycle need to be completed before an adequate solution will be found.

The author agrees with this approach and to some extent this is also what the research reported on in this dissertation set out to achieve. At a high level, the two requirements introduced in Section 2.2 were identified. Next, a number of prototypes were designed and iteratively refined. Validation took the form of case studies, semi-structured interviews, use cases and informal user observations. The successful techniques discussed in previous chapters and revisited in Section 8.3 met these requirements:

- They enabled system analysts to form an intuition about their data by performing explorative analysis.
- They allowed analysts to investigate specific aspects of their data by supporting focused analysis.

8.6.1 Information visualization design

The traditional system design methodology, outlined above, assumes that tasks to support can be clearly defined. Systems are then designed to support these tasks. As has been explained, for various reasons this knowledge was not available. It may also be argued that if this had been the case (that is, if system analysts were able to clearly define their needs) they would have already developed (non-visual) tools to support these. It may be argued that this is true for many other application domains of visualization. As a result, iterative prototyping is fundamental, both to get better solutions and to increase understanding of the problem.

To design information visualizations, a thorough understanding of the data, the users and the application domain is essential. This knowledge informs design decisions, while new solutions (successful or failed) cast a fresh perspective on the problem space. Evaluation of which aspects of such solutions work, which do not, and why, leads to a better understanding of the problem and establishes an iterative feedback loop.

To achieve the above, it is useful to position solutions in the context of the problem space. Figure 8.4 shows the approach that was taken toward information visualization design in this dissertation. Visualizations, in the form of different prototypes developed by the visualization designer, serve as bridges to provide users with insight in their data. Although they explore different parts of the solution space, they are all firmly rooted in the application domain. Such an approach is also advocated by Plaisant [58]. She argues that it allows the designer to consider the feasibility of visualization techniques by involving real users who analyze their actual data in their natural working environment.

To develop solutions, several points of departure can be taken. First, as Prototype A in Figure 8.4 illustrates, the data can be taken as a starting point. This entails the identification of certain perspectives or aspects of the data that should be highlighted. Consequently, the challenge is to come up with visual techniques that emphasize these. Second, user needs can dictate design decisions (see Prototype B in Figure 8.4). This is similar to the traditional approach of designing interactive systems outlined above. However, as with the other approaches discussed here, the relationship between the solution and its context is stressed. Third, it is possible to take the application domain as point of departure (Prototype C in Figure 8.4). For example, a specific methodology may be identified and a visualization technique designed to support it.

Information visualization design does not always imply conceiving of brand new ideas. It also includes exposing users to existing techniques that may be applicable. Alternatively, existing techniques can be combined or can serve as initial inspiration. By repeated refinement, such solutions gradually diverge from previous work to become valid techniques in their own right. This fourth approach is illustrated by Prototype D in Figure 8.4. As has been pointed out, not all attempts will be successful and as Prototype E

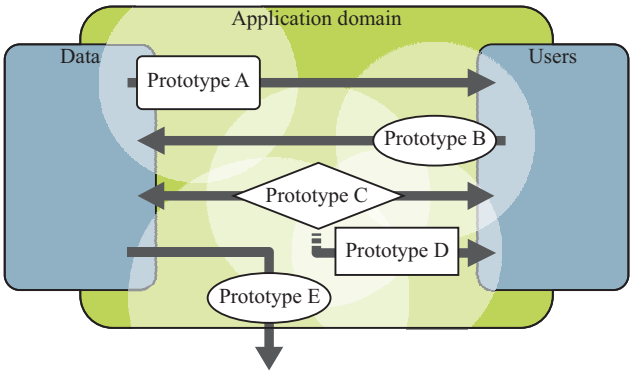


Figure 8.4: Information visualization design.

shows, sometimes detours doomed to failure are taken.

Ideally, the designer will have access to data, users and an application domain. However, visualizations are designed for different purposes [94] and as a result this will not always be the case. For instance, curiosity-driven visualization design typically does not have an application domain defined at the outset and the developer and user is likely to be the same person [93]. In such cases, the only suitable approach will be the one taken with Prototype A. In other cases, the users and application domain may be known but access to real data is restricted or confidential (financial data, for instance). Here the approaches taken with Prototype B and C may be applicable. In both the above cases, initial work is likely to spawn further experimentation (Prototype D).

Fortunately, due to the evolutionary nature of the approach to design sketched above, successes and failures assist the designer. Knowledge of the data, the users and the application domain increase and become more accurate over time. This may be compared to investigating a dark room with a torch. By casting light on different parts of the problem space, the designer gradually builds up a mental map of the issue. This is illustrated in Figure 8.4 by the different prototypes illuminating different parts of the data, users and application domain.

The approach outlined above, leads to different solutions as time progresses. This is illustrated by the varying shapes of the prototypes in Figure 8.4. This evolution is discrete at times, for example, when a totally new solution is developed. However, it can also be more continuous as solutions gradually transform. When the solution space is sampled in this way, the sum of the insights gained by the designer increases dramatically over time (consider the total area illuminated in Figure 8.4), resulting in more effective solutions.

8.6.2 Prototyping

To put the above into practice, an exploratory approach is advocated where a number of prototypes are developed in collaboration with users. Consider how ideas on visualizing multivariate nodes have evolved as described in this dissertation. First, techniques based on user-steered projection were developed and tested by implementing a prototype which was iteratively refined (see Section 8.3.1). After evaluating this prototype, it was realized that the positioning of nodes as a function of node attributes was powerful, but it would be even more useful if location had meaning. As shown in Section 8.3.2 this led to a new visualization technique.

Prototyping requires an investment of time and effort, but the author does not know a more effective and direct way of developing and refining ideas. Not every solution is perfect (see Section 8.4), but the knowledge taken from it to a new solution is invaluable. It is important to learn *how*, at a conceptual level, insight should present itself to the user, as opposed to finding out *what* the right solution is. A reasonable starting assumption is that no single optimal solution for a problem exists. Rather than trying to fine tune a single technique, it is more effective and more enlightening to explore the solution space. When a promising idea is uncovered, it is then possible to nurture it to a mature solution.

