Visual Analytics for Production and Transportation Systems

Von der Graduate School of Excellence Advanced Manufacturing Engineering der Universität Stuttgart zur Erlangung der Würde eines Doktor-Ingenieurs (Dr.-Ing.) genehmigte Abhandlung

Vorgelegt von

Michael Steffen Wörner

aus Schwäbisch Gmünd

Hauptberichter:	Prof. Dr. Thomas Ertl
Mitberichter:	Prof. Dr.-Ing. Prof. E. h. Dr.-Ing. E. h. Dr. h. c. mult. Engelbert Westkämper
	Prof. Giuseppe Santucci
Tag der mündlichen Prüfung:	27. Juni 2014

Institut für Visualisierung und Interaktive Systeme der Universität Stuttgart

2014

Acknowledgements

I would like to extend my sincerest thanks and gratitude to everyone who—in one way or another—contributed to the successful completion of this thesis and the research it describes. I thank my primary advisor Thomas Ertl for accepting me as a doctoral candidate and for offering his invaluable guidance and support throughout the process. I also thank Engelbert Westkämper and Giuseppe Santucci for their interest in my work and for acting as co-examiners for this thesis.

The Graduate School of Excellence advanced Manufacturing Engineering (GSaME) provided ample opportunity for gaining valuable insights into the fields of manufacturing and business administration through courses, conferences, and company visits and greatly simplified getting in touch with researchers from other fields. GSaME also provided the funding for much of my research.

Thank you to my colleagues at the Institute for Visualization and Interactive Systems and the Visualization Research Centre. I enjoyed the collaboration with Florian Heimerl, Steffen Koch, Robert Krüger, and Dennis Thom and the interesting and often instructive discussions with Benjamin and Markus Höferlin and Harald Sanftmann. I specifically thank Harald Bosch for his work on our joint VAST contest contribution and Martin Falk for our discussions on the finer points of typography and layout. Thank you, too, to the students I supervised, most notably Michael Metzger, who implemented one of the presented systems as part of his diploma thesis.

Stuttgarter Straßenbahnen AG kindly provided an extensive public transport data set and insightful feedback from a domain expert's perspective. Trapeze Switzerland GmbH assisted with interpreting the raw data. The machine diagnostics data set was made available by a company preferring to remain unnamed to protect its competitive edge.

Not least of all, I am very grateful to my family, whose support and encouragement have seen me through this project, and to Julia Möhrmann, who started and completed her PhD at about the same time as I did and has been a constant and much valued companion throughout this journey. Thank you for remaining the close friend you have become since we first shared an office.

Michael Wörner
July 2014

Contents

Abstract

The manufacturing sector, as any major part of the economy, is facing new challenges in the increasing pace at which its business environment changes on both the local and the global scale. Globalization increases competition, but it also opens up new markets with customers that may have very different needs and expectations. Emerging markets typically offer inexpensive labour and low production costs whereas the established markets focus on technological sophistication and highly qualified workers. To remain competitive, manufacturers look for ways to increase their efficiency by reducing costs, delays, and throughput times while increasing product quality and other customer benefits such as customization options. Advanced manufacturing engineering designs future factories as complex systems under technological, organizational, and social considerations. These systems exist in an ever-changing environment that dictates raw material prices and customer demands, brings occasional technological advances, and imposes restrictions through laws and regulations. To ensure their long-term survival, they need to be flexible and adaptable. They must be able to react to sudden changes in these environmental factors and to adapt their inner structure to evolve along with the world they exist in.

Information technology plays an important role in reaching these goals. The advent of digital engineering has now introduced various ways of collecting and processing data on the manufacturing process. There is, however, a certain lack in the ability to analyse this data in order to understand and improve the process and to ensure its flexibility and adaptability. Visual analytics is a technique that combines the processing power of automatic data analysis algorithms with the creativity and implicit knowledge of human analysts. Its application to the manufacturing domain can contribute to a successful exploitation of the available data. As an example of how visual analytics may be implemented in future, flexible manufacturing systems, this thesis applies the technique to the task of planning process layouts in a reconfigurable manufacturing system. Process simulation is used as an automatic analysis component and its results are presented visually to aid the human process planner in the search for suitable process layouts. An evolutionary algorithm is used to automatically find and suggest layout variations. This is proposed as an alternative to the traditional process planning and control, which plans for weeks in advance and is not generally meant to take full advantage of the flexibility of a reconfigurable manufacturing system.

The layout planning prototype operates on implicit data, given by the specification of the available process elements, their behaviour, and their interactions. The challenge in handling this data lies not in accessing a large volume of stored data but in evaluating a given point in a virtual data space. Handling an explicit data set is discussed in the context of another system for the classification of machine conditions based on pre-recorded diagnostic data. Issues of increasing response times when handling large volumes of explicit data are then addressed in a system for the exploration of a large real-world data set from the public transport domain. The thesis also includes the proposition of a novel way of navigating through data ordered along one dimension such as time, as analysing data over time is very common in both manufacturing and public transport.

The contribution of this thesis is a classification of visual analytics data space types and analysis tasks, each derived from one of the presented systems. It is shown how this classification can be extended to other works of contemporary visual analytics research.

Zusammenfassung

Wie alle großen Wirtschaftszweige sieht sich die Fertigungsindustrie neuen Herausforderungen gegenüber, die sich aus dem schneller werdenden Wandel des Geschäftsumfeldes im lokalen wie globalen Maßstab ergeben. Globalisierung verstärkt den Wettbewerb, öffnet aber auch neue Märkte mit Kunden, die möglicherweise andere Bedürfnisse und Erwartungen haben. Die Emerging Markets bieten meist geringe Lohn- und Produktionskosten. Die Stärken der etablierten Märkte sind Hochtechnologie und hochqualifizierte Fachkräfte. Um wettbewerbsfähig zu bleiben, suchen Fertigungsunternehmen nach Möglichkeiten, einerseits durch Verringerung von Kosten, Verzögerungen und Durchlaufzeiten ihre Effizienz zu steigern und andererseits die Produktqualität zu verbessern oder Kundenvorteile wie eine kundenindividuelle Variantenfertigung zu schaffen. Das Advanced Manufacturing Engineering entwirft Fabriken der Zukunft unter technologischen, organisatorischen und sozialen Gesichtspunkten als komplexe Systeme. Diese Systeme existieren in einem sich ständig verändernden Umfeld, das Rohstoffpreise und Kundenanforderungen vorgibt, gelegentlich technische Fortschritte liefert und Beschränkungen in Form von Gesetzen und Vorschriften mit sich bringt. Um ihr langfristiges Bestehen zu sichern, müssen die Systeme flexibel und wandlungsfähig sein. Sie müssen auf plötzliche Veränderungen in diesen Umfeldfaktoren reagieren und ihre innere Struktur daran anpassen können, um sich gemeinsam mit ihrem Umfeld fortzuentwickeln.

Die Informationstechnik spielt bei der Erreichung dieser Ziele eine wichtige Rolle. Mit dem Aufkommen des digitalen Engineerings sind zahlreiche Möglichkeiten entstanden, Daten über den Herstellungsprozess zu erfassen und zu verarbeiten. Es besteht jedoch ein gewisses Defizit im Hinblick auf die Fähigkeit, diese Daten zu analysieren, um so den Prozess zu verstehen und zu verbessern und seine Flexibilität und Wandlungsfähigkeit sicherzustellen. Visual Analytics ist eine Methode, die die Verarbeitungsleistung automatischer Datenanalyse-Algorithmen mit der Kreativität und dem impliziten Wissen menschlicher Analysten kombiniert. Sie auf die Fertigungstechnik anzuwenden, kann dazu beitragen, die verfügbaren Daten besser nutzbar zu machen. Als Beispiel dafür, wie Visual Analytics in zukünftigen, flexiblen Fertigungssystemen verwendet werden kann, nutzt diese Arbeit es für die Entwicklung eines Prozessplanungs-Systems für ein rekonfigurierbares Fertigungssystem. Das System enthält eine Prozesssimulation als automatische Analysekomponente und erstellt Visualisierungen der Er-

gebnisse, um den menschlichen Prozessplaner bei der Suche nach einem geeigneten Prozesslayout zu unterstützen. Ein evolutionärer Algorithmus ermöglicht es dem System, Layoutvarianten automatisch zu finden und vorzuschlagen. Dieser Ansatz wird als Alternative zur traditionellen Produktionsplanung und -steuerung vorgestellt, bei der für Wochen im Voraus geplant wird und die Ausnutzung der Flexibilität eines rekonfigurierbaren Fertigungssystems üblicherweise nicht im Vordergrund steht.