The techniques developed for this dissertation are not restricted to analyzing graphs that model system behavior. It is argued that they should be reusable for multivariate graphs, in general. Still, often targeting and solving a niche problem can be very effective. For instance, the custom visualization developed for the automated parking garage was very helpful for addressing that particular problem but also served as inspiration for more general visualization techniques (see Section 8.3.3 and 8.3.4). As argued by Plaisant [58], it was found that one reason why system analysts were willing to adopt the techniques presented here was because they were developed for their specific application domain.

Prototyping also leads to unexpected results. For instance, it was not anticipated that the tools developed in the context of this dissertation would be used as communication aids. In this regard, assumptions about the problem and solution space are often wrong. As shown, false beliefs were discovered at various stages of the work presented in previous chapters. Visualization designers should continually look out for and critically reconsider their assumptions.

8.6.3 Evaluation

Evaluation is a recurring thorn in the side of visualization research [54]. There seems to be agreement that every solution should be evaluated, but researchers disagree as to how this should be done. Throughout the project described in this dissertation, evaluation also was an issue and a topic of discussion.

Due to unclear requirements, a decision was made to move quickly and to investigate a number of different solutions. Fine-tuning a single prototype would not have resulted in sufficiently examining the solution space. This exploratory approach, combined with the fact that there were not specific tasks to support, ruled out traditional usability testing, where users are judged on the accuracy or efficiency of completing certain low-level tasks [71]. Furthermore, the user group was rather small (see Section 8.2.2), so statistical

evaluation was not feasible. Because a decision was made to deviate from the few existing techniques for visualizing state transition graphs (Section 8.3), comparative studies were not viable.

Consequently, the approach taken toward evaluation was based on longitudinal studies, as advocated by Shneiderm and Plaisant [76]. As explained in Section 2.6 an informal and anecdotal approach was adopted with close user collaboration.

Longitudinal studies are becoming increasingly popular to evaluate information visualization techniques. Examples include the evaluation of visual support for knowledge discovery in high-dimensional data [70] and the evaluation of a visualization system for analyzing system management time series [49]. These studies and the results presented in this dissertation confirm that such evaluation offers a number of advantages. First, it results in refined visualization techniques. Second, it results in a better understanding of the general principles and guidelines for the design of tools for specific application domains. Third, users achieve their goal of gaining a better understanding of their data.

The model for information visualization design proposed here does not rule out other evaluation techniques (see Figure 8.4). It does imply, though, that close collaboration with users and access to them and their work environment is indispensable.

In the context of this dissertation, the aim of evaluation was not only to convince the information visualization community of the utility of visualization techniques for state transition graphs. Perhaps more importantly, system analysts had to be convinced of the applicability of visualization for their work. As Plaisant [58] puts it, "to be convincing, utility [of information visualization] needs to be demonstrated in a real setting, that is a given application domain and set of users. For researchers, choosing and preparing convincing examples goes a long way in attracting potential adopters." In this light, an encouraging consequence of the techniques presented in previous chapters is that many of the prototypes that were developed have subsequently been included in a tool set for system analysis [79].

Chapter 9

Conclusion

In the preceding chapters a number of approaches for visualizing state transition graphs were presented. This final chapter contains concluding remarks about the work discussed in this dissertation. First, the main contributions of this dissertation are summarized and, second, their implications and opportunities for future work are discussed.

9.1 Contributions

State transition graphs are incarnations of system behavior. Their nodes represent possible system states and their directed edges represent transitions between states. Analysts attempt to capture only essential behavior in the formal specifications from which transition graphs are generated, for instance, by excluding implementation specific details. As a result, transition graphs accurately reflect the fundamental behavior of the systems they model. However, they often contain large numbers of nodes and edges. They also describe behavior at a low abstraction level. These factors hamper analysis and insight.

In this dissertation the interactive visualization of state transition graphs was investigated. More precisely, the following research question, first introduced in Chapter 1, was addressed:

Research question. How can interactive visualization be used to gain insight into state transition graphs?

In response to this question, it was shown that visualization can assist in understanding the complex system behavior represented by state transition graphs. On the one hand, by supporting explorative visual analysis, it was shown that users are enabled to gain a better intuitive understanding of their data. On the other hand, by supporting focused analysis, visualizations of transition graphs enable users to investigate particular features and answer specific questions about their data. Moreover, communication between users themselves and between users and other stakeholders was substantially enhanced by giving visual form to inherently abstract data.

Table 9.1: State transition graph visualization.

Visualization		*Approach*			*Execution*			*Results*			
		Node attributes	*Edge labels*	*Paths*	*Projection*	*Clustering*	*Diagrams*	*Nodes*	*Edges*	*Relation nodes & edges*	*Paths*
Selection & projection		✓			✓			✓			
Attribute-based clusters		✓				✓		✓		✓	
User-defined diagrams	*Clust view*	✓				✓	✓	✓		✓	
	Sim view	✓		✓			✓	✓	✓	✓	✓[1]
	Insp view	✓					✓	✓			
Multiple views traces	*Time series*	✓		✓			✓	✓			✓
	Graph view	✓				✓	✓	✓		✓	
	Insp view	✓					✓	✓			
Query nodes & edges	*Detail*	✓	✓			✓		✓	✓	✓	✓
	Context	✓	✓	✓		✓		✓	✓	✓	✓

[1] In a small local neighborhood.

The results presented in previous chapters take advantage of multivariate data associated with nodes and edges in state transition graphs. Such an approach was taken because system analysts have a very good understanding of the semantics, or meaning, of the associated data. This is in contrast to previous approaches where the emphasis was on the topology or the structure represented by edges in such graphs.

A number of techniques for the visualization of state transition graphs were developed and discussed. In Chapter 3 a number of ways to project transition graphs to the 2D plane based on a user-selected subset of node attributes were investigated. Chapter 4 introduced a technique to generate abstract, much simplified, representations of state transition graphs by employing attribute-based clustering. Chapter 5 presented a technique to investigate transition graphs with custom user-defined diagrams. In Chapter 6 an approach for visualizing paths, or traces, in transition graphs was introduced. Finally, Chapter 7 addressed the issue of also considering edge labels in the visualization of state transition graphs.

Table 9.1 provides a high level summary of the results presented in this dissertation. The visualization techniques that were discussed are listed top-to-bottom, in chronological order, at the left of the table. They are considered in terms of the approach taken, their execution and the obtained results.

Approach. Three main approaches toward visualizing state transition graphs were investigated by focusing on node attributes, on edge labels or on paths (linear sequences of neighboring nodes connected by shared edges). Most of the techniques emphasized node attributes. Paths were occasionally considered. Only the last experiment (querying nodes and edges, Chapter 7) treated node attributes, edge labels and paths.

Execution. The modes of execution can be generally categorized as either based on projection, on clustering or on diagrams. Table 9.1 shows that most techniques were based on attribute-based clustering and user-defined diagrams. As argued in the previous chapter, these techniques were more effective than those based on projection. This holds for interpreting the results, but also improved performance since fewer shapes had to be rendered on screen.

Results. In broad terms, the techniques discussed in this dissertation enabled users to gain insight into nodes, into edges, into the relations between nodes and edges, or into paths. All five techniques showed that existing domain knowledge can be effectively leveraged to learn more about state transition graphs. For example, it was shown that when users can interpret the positions of clusters of nodes and bundles of edges in terms of knowledge they already have, it is easier to learn more about a graph.

Most of the techniques assisted users in shedding light on nodes and the relationship between nodes and edges. The first instance of having edges play a more significant role was the simulation view (user-defined diagrams, Chapter 5). Users were also able to learn more about paths by traversing the graph using the small local neighborhood provided by this view. By its nature, the work that explored multiple views on traces (Chapter 6) also places more emphasis on paths. The technique discussed in Chapter 7, querying nodes and edges, was the only one that enabled users to learn more about nodes, edges, relationships between nodes and edges, and paths.

The work reported on in this dissertation was exploratory in nature and involved close collaboration with users. Few techniques had been developed to visualize state transition graphs in the past. Out front, system analysts also found it very difficult to formulate what the role of visualization should be. Consequently, it was not possible to accurately define requirements for visualization systems for transition graphs.

A decision was therefore taken to attempt to cover as much ground as possible by considering many different approaches. To achieve this, many prototypes were developed and tuned based on direct input from users. To support this iterative cycle of design, informal and anecdotal evaluation was used. This included case studies, semi-structured interviews and informal user observations.

9.2 Future work

Despite positive user feedback there are a range of opportunities for future work. Table 9.1 suggests a number of opportunities for extending the visualization techniques presented in this dissertation. These are considered below.

Nodes, edges and paths. In the work on querying nodes and edges, presented in Chapter 7, the three different approaches investigated in this dissertation (node attributes, edge labels and paths) were integrated in a single solution for the first time. In the first instance, future work could further investigate the combination of the three approaches. For example, more effective and scalable solutions for dealing with large numbers of unique edge labels could be developed.

Another promising avenue for research is to place more emphasis on paths. While users intuitively understand the notion of traversing a sequence of nodes and edges, providing adequate visual support for investigating paths is a challenge. For example, it is difficult to compare different paths, especially when they intersect. The context view, developed as part of the work on querying nodes and edges (see Chapter 7), is a first step in this direction. User feedback supports the hypothesis that there is user demand for analyzing paths, but also highlights the limitations of this approach.

Clustering and diagrams. A straightforward application of user-defined diagrams would be to incorporate them in other visualization techniques, such as topology-based approaches. Judging by the success of combining attribute-based clustering with diagrams (user-defined diagrams, Chapter 5, and multiple views on traces, Chapter 6) the work on querying nodes and edges (see Chapter 7) could also be extended with diagrams.

In particular, it is expected that this could combine the advantages of clustering nodes and bundling edges with the intuitive interpretation of the diagram-based simulation view presented in Chapter 5. Both represent the direction implied by transitions in a left-to-right fashion. Furthermore, by incorporating diagrams in the context view (Chapter 7), the limitation of only being able to consider paths in a local neighborhood could be addressed.

User-defined diagrams were only investigated in their capacity to represent individual system states. This suggests at least two further extensions of the diagram-based approach. First, it could be investigated how edge labels could be integrated better. For instance, is it possible to parameterize edge labels so that they also influence the appearance of diagrammatic representations of system behavior?

Second, the problem of efficiently and effectively comparing a large number of diagrams, representing many different states, to identify similarities and differences, remains a challenge. For example, to consider the states contained in a particular cluster, users have to animate through their diagrammatic representations (see Chapter 5). An obvious alternative would be to blend a stack of overlaid diagrams. However, since this is unlikely to scale beyond four to seven diagrams, there is a need for more scalable solutions.

Graph topology. Combining an attribute-based node clustering facility with a visual representation that emphasizes the topological structure of bundled edges remains an unsolved challenge. The failed attempt at solving this problem by providing two linked views, mentioned in the previous chapter, attests to this.

Archambault et al. [3] have taken a promising first step in this direction. They provide a single view where the user specifies clusters in terms of node attributes. Based on such a clustering, a topology-based layout is then calculated. However, this layout shifts as the user refines or collapses clusters. As a result, the positions of clusters are harder to inter-

pret in terms of the associated node attributes. A possible solution is to lock the positions of lower level clusters and to apply topology-based positioning algorithms to sub-clusters only within some neighborhood defined in terms of the positions of their parents.

There are also opportunities for further research outside the scope of Table 9.1. These include considering the scalability and general applicability of the techniques that were presented.

Scalability. In terms of size and complexity, the graphs considered in this dissertation were not trivial. It has been shown that graphs with tens of thousands of nodes and hundreds of thousands of edges can be effectively handled.

In this respect, the results presented here are an important first step in analyzing vast and complicated models of system behavior. Yet, even larger graphs are often encountered in practice. For instance, it is not uncommon to encounter transition graphs where nodes number in the millions. Some systems even have infinitely many states. This poses a formidable challenge that has not been completely solved.