Die Layoutplanung arbeitet auf impliziten Daten, die sich aus der Definition der verfügbaren Prozesselemente und deren Verhaltens- und Interaktionsmodellen ergeben. Bei der Handhabung dieser Daten liegt die Herausforderung nicht darin, auf einen großen Datensatz effizient zugreifen zu müssen, sondern in der Auswertung eines gegebenen Datenpunktes in einem virtuellen Datenraum. Die Handhabung eines expliziten Datensatzes wird im Rahmen eines Systems zur Klassifikation von Maschinenzuständen anhand zuvor aufgezeichneter Diagnosedaten behandelt. Mit dem Problem der langen Antwortzeiten bei der Analyse großer, expliziter Datensätze befasst sich schließlich ein System zur Exploration eines sehr umfangreichen Datensatzes aus dem öffentlichen Nahverkehr. Die Arbeit stellt auch eine neue Art der Navigation durch eindimensional geordnete Daten vor. Dies können beispielsweise zeitabhängige Daten sein, wie sie sowohl in der Produktionstechnik als auch im öffentlichen Nahverkehr häufig anfallen.

Der Forschungsbeitrag dieser Arbeit besteht in einer Klassifikation von Datenraum-Typen und Analyse-Aufgaben für Visual Analytics, wobei jede der Klassen aus einem der vorgestellten Systeme abgeleitet wird. Es wird gezeigt, wie diese Klassifikation auf andere aktuelle Arbeiten zum Thema Visual Analytics übertragen werden kann.

1 Introduction

‘Visual analytics’ as a term was first coined in the aftermath of the terrorist attacks on the World Trade Center in 2001. It was later discovered that what made an immediate response or even aversion difficult was not so much a lack of data but rather the limited ability to integrate and filter this data in order to extract relevant information in due time. The general conception was that automation, where it was possible, improved the accuracy and efficiency with which tasks were performed and that this applied to data processing as well. This view was supported by the realization that the data sets that were the subject of these analyses quickly grew beyond what could be handled by a human analyst.

However, real-world cases of large-scale data analysis often show fundamental differences between automatic and manual data processing: In general, automated systems can process a data set much more quickly than any human analyst. They can sum up the daily revenues of a year in an instant or quickly mine through a customer database to divide them into categories. These tasks basically require the application of a set of mathematical rules to a set of numerical values. However, there are operations that are not as easily expressed in mathematical terms: For a human being, determining whether a picture shows a face is a trivial task, which can be completed in an instant just by glancing at the picture. There is no need for any conscious ‘processing’ of the image in terms of carefully scanning it for features that might represent eyes, a nose, or a mouth. An automated system, on the other hand, needs a clear mathematical model of what a face is and how to detect it in a picture. Modern algorithms have become

very good at recognizing faces, but their approach involves considerable computational effort. These deficiencies can be compensated by modern hardware, which can process data quickly enough to come to a result in about the same time it takes a human to take a glance.

Or consider the game of chess. The match between former world champion Garry Kasparov and the chess computer Deep Blue in 1997 gained worldwide media attention and was widely regarded as a match of human against machine. It was also a competition between two very different approaches to the chess problem, with Deep Blue having to evaluate 200 million board configurations per second to reach the same playing strength as Kasparov, who was likely limited to assessing less than five configurations per second. In spite of this dramatic difference in computational power, the match was largely even. Kasparov was able to recognize and think in patterns, to devise and follow long-term strategies, and to benefit from experience and implicit knowledge that cannot be easily expressed in formulas. As a result, his search for the best move was much more directed than that of the machine, which in spite of numerous optimizations was still essentially performing a brute-force graph search. Kasparov's eventual defeat may point to another significant difference between the human and the automatic approach: His spectacular loss in the final game may have to be attributed to the psychological pressure, which made him play far below his usual strength while not affecting his artificial opponent at all.

Visual analytics is an attempt to benefit from both the analytical power of automatic processing and the intelligence, creativity, and judgement of a human analyst by designing systems that combine automatic reasoning support with interactive visual interfaces. Considering the situation in the U.S. after 2001, it is not surprising that Thomas and Cook in their 2005 book *Illuminating the Path – The Research and Development Agenda for Visual Analytics* consider the three major areas for the technique to be analysing terrorist threats, safeguarding borders and ports, and preparing for and responding to emergencies. In the light of large-scale surveillance activities, the first two of these areas may be regarded with some concern, as the ability to analyse private communications data is strongly related to the controversial debate of freedom versus security. Visual analytics as a technique, however, has a wide range of applications, many of which are much less controversial. Consequently, in 2009, Thomas and Kielman published a set of challenges for visual analytics and listed new, promising domains including health, energy, environment, commerce, transport, food, economy, and insurance. This thesis demonstrates the applicability of the visual analytics approach to the manufacturing and transport domains by presenting prototypical analysis systems that address optimization, classification, exploration, and navigation problems in these areas.

1.1 Motivation

Manufacturing, the making of physical goods, has long been an important part of the economy and, naturally, has always been influenced by technological, social, and other changes that determine the conditions under which manufacturing companies operate. One recent development is an increased demand for adaptability, the ability of a factory or an entire enterprise to adjust its inner structure to better conform to the current conditions in terms of resource supply, customer demands, or other external or internal factors. A manufacturing company must engineer not only the products it intends to produce but also the facilities and processes with which to produce them. To ensure adaptability, this ‘manufacturing engineering’ will often have to be extended to a regular ‘reengineering’ of the processes. Reconfigurable manufacturing systems are a way to greatly increase the flexibility of manufacturing processes by reducing the effort required to make physical changes to the process. Leveraging this increased flexibility requires replacing the traditional process planning done weeks in advance with an agile planning that can implement changes and react to new developments on short notice while nevertheless providing enough insight into the solution to ensure the necessary confidence.

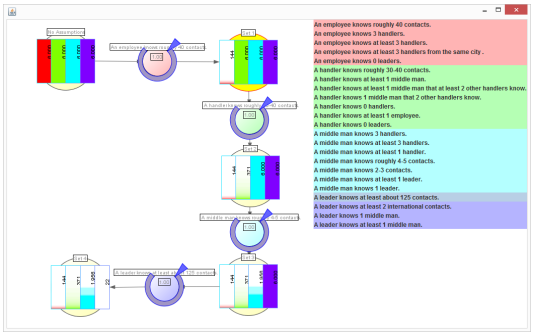
Another important task in manufacturing is assessing the state or quality of items. This can be part of a regular quality control during a manufacturing process or it may be a necessary step in diagnosing a defective product during maintenance. Scheduled maintenance and maintenance upon failure both have disadvantages, which can be alleviated by predictive maintenance, the attempt to analyse machine data to predict the occurrence of problems and fix only what must be fixed while still reliably preventing breakdowns. Automatic approaches exist but are not as widely used as one might expect. A visual analytics approach might supplement automatic methods and increase their appeal and the confidence in the results.

In a conveyor-based reconfigurable manufacturing system, individual work pieces move along predefined tracks following a planned schedule. This can be compared to vehicles travelling in a public transport network, which also follow predefined paths (streets and rails instead of conveyors) according to a central plan (the published schedule). These networks, a vital infrastructure element of many major cities, may operate hundreds of vehicles while logging their activity in on-board units, which results in massive data sets. Statistic evaluation of this data is common, but visual analytics could help to explore the data further and more thoroughly, possibly revealing potential for improving the operation of the network, understanding and preventing undesirable events, and determining the level of congruence between the schedule and the actual operation.

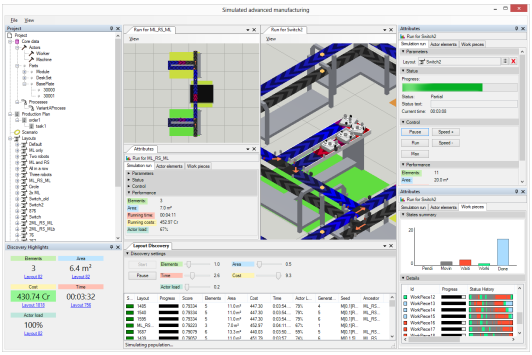
1.2 Structure and contribution

Chapter 2 will summarize the visual analytics approach and describe a practical example of a visual analytics system for solving one of the mini-challenges in the VAST contest 2009 (Bosch et al., 2009). It will also cover some implications of the application of visual analytics to implicit data sets and of using it on optimization and classification problems. Chapters 3 to 5 then present three visual analytics systems designed and implemented as part of this research. These systems address analytical tasks from the manufacturing and public transport domains. The system in Chapter 3 supports an analyst in planning process layouts for reconfigurable manufacturing systems using process simulation for an automatic evaluation of layouts (Wörner and Ertl, 2011b, 2013a). For this thesis, it has been enhanced with an evolutionary algorithm for the automatic generation of layout variants. Chapter 4 describes a system that supports an analyst in creating a classifier for the evaluation of machine data to identify defective machines and detect borderline cases (Wörner et al., 2013). In Chapter 5, a massive public transport data set is explored using a framework that ensures that the interactivity of the system is retained in spite of long-running automatic analysis tasks (Wörner and Ertl, 2012, 2014). Chapter 6 presents the *SmoothScroll* control (Wörner and Ertl, 2011a, 2013b), a multi-scale, multi-layer slider that can be used in a visual analytics system to facilitate the navigation through one-dimensional data sets. This includes, but is not limited to, time-oriented and text data. Chapter 7 summarizes and discusses the proposed systems and generalizes the results to different analysis tasks, which constitutes the primary contribution of this thesis. Chapter 8 gives an outlook on possible future directions of research.