Applicability. In more general terms, the techniques introduced in this dissertation addressed two of the hardest problems in information visualization: the visualization of large graphs and the visualization of large multivariate data sets. In this respect, an open issue is the applicability of the work presented here to arbitrary multivariate graphs.

Future research could include considering the developed techniques to inspect, for example, networks that model email traffic or social networks. This will reveal which approaches easily migrate to other applications and why. Also, it may lead to the identification of additional required functionality or extensions. These improvements are likely, in turn, to be applicable to investigating state transition graphs.

The application of the techniques discussed in preceding chapters to more general data sets would result in a larger potential user population. This would also provide an opportunity for more formal statistically validated evaluation. One aspect that can, for instance, be investigated is the extent to which cognitive switches resulting from different perspectives on the same data sets, such as multiple views on traces (Chapter 6), lead to more insight.

In his Turing Award acceptance lecture [16], the 1972 laureate Edsger Dijkstra argued that as computer hardware has become orders of magnitude more powerful, reliable and predictable computer software has become a "gigantic problem." David Gelernter [22] states that, "a single programmer alone at his keyboard can improvise software machines of fantastic or even incomprehensible complexity." This sentiment is shared by many others, including John Holland [34] in the quote on Page 1 of this dissertation.

Fortunately, the increased power of computer hardware has presented new opportunities for harnessing the complexity of system behavior. One consequence is that even modest consumer-grade computers now offer real-time interactive computer graphics capabilities. This presents the opportunity to inspect and interact with visual representations of state transition graphs. As this dissertation shows, interactive visualization can, and should, become part of the system analyst's set of tools for analyzing system behavior.

Bibliography

[1] AGHA, G. Computing in pervasive cyberspace. *Communications of the ACM 51*, 1 (2008), 68–70.

[2] ANKERST, M., BERCHTOLD, S., AND KEIM, D. A. Similarity clustering of dimensions for an enhanced visualization of multidimensional data. In *Proceedings of the IEEE Symposium on Information Visualization* (Research Triangle Park, NC, USA, 1998), IEEE Computer Society Press, pp. 52–60.

[3] ARCHAMBAULT, D., MUNZNER, T., AND AUBER, D. GrouseFlocks: steerable exploration of graph hierarchy space. *IEEE Transactions on Visualization and Computer Graphics 14*, 4 (2008), 900–913.

[4] ARNOLD, A. *Finite Transition Systems: Semantics of Communicating Systems*. Prentice Hall, Hertfordshire, UK, 1994.

[5] ASIMOV, D. The grand tour: a tool for viewing multidimensional data. *SIAM Journal on Scientific and Statistical Computing 6*, 1 (1985), 128–143.

[6] BAKER, M. J., AND EICK, S. G. Space-filling software visualization. *Journal of Visual Languages and Computing 6*, 2 (1995), 119–133.

[7] BATTISTA, DI, G., EADES, P., TAMASSIA, R., AND TOLLIS, I. G. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, Upper Saddle River, NJ, USA, 1999.

[8] BEEK, VAN, D. A., MAN, K. L., RENIERS, M. A., ROODA, J. E., AND SCHIFFELERS, R. R. H. Syntax and consistent equation semantics of Hybrid Chi. *Journal of Logic and Algebraic Programming 68*, 1–2 (2006), 129–210.

[9] BERTIN, J. Graphics and graphic information processing. In *Readings in Information Visualization: Using Vision to Think*, S. K. Card, J. D. Mackinlay, and B. Shneiderman, Eds. Morgan Kaufman, San Francisco, CA, USA, 1999, pp. 62–65.

[10] BEZEM, M., AND GROOTE, J. F. A formal verification of the Alternating Bit Protocol in the calculus of constructions. Logic Group Preprint Series 88, Utrecht University, Department of Philosophy, the Netherlands, 1993.

[11] BLOM, S., FOKKINK, W., GROOTE, J. F., LANGEVELDE, VAN, I., LISSER, B., AND POL, VAN DE, J. μCRL: a toolset for analysing algebraic specifications. In *Proceedings of the International Conference on Computer Aided Verification* (Paris, France, 2001), vol. 2102 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 250–254.

[12] CARD, S. K., MACKINLAY, J. D., AND SHNEIDERMAN, B. *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann, San Francisco, CA, USA, 1999.

[13] CHAMBERS, J. M., CLEVELAND, W. S., KLEINER, B., AND TUKEY, P. A. *Graphical Methods for Data Analysis*. Chapman and Hall, New York, NY, USA, 1983.

[14] DAMS, D., GERTH, R., AND GRUMBERG, O. Abstract interpretation of reactive systems. *ACM Transactions on Programming Languages and Systems 19*, 2 (1997), 253–291.

[15] DEFANTI, T. A., BROWN, M. D., AND MCCORMICK, B. H. Visualization: expanding scientific and engineering research opportunities. *Computer 22*, 8 (1989), 12–25.

[16] DIJKSTRA, E. W. The humble programmer. *Communications of the ACM 15*, 10 (1972), 859–866.

[17] DORLING, D., AND BARFORD, A. What's wrong with this picture? *New Scientist 2550* (2006), 39–41.

[18] FOKKINK, W. *Introduction to Process Algebra*. Springer-Verlag, Secaucus, NJ, USA, 2000.

[19] FOLEY, J., DAM, VAN, A., FEINER, S., AND HUGHES, J. *Computer Graphics: Principles and Practice in C*, second ed. Addison-Wesley, Reading, MA, USA, 1995.

[20] FREDERICK, M. *101 Things I Learned in Architecture School*. MIT Press, Cambridge, MA, USA, 2007.

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

[22] GELERNTER, D. H. *Machine Beauty: Elegance and the Heart of Technology*. Basic Books, New York, NY, USA, 1998.

[23] GLASS, R. L. Sorting out software complexity. *Communications of the ACM 45*, 11 (2002), 19–21.