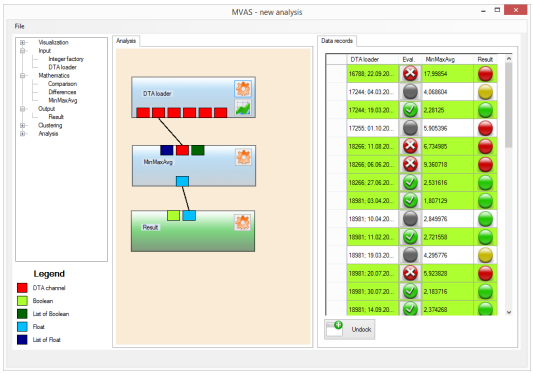
Beyond the scope of this thesis, work has been published on the semantic annotation of documents using a distortion technique to provide context around annotated words while visually compressing the text in between (Giereth et al., 2008). A hex editor with semantic lenses provided visual support for editing raw data inaccessible to high-level tools due to data corruption or an unknown data format (Wörner et al., 2010). Aspects of visualization in the smart real-time factory were discussed in a collaboration with Hameed et al. (2011). A density map visualization for geolocated microblog messages was contributed to a VAST contest 2011 entry (Bosch et al., 2011), which later evolved into the *ScatterBlogs* system for the detection of spatiotemporal anomalies in Twitter messages (Thom et al., 2012). The subsequent *ScatterBlogs2* system (Bosch et al., 2013) included the *SmoothScroll* control presented in Chapter 6, as did the *TrajectoryLenses* approach for the interactive exploration of movement trajectories (Krüger et al., 2013).



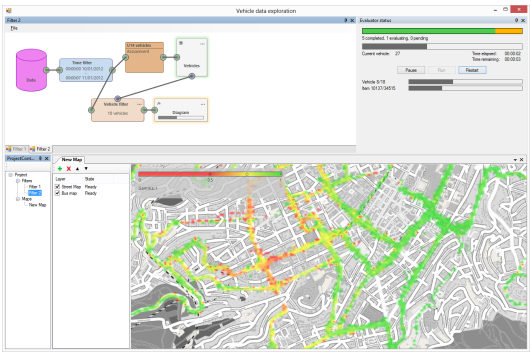
Chapter 2



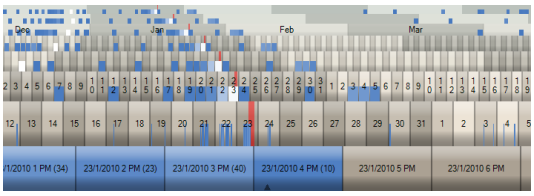
Chapter 3



Chapter 4



Chapter 5



Chapter 6

Figure 1.1: A visual overview of the systems and topics covered in this thesis: Social network analysis (Chapter 2), manufacturing layout planning (Chapter 3), machine diagnosis (Chapter 4), vehicle data exploration (Chapter 5), and linear data navigation (Chapter 6).

2

Visual Analytics

In their roadmap publication *Illuminating the Path – The Research and Development Agenda for Visual Analytics*, Thomas and Cook (2005) define visual analytics as 'the science of analytical reasoning facilitated by interactive visual interfaces'. Reasoning, in general, is the acquisition of new knowledge from existing knowledge by applying rational techniques such as deduction or induction. Analytical reasoning is a systematic approach, which—in its literal sense—breaks a problem into smaller parts and solves each in turn to arrive at an overall solution. In contrast to intuitive reasoning, which regards and solves a problem as a whole, analytical reasoning often involves applying a series of simple, clearly defined steps, which makes its conclusions reproducible and verifiable. It also opens up the possibility for a (possibly partial) formalization and automatization of the process. Consequently, a later definition by Keim et al. (2010) states that 'Visual analytics combines automated analysis with interactive visualisations for an effective understanding, reasoning and decision making on the basis of very large and complex datasets' and thus explicitly in-

Parts of this chapter have previously been published in:

Bosch, H., Heinrich, J., Müller, C., Höferlin, B., Reina, G., Höferlin, M., Wörner, M., and Koch, S. (2009), 'Innovative Filtering Techniques and Customized Analytics Tools', in *Proceedings of the 2009 IEEE Symposium on Visual Analytics Science and Technology – VAST 2009* (IEEE), 269–270 ©2009 IEEE;

Wörner, M. and Ertl, T. (2013a), 'Simulation-based Visual Layout Planning in Advanced Manufacturing', in *Proceedings of the 46th Hawaii International Conference on System Sciences – HICSS 2013* (IEEE), 1532–1541 ©2013 IEEE.

cludes automatic analysis as a central building block of the visual analytics approach.

For both automatic data analysis and interactive visualizations, established research communities existed well before 2005. Visual analytics is not intended to replace or reinvent any of these areas but to combine elements from these existing scientific tool kits into an integrated system that is more powerful than any of its parts. The potential of this combination lies in the integration of automatic and human reasoning. Visualization is an effective way of conveying information to a human and an interactive visualization can provide the means to control and steer algorithms that excel in speed, precision, and endurance but lack judgement and inspiration. In fact, the approach can be beneficial even for problems that can be solved without human intervention as an interactive analysis supported by visualization can provide insight into the reasoning that lead to the proposed result. For complex problems, this insight might reveal flaws in an automatic solution that—even though it fulfils all formal requirements—is unacceptable for reasons difficult to express in formal terms. Also, a purely automatic approach may be of limited use when the situation requires someone to take responsibility for the consequences of a decision. Algorithms and computers cannot be held responsible for any adverse effects of their results and human decision-makers may be unwilling to rely on or advocate them unless they can understand and agree with their reasoning. Visual analytics may provide a way to benefit from powerful but inscrutable automatic methods while retaining clear accountability.

2.1 Automatic data analysis

The automatic data analysis part of a visual analytics system supports a user's analytical reasoning through automatic computation. This does not necessarily imply automatic reasoning, although this is certainly possible to some extent if a formal expression of the necessary axioms, definitions, and theorems can be found (for example, Benzmüller and Woltzenlogel Paleo (2013) achieved an automatic verification of Kurt Gödel's rather complex ontological proof of the existence of God, the implications of which are, of course, philosophical rather than theological). A visual analytics system may leave any actual reasoning to the user and instead use automatic data analysis to support this reasoning by providing some form of insight beyond what can be seen from the original data. This will always involve some form of transforming the data into a different representation. Numerous methods may be employed and their usefulness will vary greatly with the problem at hand. They include techniques from the areas of machine

learning (Mitchell, 1997) and data mining (Han et al., 2006; Hand et al., 2001). Machine learning is a class of techniques that predict properties of a data item based on known data items with known properties. This includes decision trees, artificial neural networks, and various clustering techniques. Data mining algorithms find previously unknown patterns in data. However, even seemingly simple operations such as the computation of statistical indicators or a sophisticated visual arrangement of data items may provide valuable analytical insight to the user.

Techniques used in the systems presented in the subsequent chapters include simulation and optimization using an evolutionary algorithm (Chapter 3), clustering and linear separation (Chapter 4), and filtering, aggregation, and diagram generation (Chapter 5).

2.2 Visualization

Visualization transforms data into images, presenting a concrete representation of the abstract. Strictly speaking, even a numerical listing of data values is a kind of visualization, but the term usually refers to a representation in shapes and colours such as diagrams, maps, or pictures. Visualization is ubiquitous in media, science, and teaching as the human brain's considerable visual capabilities extend far beyond a mere perception of light. They include the ability to recognize patterns, objects, and faces and to perceive whole structures and configurations instead of only a collection of elements (properties studied by Gestalt psychology). As a result, visualizing a data set can often provide insights that would have been difficult or impossible to obtain by perusing the raw data.

2.3 Visual analytics

Visual analytics combines automatic data analysis and interactive visualization into a system that supports a user's analytical reasoning. It assists the user in performing an analysis task on a data set. Automatic analysis can extract relevant information from the data and visualization presents this information in a way that supports the user in understanding it and possibly in gaining insights beyond the results of the automatic analysis. The interactivity of the visualization provides possibilities for manual adjustments by focussing on certain aspects or changing the spatial arrangement.

Working with large data sets is often facilitated by the focus + context and overview + detail principles, sometimes in combination with a distorted display. The overview + detail approach (Spence and Apperley, 1982;

Robertson and Mackinlay, 1993) presents multiple visualizations of the same data at different levels of detail. By providing both high-detail information about the focus point and low-detail information about the data surrounding it, it can help to answer questions such as 'What am I looking at?', 'Where in the data am I?' and 'What else is there?' all at once. One common example is a calendar application that shows a larger day view and a smaller month view with an indicator highlighting the current day. The scale of the overview is often a trade-off between being too detailed, when the overview does not display useful context information, and being too coarse, when the overview is no longer useful for navigating the data.

The focus + context principle is similar to overview + detail but combines low- and high-detail information in a single view. The reduction in detail may be achieved by scaling down the visual representation of the data items, by aggregating data items to larger entities, or by a combination of both. The transition between the different levels of detail may be either abrupt or gradual. A gradual transition in detail can be achieved by distorting the view in a way that allocates a larger portion of the display to the focused data items and a smaller portion to those not in focus (Leung and Apperley, 1994). The classic approaches include the Fisheye view (Furnas, 1986), which has been applied to one-dimensional (Bederson, 2000) and two-dimensional (Sarkar and Brown, 1992) data alike. Using distortion, it is possible for all the data to remain visible without occlusion. Another advantage of this approach is that in comparison to the overview + detail concept, it removes the need to relate information in different views. On the other hand, the distorted space makes it very difficult to put item positions, distances, and sizes in relation to each other.

Visual analytics is usually not a one-step process. The true power of the combination of human and automatic analysis lies in the possibility of using the insights gained from the visualization to further improve the analysis. With the interactive interface of a visual analytics system, a user can not only adjust the visualization but also influence the analysis to incorporate new insights. This feedback creates an analysis loop or cycle that repeats automatic analysis and human intervention until the results are satisfactory.

Figure 2.1 shows the visual analytics cycle as presented by Keim et al. (2010). The overall process is one of transforming data into knowledge (or insight). 'Data', in this sense, is a pure accumulation of values whereas 'knowledge' is useful information that provides some kind of benefit, such as being able to take informed and advantageous action. In Keim's model, visual analytics supports this transformation via models and visualization. A 'model' can be understood as a formal interpretation of data derived by data mining. It can be presented to the user via model visualization.

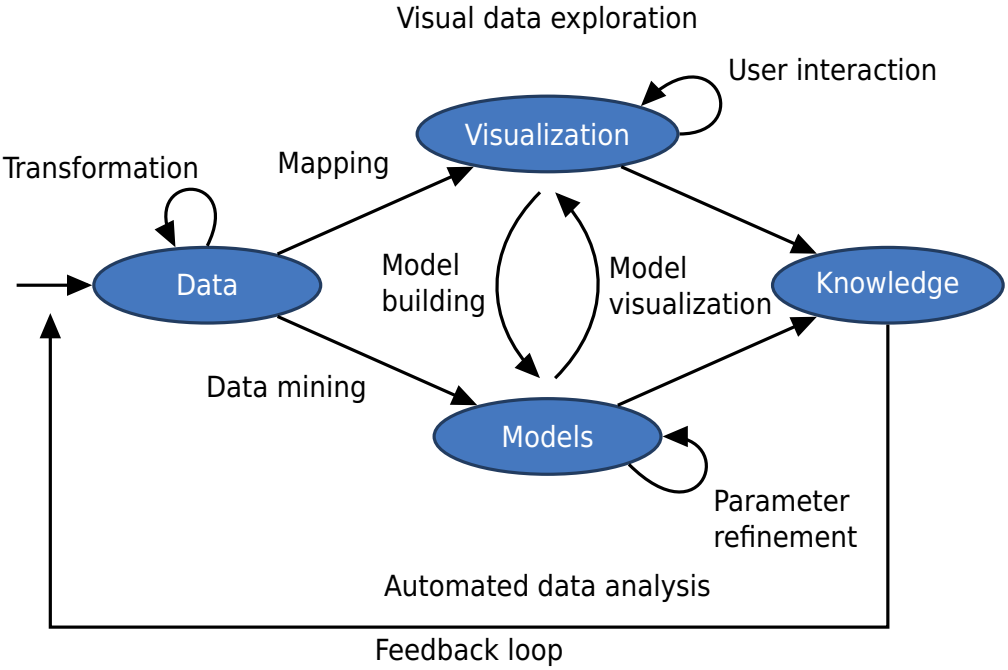


Figure 2.1: The visual analytics cycle according to Keim et al. (2010)

A visualization can also be generated directly from the data, without a previous data mining step ('mapping'). The user can interact with the visualization to build models manually. The combination of visualization and models produces knowledge, which may enable the user to provide feedback to the start of the process.

It is apparent that the visual analytics approach is not limited to the original applications given by Thomas and Cook: analysing terrorist threats, safeguarding borders and ports, and preparing for and responding to emergencies. In addition to security-related topics, Thomas and Kielman (2009) list health, energy, environment, commerce, transportation, food/agriculture, economy, insurance, knowledge work, and individual or personal use as domains in which visual analytics can make important contributions. The domains addressed in this thesis are manufacturing (Chapters 3 and 4) and public transport (Chapter 5).

2.4 Solving the VAST contest 2009

Visual analytics is a young scientific discipline. To stimulate the advancement of visual analytics research, the IEEE VAST conference ('Visual Analytics Science and Technology', formerly a symposium) has held annual contests since 2006. These contests typically provide a data set along with

a number of analysis questions and invite academic and corporate researchers to submit their solutions along with a detailed description of the tools that were used and the analysis steps that were taken to arrive at the final conclusions. A jury judges the submissions not just by the quality of the answers but also—and possibly more importantly—by how visual analytics techniques were employed to obtain them. For their analysis, contestants are free to use available software or create their own. VAST contest data sets included social network data, genetic sequences, and microblog messages. With a group of my fellow researchers, I participated in the 2009 VAST contest. To give a practical example for the visual analytics approach, this section describes both the contest scenario and our solution (Bosch et al., 2009).

2.4.1 The contest scenario

The 2009 contest provided a fictional scenario involving a spy in an embassy. The contest tasks were divided into three so-called mini challenges, each with a different data set. Challenge 1 was concerned with badge and network traffic, challenge 2 with a social network, and challenge 3 with video data. In collaboration with Harald Bosch and Steffen Koch, I contributed a solution to the second challenge.

Mini challenge 2 provided a fictional social network called ‘Flitter’ that consisted of 6,000 named accounts and about 30,000 links. These accounts included a number of persons to be identified, and the challenge description gave textual descriptions of two possibilities of how these persons were organized within the network:

Configuration A: ‘The employee has about 40 Flitter contacts. Three of these contacts are his “handlers”, people in the criminal organization assigned to obtain his cooperation. Each of the handlers probably has between 30 to 40 Flitter contacts and share a common middle man in the organization, who we have code-named Boris. Boris maintains contact with the handlers, but does not allow them to communicate among themselves using Flitter. Boris communicates with one or two others in the organization and no one else. One of these contacts is his likely boss, who we’ve code-named Fearless Leader. Fearless Leader probably has a broad Flitter network (well over 100 links), including international contacts.’

Configuration B: ‘The employee has about 40 Flitter contacts. Three of these contacts are his “handlers”, people in the organization assigned to obtain his cooperation. Each of the handlers likely has between 30 to 40 Flitter contacts, and each probably has his or her own middle man in the organization, who we’ve code-named Boris, Morris and Horace. It is probable the middle men will not allow the handlers to communicate among

themselves using Flitter. Each of the middle men probably communicate with one or two others in the organization, and no one else. One of the contacts for all of the middle men is the head of the organization, Fearless Leader. Fearless Leader has a broad Flitter network (well over 100 links) including international contacts.'