[24] GROOTE, J. F., MATHIJSSEN, A., RENIERS, M., USENKO, Y., AND WEERDENBURG, VAN, M. The formal specification language mCRL2. In *Methods for Modelling Software Systems* (Dagstuhl, Germany, 2007), no. 06351 in Dagstuhl Seminar Proceedings, Internationales Begegnungs- und Forschungszentrum für Informatik.

[25] HAM, VAN, F., WETERING, VAN DE, H., AND WIJK, VAN, J. J. Interactive visualization of state transition systems. *IEEE Transactions on Visualization and Computer Graphics 8*, 4 (2002), 319–329.

[26] HAM, VAN, F., AND WIJK, VAN, J. J. Beamtrees: compact visualization of large hierarchies. In *Proceedings of the IEEE Symposium on Information Visualization* (Boston, MA, USA, 2002), IEEE Computer Society Press, pp. 31–39.

[27] HAMER, J. Model based engineering of a paint factory with supervisory control theory. Master's thesis, Technische Universiteit Eindhoven, Department of Mechanical Engineering, Eindhoven, the Netherlands, 2007.

[28] HAO, M. C., DAYAL, U., KEIM, D. A., AND SCHRECK, T. Importance-driven visualization layouts for large time series data. In *Proceedings of the IEEE Symposium on Information Visualization* (Minneapolis, MN, USA, 2005), IEEE Computer Society Press, pp. 203–210.

[29] HARTIGAN, J. A. *Clustering Algorithms*. John Wiley & Sons, New York, NY, USA, 1975.

[30] HENDRIKS, M., NIEUWELAAR, VAN DEN, B., AND VAANDRAGER, F. Model checker aided design of a controller for a wafer scanner. *International Journal on Software Tools for Technology Transfer 8*, 6 (2006), 633–647.

[31] HERMAN, I., MELANÇON, G., AND MARSHALL, M. S. Graph visualization and navigation in information visualization: a survey. *IEEE Transactions on Visualization and Computer Graphics 6*, 1 (2000), 24–43.

[32] HOARE, C. A. R. *Communicating Sequential Processes*. Prentice Hall, Hertfordshire, UK, 1985.

[33] HOCHHEISER, H., AND SHNEIDERMAN, B. Dynamic query tools for time series data sets: timebox widgets for interactive exploration. *Information Visualization 3*, 1 (2004), 1–18.

[34] HOLLAND, J. H. *Emergence: From Chaos to Order*. Addison-Wesley Longman, Boston, MA, USA, 1998.

[35] HOLTZBLATT, K., AND JONES, S. Contextual inquiry: a participatory technique for system design. In *Participatory Design: Perspectives on System Design*, D. Schuler and A. Namioka, Eds. Lawrence Erlbaum Associates, Hillsdale, NJ, USA, 1993, pp. 177–210.

[36] INSELBERG, A., AND DIMSDALE, B. Parallel coordinates: a tool for visualizing multi-dimensional geometry. In *Proceedings of the IEEE Conference on Visualization* (San Francisco, CA, USA, 1990), IEEE Computer Society Press, pp. 361–378.

[37] JÉRON, T., AND JARD, C. 3D layout of reachability graphs of communicating processes. In *Proceedings of the DIMACS International Workshop on Graph Drawing* (Princeton, NJ, USA, 1995), vol. 894 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 25–32.

[38] JOHNSON, C., MOORHEAD, R., MUNZNER, T., PFISTER, H., RHEINGANS, P., AND YOO, T. S. *NIH-NSF Visualization Research Challenges*. IEEE Computer Society Press, Los Alamitos, CA, USA, 2006.

[39] JOLLIFFE, I. T. *Principal Component Analysis*, second ed. Springer-Verlag, Heidelberg, Germany, 2002.

[40] KAKOULIS, K. G., AND TOLLIS, I. G. On the edge label placement problem. In *Proceedings of the Symposium on Graph Drawing* (Berkeley, CA, USA, 1996), vol. 1190 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 241–256.

[41] KANDOGAN, E. Visualizing multi-dimensional clusters, trends, and outliers using star coordinates. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, CA, USA, 2001), ACM, pp. 107–116.

[42] KERR, B. THREAD ARCS: an email thread visualization. In *Proceedings of the IEEE Symposium on Information Visualization* (Seattle, WA, USA, 2003), IEEE Computer Society Press, pp. 211–218.

[43] KNUTH, D. E. *Literate Programming*. Center for the Study of Language and Information, Stanford, CA, USA, 1992.

[44] LEUSCHEL, M., AND TURNER, E. Visualizing larger state spaces in ProB. In *Proceedings of the International Conference of B and Z Users* (Guildford, UK, 2005), vol. 3455 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 6–23.

[45] LIN, J., KEOGH, E., AND LONARDI, S. Visualizing and discovering non-trivial patterns in large time series databases. *Information Visualization 4*, 2 (2005), 61–82.

[46] LIN, J., KEOGH, E., LONARDI, S., AND CHIU, B. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the ACM SIGMOD workshop on Research Issues in Data Mining and Knowledge Discovery* (San Diego, CA, USA, 2003), ACM, pp. 2–11.

[47] MAO, J.-Y., VREDENBURG, K., SMITH, P. W., AND CAREY, T. The state of user-centered design practice. *Communications of the ACM 48*, 3 (2005), 105–109.

[48] MATHIJSSEN, A., AND PRETORIUS, A. J. Verified design of an automated parking garage. In *Proceedings of the International Workshop on Formal Methods for Industrial Critical Systems* (Bonn, Germany, 2006), vol. 4346 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 165–180.

[49] MCLACHLAN, P., MUNZNER, T., KOUTSOFIOS, E., AND NORTH, S. LiveRAC: interactive visual exploration of system management time-series data. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems* (Florence, Italy, 2008), ACM, pp. 1483–1492.

[50] MIHALISIN, T., TIMLIN, J., AND SCHWEGLER, J. Visualization and analysis of multi-variate data: a technique for all fields. In *Proceedings of the IEEE Conference on Visualization* (San Diego, CA, USA, 1991), IEEE Computer Society Press, pp. 171–178.