Both descriptions are very similar. The only factual difference is that configuration A uses a single 'middle man' whereas configuration B has three. A more subtle difference is that configuration B classifies most assumptions on the middle men as 'probable' whereas configuration A states them as facts. Conversely, configuration A says that the head of the organization 'probably' has a broad network, whereas in configuration B, this fact is certain. Both descriptions contain two different kinds of uncertainty. One is that often only an approximate number of contacts is known ('about 40 contacts', 'well over 100 links'), the other that some of the statements themselves are uncertain ('probably 30 to 40 contacts'). Configuration B ('Fearless Leader has a broad Flitter network') essentially rules out the possibility for an account with very few contacts to be the leader. Configuration A ('Fearless Leader probably has a broad Flitter network') does not as the statement as a whole is uncertain.

The analytical task to be completed for mini challenge 2 was to determine which of the configurations was actually present in the data and to identify the respective accounts as well as their roles in the organization (employee, handler, middle man, or leader).

2.4.2 Automatic analysis

Drawing a network of 6,000 nodes and scanning it for one of the described structures is a hopeless undertaking. Quite clearly, a system that can be used for solving this challenge requires some form of automatic support. The approach I chose was breaking down the configuration descriptions into a number of formal rules and implementing an algorithm to determine which accounts in the network comply with a given rule. These rules take the form

A *role1* knows [at least] *range* [international] *role2*.

role1 and *role2* are two account roles, that is, one of 'employee', 'handler', 'middle man', 'leader', or the special role 'contact', which includes all accounts. *range* is a numeric value range such as '4-5' or 'about 40'. Adding 'at least' includes accounts that have more than the specified number of acquaintances and adding 'international' requires the acquaintances to live in a different country than the account in question. This form can be used to express many of the configuration statements:

An *employee* knows *roughly 40 contacts*.
An *employee* knows *3 handlers*.
A *handler* knows *30–40 contacts*.
A *handler* knows *1 middle man*.
A *middle man* knows *1 leader*.
A *leader* knows at least *100 contacts*.
A *leader* knows at least *1* international *contact*.

These rules apply to both configurations. The statement that the middle men communicate ‘with one or two others in the organization, and no one else’ translates to

A *middle man* knows *4-5 contacts*.

for configuration A (where all handlers share a middle man) and to

A *middle man* knows *2-3 contacts*.

for configuration B (where all handler have a middle man of their own). This set of rules is not a complete reproduction of the textual descriptions (for example, it lacks the restrictions that the handlers must not know each other). Nevertheless, it can be used to narrow down the set of possible solutions. For example, all accounts that have less than 100 contacts or no international contacts cannot be the leader. This leads to a process of elimination. Starting from the initial assumption that every account can take every role, one can quickly identify accounts that cannot take some of the roles due to the number of their contacts. In subsequent steps, one can also exclude accounts based on the roles their remaining contacts can take. For example, an account that is not in contact with any possible leader cannot be a middle man.

A process of elimination deals with possibilities. One can only be certain that an account plays a given role when all other possibilities have been eliminated. However, with each elimination step, certainty grows on what roles an account *cannot* play. For this strategy to be successful, the rules must be adjusted so that they only ever restrict the minimum number of acquaintances in a role, never the maximum. With the initial assumption of 6,000 possible candidates for every role, a rule like ‘A *middle man* knows *1 leader*.’ would instantly eliminate all middle man candidates. Requiring middle men to know ‘at least *1 leader*’ eliminates only those accounts that have no contact to any of the remaining leader candidates.

Since every change in an account’s possible roles might influence other accounts, this evaluation must be repeated in an iterative process until a stable configuration is reached. One remaining question is how to handle

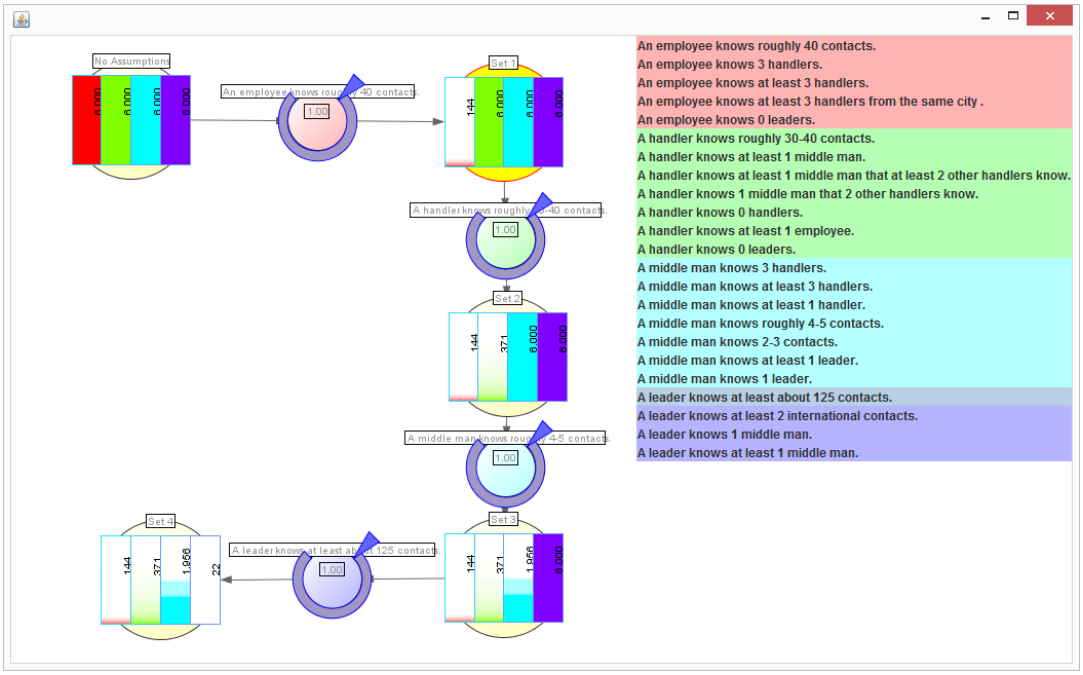


Figure 2.2: The filter graph view.

the uncertainties in the configuration descriptions. Instead of a Boolean classification in ‘yes’ and ‘no’, the rules assign a certainty value to an account/role combination. An account that exactly matches a criterion, such as one whose number of contacts is within the range specified in the rule, receives a certainty of 1.0. If the number is well outside the range, the certainty is 0.0. Between these extremes, there is a gradual falloff and the combination of multiple certainty values from multiple rules gives a differentiated view on whether an account meets the requirements of a role.

2.4.3 Visual interface

We implemented a visual analytics system that can be used to solve mini challenge 2. Using its interactive visualization, a user can define and apply rules while the automatic rule evaluation computes the resulting candidate sets. Based on these results, the user may decide to apply additional rules or take back rules. Once the possibilities have been narrowed down sufficiently, a visualization of the remaining network may provide additional insights. The visual interface provides multiple views on the data, as is common in visual analytics systems.

The central view is the **filter graph view** (Figure 2.2), which displays candidate sets as groups of four coloured bars. Rules can be dragged into

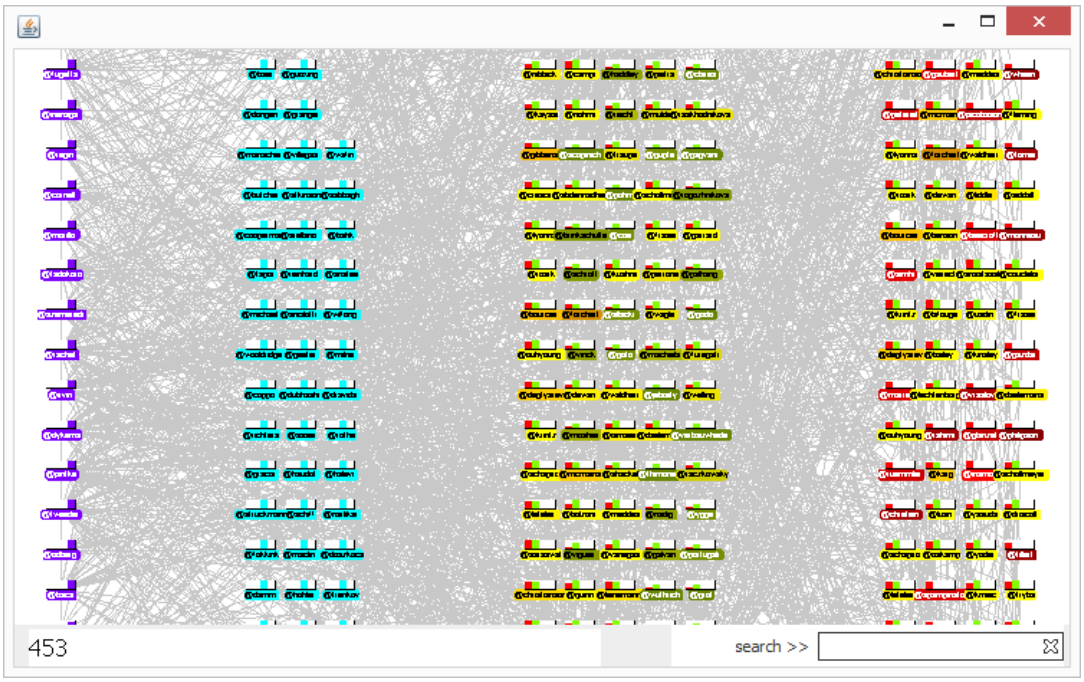


Figure 2.3: The network view shows all remaining candidates for (from left to right) leaders, middle men, handlers, and employees.

the graph from the list of rules to the right. This creates a new group of bars which reflects the effects of the rule. The colours of the bar correspond to the four roles ‘employee’ (red), ‘handler’ (green), ‘middle man’ (blue), and ‘leader’ (violet). A full bar means all accounts are possible candidates for this role. Applying rules will often eliminate some possibilities and reduce the height of the corresponding bar. Small numeric labels on the bars list the number of remaining candidates.