[51] NARDI, B. A., AND ZARMER, C. L. Beyond models and metaphors: visual formalisms in user interface design. *Journal of Visual Languages and Computing 4*, 1 (1993), 5–34.

[52] NEUMANN, P., SCHLECHTWEG, S., AND CARPENDALE, S. ArcTrees: visualizing relations in hierarchical data. In *Proceedings of the Eurographics - IEEE VGTC Symposium on Visualization* (Leeds, UK, 2005), Eurographics Association, pp. 53–60.

[53] NORMAN, D. A. *Things that Make Us Smart: Defending Human Attributes in the Age of the Machine*. Addison-Wesley Longman, Boston, MA, USA, 1993.

[54] NORTH, C. Toward measuring visualization insight. *IEEE Computer Graphics and Applications 26*, 3 (2006), 6–9.

[55] PANG, J., FOKKINK, W., HOFMAN, R., AND VELDEMA, R. Model checking a cache coherence protocol for a Java DSM implementation. In *Proceedings of the International Symposium on Parallel and Distributed Processing* (Nice, France, 2003), IEEE Computer Society Press, pp. 238–248.

[56] PELLEGRINETTI, G., AND BENTSMAN, J. Nonlinear control oriented boiler modeling - a benchmark problem for controller design. *IEEE Transactions on Control Systems Technology 4*, 1 (1996), 57–64.

[57] PETERSON, J. L. Petri nets. *ACM Computing Surveys 9*, 3 (1977), 223–252.

[58] PLAISANT, C. The challenge of information visualization evaluation. In *Proceedings of the Working Conference on Advanced Visual Interfaces* (Gallipoli, Italy, 2004), ACM, pp. 109–116.

[59] PRETORIUS, A. J., AND WIJK, VAN, J. J. Multidimensional visualization of transition systems. In *Proceedings of the International Conference on Information Visualisation* (London, UK, 2005), IEEE Computer Society Press, pp. 323–328.

[60] PRETORIUS, A. J., AND WIJK, VAN, J. J. Visual analysis of multivariate state transition graphs. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 685–692.

[61] PRETORIUS, A. J., AND WIJK, VAN, J. J. Bridging the semantic gap: visualizing transition graphs with user-defined diagrams. *IEEE Computer Graphics and Applications 27*, 5 (2007), 58–66.

[62] PRETORIUS, A. J., AND WIJK, VAN, J. J. Multiple views on system traces. In *Proceedings of the IEEE Pacific Visualization Symposium* (Kyoto, Japan, 2008), IEEE Computer Society Press, pp. 95–102.

[63] PRETORIUS, A. J., AND WIJK, VAN, J. J. Visual inspection of multivariate graphs. *Computer Graphics Forum 27*, 3 (2008), 967–974.

[64] RAO, R., AND CARD, S. K. The table lens: merging graphical and symbolic representations in an interactive focus+context visualization for tabular information. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems* (Boston, MA, USA, 1994), ACM, pp. 401–408.

[65] RESNIKOFF, H. L. *The Illusion of Reality*. Springer-Verlag, New York, NY, USA, 1989.

[66] RIBARSKY, W., AYERS, E., EBLE, J., AND MUKHERJEA, S. Glyphmaker: creating customized visualizations of complex data. *Computer 27*, 7 (1994), 57–64.

[67] SARKAR, M., AND BROWN, M. H. Graphical fisheye views of graphs. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems* (Monterey, CA, USA, 1992), ACM, pp. 83–91.

[68] SCAIFE, M., AND ROGERS, Y. External cognition: how do graphical representations work? *International Journal of Human-Computer Studies 45*, 2 (1996), 185–213.

[69] SEO, J., AND SHNEIDERMAN, B. A rank-by-feature framework for interactive exploration of multidimensional data. *Information Visualization 4*, 2 (2005), 96–113.

[70] SEO, J., AND SHNEIDERMAN, B. Knowledge discovery in high-dimensional data: case studies and a user survey for the rank-by-feature framework. *IEEE Transactions on Visualization and Computer Graphics 12*, 3 (2006), 311–322.

[71] SHARP, H., ROGERS, Y., AND PREECE, J. *Interaction Design: Beyond Human-Computer Interaction*, second ed. John Wiley & Sons, New York, NY, USA, 2006.

[72] SHIFLET, A. B., AND SHIFLET, G. W. *Introduction to Computational Science: Modeling and Simulation for the Sciences*. Princeton University Press, Princeton, NJ, USA, 2006.

[73] SHNEIDERMAN, B. Tree visualization with tree-maps: a 2-d space-filling approach. *ACM Transactions on Graphics 11*, 1 (1992), 92–99.

[74] SHNEIDERMAN, B. The eyes have it: a task by data type taxonomy for information visualizations. In *Proceedings of the IEEE Symposium on Visual Languages* (Boulder, CO, USA, 1996), IEEE Computer Society Press, pp. 336–343.

[75] SHNEIDERMAN, B., AND ARIS, A. Network visualization by semantic substrates. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 733–740.

[76] SHNEIDERMAN, B., AND PLAISANT, C. Strategies for evaluating information visualization tools: multi-dimensional in-depth long-term case studies. In *Proceedings of the AVI Workshop on Beyond Time and Errors* (Venice, Italy, 2006), ACM, pp. 1–7.

[77] SHRINIVASAN, Y. B., AND WIJK, VAN, J. J. Supporting the analytical reasoning process in information visualization. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems* (Florence, Italy, 2008), ACM, pp. 1237–1246.

[78] SPENCE, R. *Information Visualization*. Addison-Wesley, Harlow, UK, 2000.

[79] TECHNISCHE UNIVERSITEIT EINDHOVEN, DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE. mCRL2 website. http://www.mcrl2.org, 2008.

[80] THOMAS, J. J., AND COOK, K. A. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. IEEE Computer Society Press, Los Alamitos, CA, USA, 2005.

[81] TUFTE, E. R. *The Visual Display of Quantitative Information*, second ed. Graphics Press, Cheshire, CT, USA, 2001.