The **network view** (Figure 2.3) is a node-link diagram that displays the remaining candidates for each role. This will generally not be useful until the candidate sets have been greatly reduced by the application of rules. For smaller networks, however, it provides the possibility for a visual inspection and verification of possible solutions. Some aspects of the configuration descriptions cannot be expressed using the automatic rules. If the user determines that an account cannot take the role for which it is listed in the display, this external knowledge can be introduced into the system using a right-click menu. This sets the account’s confidence for this role to zero, which may have cascading effects on the role confidences of other accounts.

2.4.4 The analysis

To solve the mini challenge task, a user drags rules into the filter graph to reproduce the configuration descriptions. Applying only rules that are true for both configurations, one can reduce the sets to 116 possible employees, 298 possible handlers, 2,285 possible middle men, and 22 possible leaders. At this point, the user can split the graph to examine the two hypotheses in parallel. For configuration A, a restriction can be added that a middle man must have 4–5 contacts, 3 of which have to be handlers. For configuration B, a middle man knows 2–3 contacts including at least 1 handler. This reduces the number of candidates to 2/6/2/2 for configuration A and 0/0/0/0 for configuration B. There is no group of accounts in the data that meets all the requirements in the description of configuration B, which partly answers the first analysis question. To determine whether there is a group in the form described by configuration A, the user can open the network view on the remaining 12 accounts (Figure 2.4) and examine the situation. The accounts form two possible networks, both of which satisfy all automatic rules. In this visualization, however, one can see that @bailey and @letelier know each other, something explicitly forbidden in the configuration description.

There is no automatic rule that could tell the system that @bailey and @letelier cannot be handlers in the same network. However, the user can provide that information by right clicking @bailey and choosing ‘is not a handler’. With the automatic rules still in effect and only two possible handlers remaining in that network, this eliminates the entire network and leaves the other solution as the only group of accounts that satisfies all restrictions in either of the configuration descriptions.

This practical example highlights some typical features of a visual analytics system: There is an automatic component that supports the analysis process by computing the effects of including or excluding simple logical rules derived from the natural language descriptions in the analysis question. There also is a visualization that displays the sequence of analysis steps and their results (in the form of coloured bars) and the candidate networks of solutions. The visualization is interactive: The user can pan and zoom the views to adjust the amount and scale of information that is visible. The filter graph view can be used to construct a hypothesis on which rules are actually reflected in the data. The automatic component works in the background to evaluate these rules and compute the resulting candidate sets. The visualization of these sets can help to verify or disprove a hypothesis. In the network view, the user can manually change the evaluation of individual data items, introducing external knowledge into the system that is not reflected in any of the rules but can nevertheless

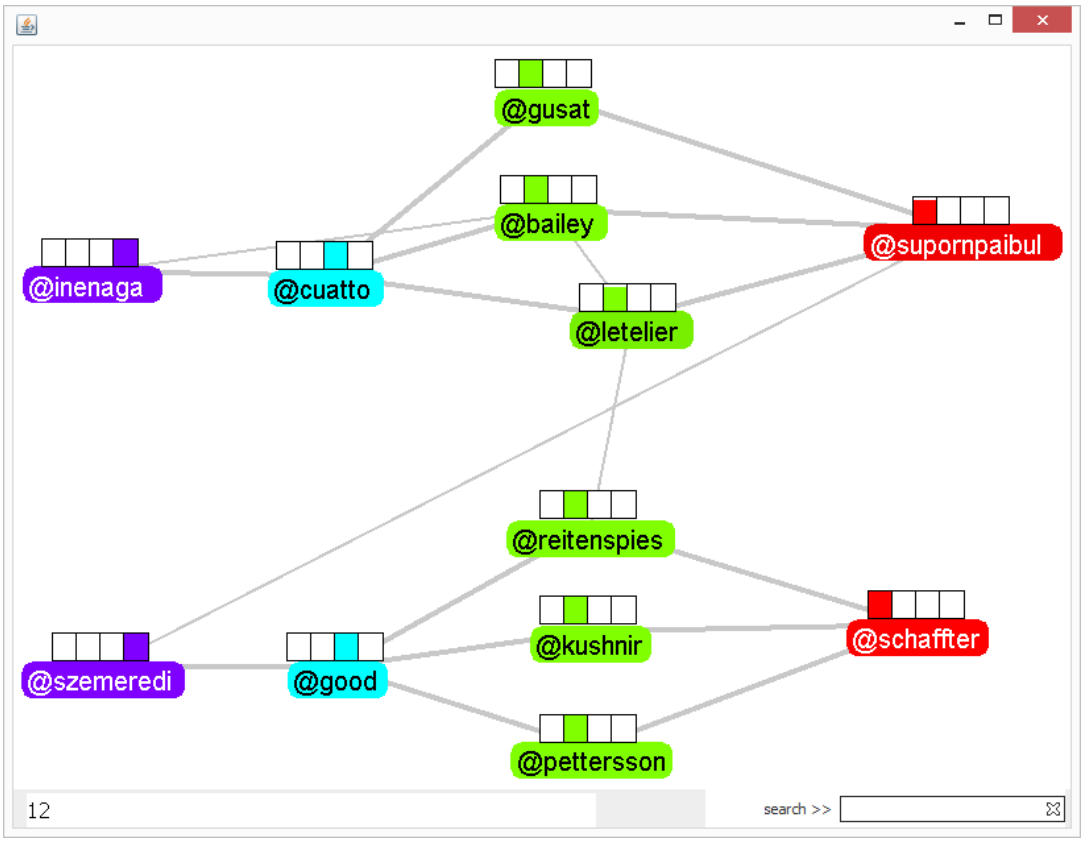


Figure 2.4: The network view showing the remaining candidates. The network at the top cannot be a solution, because two of the handlers know each other.

be used by the automatic evaluation to compute the consequences and effects on other items. This forms a cycle, in which the automatic analysis provides information to the user but the user also has the possibility to feed information back to the automatic analysis.

2.5 Visual analytics on implicit data spaces

Visual analytics is commonly applied to data sets that are too large to be analysed manually. Typically, these data sets are explicitly defined in the form of large collections of data items. However, this is not always the case. Chapter 3 describes a system for planning manufacturing process layouts. A process planner can use the system to find a suitable process layout from the set of all possible layouts and production plans. This data space is not defined in terms of a complete enumeration of all elements but by a model that describes these elements in abstract terms. By listing the available

layout elements and their connectivity options and characterizing their behaviour, this model provides an implicit definition of the data space. In both implicit and explicit data spaces, a point in the space can be identified by a set of parameters or coordinates (such as an index value). In an explicit data space, this point can then be obtained by looking it up, that is, finding the corresponding data item in the data set. In an implicit data space, the point is obtained by evaluating the data space model for the given parameter set. Generally speaking, an implicit data space definition tends to require less memory or disk space, but the evaluation of a single point is more complex.

An analysis process will usually involve looking at or otherwise processing certain properties or attributes of a data point. For example, in an explicit data space of vehicle data (such as the one used in Chapter 5), one may be interested in the vehicle speed value associated with a given parameter set (such as a vehicle identification and a point in time). If this attribute is part of the data space, it can simply be read from the data source once the point itself has been located. In the layout planning case, one may be interested in the total time a layout takes to process a given manufacturing task. However, since neither the layout nor its attributes are listed in a data source, this value has to be obtained by other means, such as by computing a simulation of the layout to estimate the required time.

Visual analytics in implicit spaces has been explored in *HyperMoVal* (Piringer et al., 2010), a system for the validation of regression models that supports both comparing actual results with a model predictions and comparing different model predictions with each other. These models provide implicit definitions of a continuous, high-dimensional parameter space. *HyperMoVal* supports analysing this space by sampling it at distinct points to create plots of pairs of dimensions. In an extension to this work, Berger et al. (2011) added uncertainty visualization and the option to choose between two different sampling strategies. *Vismon* (Booshehrian et al., 2012) is a tool for analysing simulation results and is optimized for an application that uses only two input dimensions and a limited number of output dimensions. Unlike *HyperMoVal*, it works on a static data file, but this data file, too, has been created by sampling an implicit space, in this case using a simulation model that calculates various indicators for different combinations of the two input parameters.

Chapter 3 presents *Sam* (‘Simulated advanced manufacturing’), which extends this approach of exploring an implicit data space to reconfigurable manufacturing processes. Many factors contribute to the evaluation of a process layout, and as process planners may not always be able to quantify exactly what properties they are looking for, a layout’s true merit

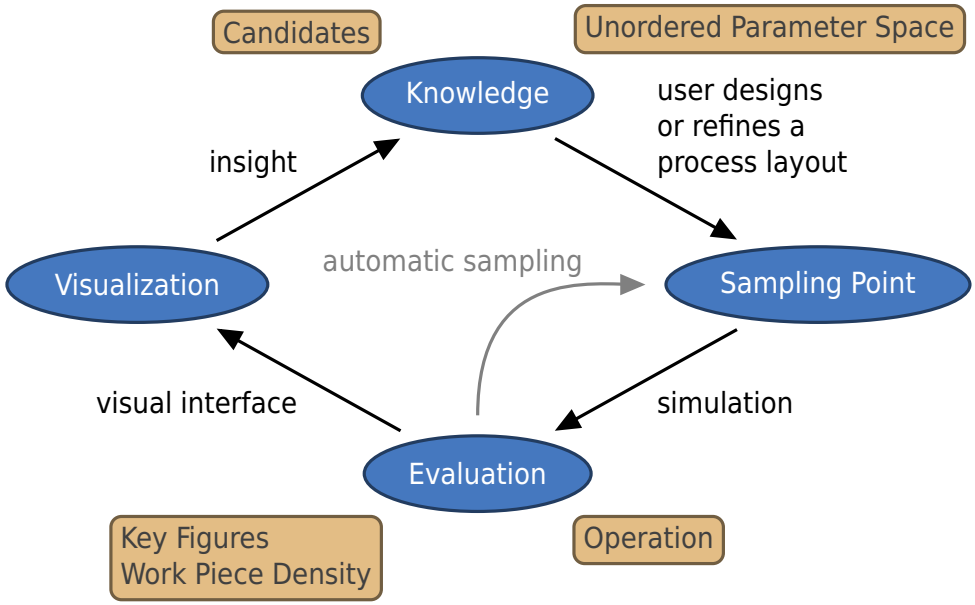


Figure 2.5: A visual analytics cycle on an implicit data set

can probably only be assessed by a human expert. Planners will usually want to minimize the production costs, but there may be cases where a short production time is more important. Also, a layout that requires less space or elements may be preferable if this only slightly increases cost and time. And maybe the planner discards a cost-effective layout because it leaves a worker idle for too long. Automatic systems can support this complex assessment, for example by calculating key figures to characterize some properties of a layout. Like *Vismon*, *Sam* uses a simulation model to evaluate a given layout. One important difference, however, is that its input space is inherently discrete. Ways can be found to enumerate process layouts and production plans (an example is given in Section 3.8.2), but there is no intuitive way of representing them as points in a continuous space. *Sam* relies on its users to select promising sample points by interactively designing layouts, but it also offers automatic support by automatically sampling and evaluating variants of these layouts.

The resulting visual analytics cycle (Figure 2.5) can be seen as a variant of the one presented by Keim et al. (Figure 2.1). Starting with some initial knowledge about the problem domain, an analyst selects a sampling point in the unordered parameter space—in this case by designing an initial process layout. This layout is then passed to the automatic analysis component, which evaluates the point. In *Sam*, this evaluation is done by simulating the corresponding layout and results in a number of key figures, a visual representation of the work piece density, and the possibility of

understanding the operation of a layout by watching it in action while it is being simulated. The evaluation is presented in an interactive visual interface that allows comparing the evaluations of different layouts and thus illustrates the effects and consequences of any choices made. By examining the visualization of a layout and the visual comparison of different layouts, the analyst can gain insight into the advantages and disadvantages of each layout and determine a set of promising layout candidates. This new knowledge can then be used to adjust one of the current candidates or create a new variant, completing the visual analytics cycle until a satisfactory configuration has been found.

As an optional extension to this process, *Sam* offers an automatic sampling component, which constructs and evaluates new layout variants based on the layouts seen so far and their automatic evaluation. If the evaluation of one of these variants improves upon the known layouts in some aspect, the user is notified and can examine a visualization of this layout variant and its evaluation.

2.6 Visual analytics on optimization problems

Visual analytics often deals with exploratory data analysis. Analysts explore visualizations of a data set to get a 'feel' for the data, form hypotheses, and then verify of falsify these hypotheses by further analysis. While the layout planning process as discussed in Chapter 3 follows this approach, it is also an optimization problem.

In a mathematical sense, an optimization problem is given by (Boyd and Vandenberghe, 2004):

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f_0(x) \\ \text{subject to} \quad & f_i(x) \leq 0, \quad i = 1, \ldots, m. \end{aligned}$$

Here, x is a vector of parameters and f_0 is an evaluation function. The goal of the optimization is to find a set of parameters x' for which f_0 takes the lowest possible value: $\forall x : f_0(x') \leq f_0(x)$. Usually, there also are a number of constraints given by f_1 to f_m and any set of parameters is considered a valid solution only if it satisfies these constraints. Typically, the parameter space is continuous, such as $x \in \mathbf{R}^n$ and $f_i : \mathbf{R}^n \to \mathbf{R}$ ($i = 0, \ldots, m$). In combinatorial optimization problems (Schrijver, 2003), however, x is from a finite set of discrete objects. In *Sam*'s layout planning problem, the space of possible process planning parameters is discrete, and although it is infinite in theory, practical constraints limiting the maximum number of process elements or the area to be covered by the process layout can only be satisfied by a finite number of elements.

Optimization problems tend to be complex. One well-known example is the travelling salesman problem, which, given a set of cities and distances, asks for the shortest route that visits every city exactly once. For this problem, the evaluation function f_0 is simply the total length of a given route and can easily be calculated. The set I contains all possible routes and is finite for a given number of cities. The problem can therefore be solved by enumerating all routes in I and finding a route x' with minimal $f_0(x')$. However, with n cities to visit, there are $(n-1)!$ routes to consider (or half of that number if not counting the reversal of a route). Consequently, even for the relatively small number of 20 cities, one would have to evaluate $6 \cdot 10^{16}$ routes. More efficient algorithms exist, but the problem has been shown to be **NP**-hard, meaning that under the general assumption that $\mathbf{P} \neq \mathbf{NP}$, there most likely is no algorithm that can compute an exact solution in less than exponential time.

Many optimization problems can be solved much more easily if one relaxes the requirement of finding an optimal solution. For example, there is an $\mathcal{O}(n^3)$ algorithm which finds a travelling salesman route with a length that is guaranteed to be within 150% of that of the optimal solution (Christofides, 1976). This is a heuristic approach, which trades the guaranteed optimality of the solution for time. There are less specialized heuristics that require little more than the definition of the evaluation function f_0. Evolutionary algorithms (Bäck and Schwefel, 1993) create a 'population' of solution candidates, mutate and possibly recombine them using supplied operators, then make a selection of which solutions to keep for the next generation based on their 'fitness' according to the evaluation function. This technique can be adapted to a wide range of optimization problems simply by providing an evaluation function and appropriate mutation and recombination operators.