[82] TVERSKY, B., MORRISON, J. B., AND BÉTRANCOURT, M. Animation: can it facilitate? *International Journal of Human-Computer Studies 57*, 4 (2002), 247–262.

[83] TWEEDIE, L. Characterizing interactive externalizations. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, GA, USA, 1997), ACM, pp. 375–382.

[84] VERBEEK, H. M. W., PRETORIUS, A. J., AALST, VAN DER, W. M. P., AND WIJK, VAN, J. J. Assessing state spaces using Petri net synthesis and attribute-based visualization. *LNCS Transactions on Petri Nets and Other Models of Concurrency* (2008).

[85] WAGNER, F., AND WOLFF, A. A combinatorial framework for map labeling. In *Proceedings of the International Symposium on Graph Drawing* (Montreal, Quebec, Canada, 1998), vol. 1547 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 316–331.

[86] WARD, M. O. XmdvTool: integrating multiple methods for visualizing multivariate data. In *Proceedings of the IEEE Conference on Visualization* (Washinton, DC, USA, 1994), IEEE Computer Society Press, pp. 326–333.

[87] WARE, C. *Information Visualization: Perception for Design*. Morgan Kaufmann, San Francisco, CA, USA, 2000.

[88] WARE, C. Designing with a $2\frac{1}{2}$D attitude. *Information Design Journal 10*, 3 (2001), 258–265.

[89] WATTENBERG, M. Sketching a graph to query a time-series database. In *Extended abstracts of the ACM SIGCHI Conference on Human Factors in Computing Systems* (Seattle, WA, USA, 2001), ACM, pp. 381–382.

[90] WATTENBERG, M. Arc diagrams: visualizing structure in strings. In *Proceedings of the IEEE Symposium on Information Visualization* (Boston, MA, USA, 2002), IEEE Computer Society Press, pp. 110–116.

[91] WATTENBERG, M. Visual exploration of multivariate graphs. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems* (Montreal, Quebec, Canada, 2006), ACM, pp. 811–819.

[92] WEBER, M., ALEXA, M., AND MÜLLER, W. Visualizing time-series on spirals. In *Proceedings of the IEEE Symposium on Information Visualization* (San Diego, CA, USA, 2001), IEEE Computer Society Press, pp. 7–13.

[93] WIJK, VAN, J. J. Bridging the gaps. *IEEE Computer Graphics and Applications 26*, 6 (2006), 6–9.

[94] WIJK, VAN, J. J. Views on visualization. *IEEE Transactions on Visualization and Computer Graphics 12*, 4 (2006), 421–432.

[95] WIJK, VAN, J. J., LIERE, VAN, R., AND MULDER, J. D. Bringing computational steering to the user. In *Proceedings of the Conference on Scientific Visualization* (Dagstuhl, Germany, 1997), IEEE Computer Society Press, pp. 304–313.

[96] WIJK, VAN, J. J., AND SELOW, VAN, E. R. Cluster and calendar based visualization of time series data. In *Proceedings of the IEEE Symposium on Information Visualization* (San Francisco, CA, USA, 1999), IEEE Computer Society Press, pp. 4–9.

[97] WONG, P. C., MACKEY, P., PERRINE, K., EAGAN, J., FOOTE, H., AND THOMAS, J. Dynamic visualization of graphs with extended labels. In *Proceedings of the IEEE Symposium on Information Visualization* (Minneapolis, MN, USA, 2005), IEEE Computer Society Press, pp. 73–80.

[98] YANG, J., WARD, M. O., RUNDENSTEINER, E. A., AND HUANG, S. Visual hierarchical dimension reduction for exploration of high dimensional datasets. In *Proceedings of the Eurographics - IEEE VGTC Symposium on Visualization* (Grenoble, France, 2003), Eurographics Association, pp. 19–28.

[99] ZHANG, J., AND NORMAN, D. A. Representations in distributed cognitive tasks. *Cognitive Science 18*, 1 (1994), 87–122.

List of Publications

The work presented in this dissertation is based on the following publications:

PRETORIUS, A. J., AND WIJK, VAN, J. J. Multidimensional visualization of transition systems. In *Proceedings of the International Conference on Information Visualisation* (London, UK, 2005), IEEE Computer Society Press, pp. 323–328. (Chapter 3)

PRETORIUS, A. J., AND WIJK, VAN, J. J. Visual analysis of multivariate state transition graphs. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 685–692. (Chapter 4)

PRETORIUS, A. J., AND WIJK, VAN, J. J. Bridging the semantic gap: visualizing transition graphs with user-defined diagrams. *IEEE Computer Graphics and Applications 27*, 5 (2007), 58–66. (Chapter 5)

MATHIJSSEN, A., AND PRETORIUS, A. J. Verified design of an automated parking garage. In *Proceedings of the International Workshop on Formal Methods for Industrial Critical Systems* (Bonn, Germany, 2006), vol. 4346 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 165–180. (Chapter 5, Section 5.1)

VERBEEK, H. M. W., PRETORIUS, A. J., AALST, VAN DER, W. M. P., AND WIJK, VAN, J. J. Assessing state spaces using Petri net synthesis and attribute-based visualization. *LNCS Transactions on Petri Nets and Other Models of Concurrency* (2008). (Chapter 5, Section 5.7)

PRETORIUS, A. J., AND WIJK, VAN, J. J. Multiple views on system traces. In *Proceedings of the IEEE Pacific Visualization Symposium* (Kyoto, Japan, 2008), IEEE Computer Society Press, pp. 95–102. (Chapter 6)

PRETORIUS, A. J., AND WIJK, VAN, J. J. Visual inspection of multivariate graphs. *Computer Graphics Forum 27*, 3 (2008), 967–974. (Chapter 7)

Summary

State transition graphs are important in computer science and engineering where they are used to analyze the behavior of computer-based systems. In such a graph nodes represent states a system can be in. Links, or directed edges, represent transitions between states.

Research in visualization investigates the application of interactive computer graphics to understand large and complex data sets. Large state transition graphs fall into this category. They often contain tens of thousands of nodes, or more, and tens to hundreds of thousands of edges. Also, they describe system behavior at a low abstraction level. This hinders analysis and insight.

This dissertation presents a number of techniques for the interactive visualization of state transition graphs. Much of the work takes advantage of multivariate data associated with nodes and edges. Using an experimental approach, several new methods were developed in close collaboration with a number of users. The following approaches were pursued:

- *Selection and projection.* This technique provides the user with visual support to select a subset of node attributes. Consequently, the state transition graph is projected to 2D and visualized in a second, correlated visualization.

- *Attribute-based clustering.* By specifying subsets of node attributes and clustering based on these, the user generates simplified abstractions of a state transition graph. Clustering generates hierarchical, relational, and metric data, which are represented in a single visualization.

- *User-defined diagrams.* With this technique the user investigates state transition graphs with custom diagrams. Diagrams are parameterized by linking their graphical properties to the data. Diagrams are integrated in a number of correlated visualizations.

- *Multiple views on traces.* System traces are linear paths in state transition graphs. This technique provides the user with different perspectives on traces.

- *Querying nodes and edges.* Direct manipulation enables the user to interactively inspect and query state transition graphs. In this way relations and patterns can be investigated based on data associated with nodes and edges.

This dissertation shows that interactive visualization can play a role during the analysis of state transition graphs. The ability to interrogate visual representations of such graphs allows users to enhance their knowledge of the modeled systems. It is shown how the above techniques enable users to answer questions about their data. A number of case studies, developed in collaboration with system analysts, are presented.

Finally, solutions to challenges encountered during the development of the visualization techniques are discussed. Insights generic to the field of visualization are considered and directions for future work are recommended.

Samenvatting

Eindige automaten (state transition graphs) spelen een belangrijke rol in informatica en techniek. Deze grafen worden ondermeer gebruikt om het gedrag van computergestuurde systemen te analyseren. In dergelijke grafen stellen knooppunten (nodes) toestanden voor waarin een systeem zich kan bevinden. Gerichte zijden (edges) stellen overgangen (transities) tussen toestanden voor.

In visualisatieonderzoek wordt de toepassing van interactieve computergrafiek voor het begrijpen van grote en complexe dataverzamelingen bestudeerd. Eindige automaten zijn een voorbeeld van dit soort data. Veelal bestaan ze uit tienduizenden, of meer, knopen en tien- tot honderdduizenden zijden. Bovendien beschrijven ze systeemgedrag op een laag abstractieniveau, wat analyse en inzicht bemoeilijkt.

In dit proefschrift worden een aantal technieken voorgesteld voor de interactieve visualisatie van eindige automaten. De multivariabele data geassocieerd met de knopen en zijden van dergelijke grafen vormden het vertrekpunt van dit onderzoek. Door middel van een experimentele aanpak werden nieuwe technieken ontwikkeld, in nauwe samenwerking met gebruikers. De volgende uitgangspunten werden onderzocht:

- *Selectie en projectie.* Deze techniek verschaft de gebruiker visuele ondersteuning bij het selecteren van een deelverzameling van knoopattributen. Vervolgens wordt de eindige automaat geprojecteerd naar 2D en weergegeven in een tweede gecorreleerde visualisatie.

- *Attribuutgebaseerde clustering.* Door deelverzamelingen van knoopattributen te specificeren, en door knooppunten te clusteren op basis van deze attributen, genereert de gebruiker een vereenvoudigde abstractie van een eindige automaat. Het clusteren resulteert in hiërarchische, relationele en metrische data die weergegeven worden in één visualisatie.

- *Gebruikergedefinieerde diagrammen.* Met deze techniek onderzoekt de gebruiker eindige automaten aan de hand van zelfgespecificeerde diagrammen. Diagrammen worden geparametriseerd door de koppeling van hun grafische eigenschappen aan de data. De diagrammen worden geïntegreerd in gecorreleerde visualisaties.

- *Meerdere perspectieven op paden.* Systeempaden (traces) zijn lineaire paden in eindige automaten. Deze techniek biedt de gebruiker meerdere perspectieven op systeempaden.

- *Inspectie van knopen en zijden.* Directe manipulatie laat de gebruiker toe om grafen interactief te inspecteren en bevragen. Op deze wijze kunnen relaties en patronen onderzocht worden op basis van de data die geassocieerd zijn met de knopen en zijden.

Dit proefschrift toont aan dat interactieve visualisatie een rol kan vervullen in de analyse van eindige automaten. De mogelijkheid visuele voorstellingen van deze grafen te bestuderen laat gebruikers toe hun kennis van het gemodelleerde systeem uit te breiden. Dit proefschrift toont hoe de bovenvermelde technieken gebruikers in staat stellen vragen over hun data te beantwoorden. Een aantal gevalstudies, ontwikkeld in samenwerking met systeemanalisten, worden beschreven.

Daarnaast worden oplossingen besproken voor problemen die frequent voorkwamen tijdens het ontwikkelen van de bovenvermelde visualisatietechnieken. Inzichten eigen aan het veld van visualisatie worden besproken en aanbevelingen worden gedaan voor toekomstig onderzoek.

Curriculum Vitae

Andries Johannes (Hannes) Pretorius was born on 21 September 1977 in Pretoria, South Africa, where he completed his secondary education at Die Hoërskool Menlopark. In 2002 he graduated from the University of Pretoria with a BSc Mathematics and a BIS Multimedia. Both degrees were awarded with distinction. After graduation, he worked for Accenture, a management consultancy, in South Africa. In 2003 he enrolled for an MSc in Computer Science at the Vrije Universiteit Brussel, Belgium, and graduated with greatest distinction in 2004. In October 2004 he started his PhD research at the Department of Mathematics and Computer Science, Eindhoven University of Technology, the Netherlands. For this research, he developed interactive visualization techniques to analyze large state transition graphs, networks that model system behavior. This research was conducted under the supervision of Professor Jarke J. van Wijk and funded by the Netherlands Organisation for Scientific Research (NWO). The results of his PhD research, completed in 2008, were published in a number of articles in international conference proceedings and in journals. He is married to Tineke Brunfaut, a linguist, with whom he has a son, Casper.