All automatic approaches for solving optimization problems depend on a clearly defined evaluation function f_0. With many practical problems, there is a general agreement that some solutions are better than others, but this notion cannot easily be expressed in mathematical terms. As an example, scheduling a number of events for a group of people can be a complex task, not only because the number of possible solutions may be larger than expected (when scheduling 8 one-hour events over 5 days with one-hour slots from 9 a.m. to 5 p.m., one can choose from 8.691.104.822.400 possible schedules), but also because in the real world, people tend to not specify their availability strictly in terms of *yes* and *no*. Instead, one might hear 'I'd prefer to not have any long breaks between my events' or 'I'm at work until 3 o'clock, but if there's no other way, I can take a holiday for the week, then I'm available all day' or 'I'd like to finish as early in the day as possible'. When no solution can satisfy all these constraints, an evaluation function

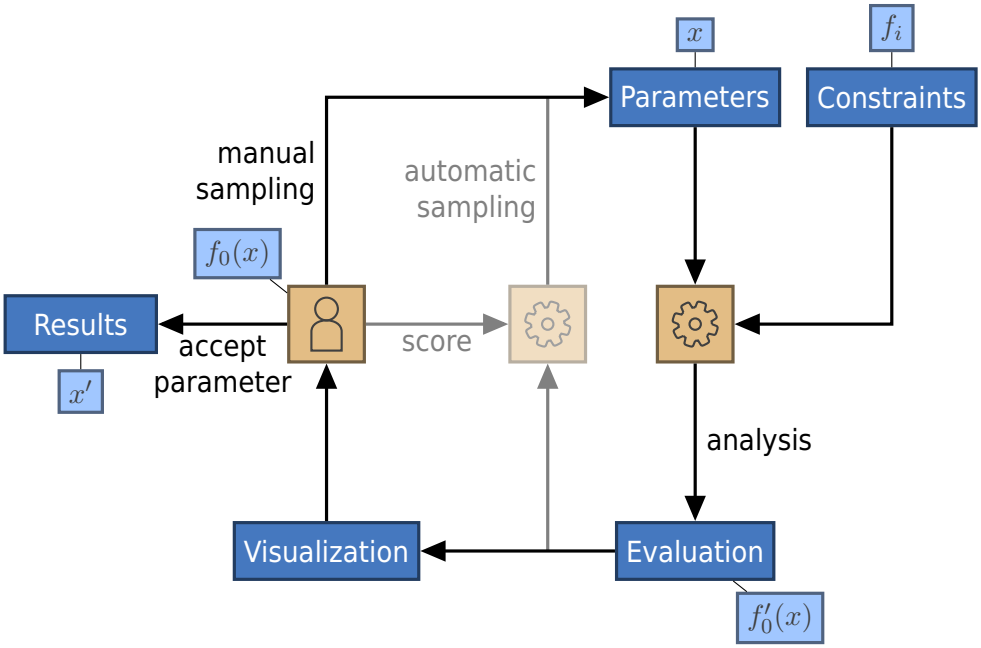


Figure 2.6: A visual analytics approach to optimization problems. A human expert evaluates a parameter x based on a preliminary evaluation $f_0'(x)$ generated by automatic analysis. This may prompt another cycle for a different parameter (manual sampling), which may optionally be complemented by automatic sampling based on the automatic analysis result or a user-provided score value.

for a given schedule will also have to prioritize and weigh the individual requirements against each other.

In these cases, the actual definition of the evaluation function f_0 is often a matter of careful balancing by a person. In the extreme, the mathematical evaluation function may be replaced entirely by the assessment of a human expert. Continuing this train of thought, one may arrive at a system that uses the visual analytics approach in an implicit data space (the domain of f_0) and combines automatic and human sampling and evaluation to solve a complex optimization problem. This approach is illustrated in Figure 2.6: Given a parameter x, an automatic analysis component checks the constraints f_i and provides some form of initial evaluation $f_0'(x)$ of this parameter. This evaluation is visualized and presented to the user of the system. The user takes the role of the evaluation function $f_0(x)$ and either accepts this parameter as a solution x' or uses the knowledge gained by evaluating x to select a new parameter to be evaluated. As Chapter 3 shows, this search may be supported by automatic sampling.

2.7 Visual analytics on classification problems

Classification associates objects with classes. A classifier function $f(x)$ maps an object x to a set of classes the object belongs to. As an example, Chapter 4 describes a system that classifies machine data records into ones indicating 'working' machines and ones indicating 'defective' machines. Classification is a two-step process: First, the classifier function has to be defined, then it can be used to classify objects. The first step of finding a suitable classifier function is often much more complex than the second step of evaluating this function for a number of objects. A classification problem can be defined as given a set of objects and classifications $\{(x_1, c_1), (x_2, c_2), \ldots, (x_n, c_n)\}$, find a classifier function $f(x)$ so that $f(x_i) = c_i$ for $i = 1 \ldots n$. Classification problems are often solved using machine learning, in which algorithms are 'trained' to learn a classification. A trivial solution involves a look-up table that associates all objects x_i with their corresponding class c_i and is undefined for all other objects. This solution produces perfect results for the known set of $i = 1 \ldots n$ but is generally useless for classifying as of yet unknown objects. To verify that a classifier function produces meaningful results for unknown objects, the set of known objects and classes is often divided into a *training set* and a *test set*. Only the training set is used to construct the classifier $f(x)$ whereas the test set is used to assess its quality. If $f(x)$ produces the correct classes for most objects in both the training and the test set, it can be assumed to be suitable for classifying unknown objects in general. Designing a classifier function can be seen as a special kind of optimization problem. The approach described in the previous section can be adapted to classification problems by designing a visual analytics system that supports a human analyst in creating a classifier function. As before, this approach is promising whenever human expert knowledge may be required for designing the classifier or judging the classification results. Figure 2.7 shows a relabelled version of the optimization cycle in Figure 2.6. Here, the classifier is the parameter to be optimized and the test set acts as a set of constraints. The classification of this test set provides the evaluation of the classifier which can be assessed either automatically or by the user, possibly using a visual interface. The cycle is repeated until the user accepts the classification results and the classifier. Again, optional automatic support can be provided by an automatic refinement of the classifier based on the evaluation of the test set.

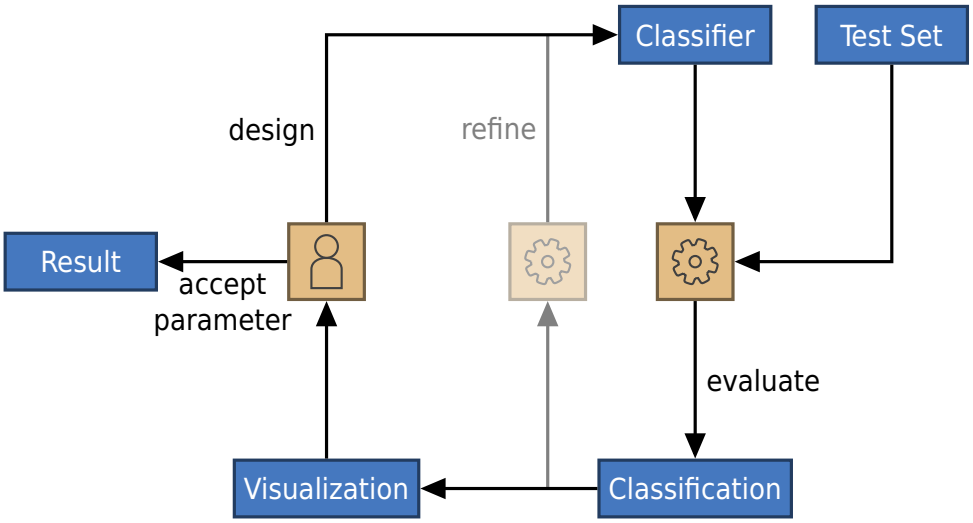


Figure 2.7: A visual analytics approach to classification problems.

3

Reconfigurable Manufacturing Layout Optimization

This chapter presents *Sam* ('Simulated advanced manufacturing'), a visual analytics system for planning process layouts for reconfigurable manufacturing systems (Figure 3.1). The increased flexibility of these systems as compared to a traditional transfer line brings with it a number of specific requirements for planning their processes. The prototypical implementation of a layout planning system that includes both an automatic and an interactive visual evaluation of layouts will show how these requirements may be handled in a future layout planning system. As the name implies, *Sam*'s major analysis component is a simulation of manufacturing processes.

The visual analytics method is arguably well suited for this problem. Given a formal description of a problem and enough time, a purely automatic approach can often find an optimal solution more reliably than an interactive process. Visual analytics, however, represents a more dynamic approach, integrating the human planner into the analysis. It can benefit from human experience, implicit knowledge, and judgement and is there-

Parts of this chapter have previously been published in:

Wörner, M. and Ertl, T. (2011b), 'Visual Analysis of Advanced Manufacturing Simulations', in *EuroVA 2011: International Workshop on Visual Analytics* (Goslar: Eurographics Association), 29–32;

Wörner, M. and Ertl, T. (2013a), 'Simulation-based Visual Layout Planning in Advanced Manufacturing', in *Proceedings of the 46th Hawaii International Conference on System Sciences – HICSS 2013* (IEEE), 1532–1541 ©2013 IEEE.

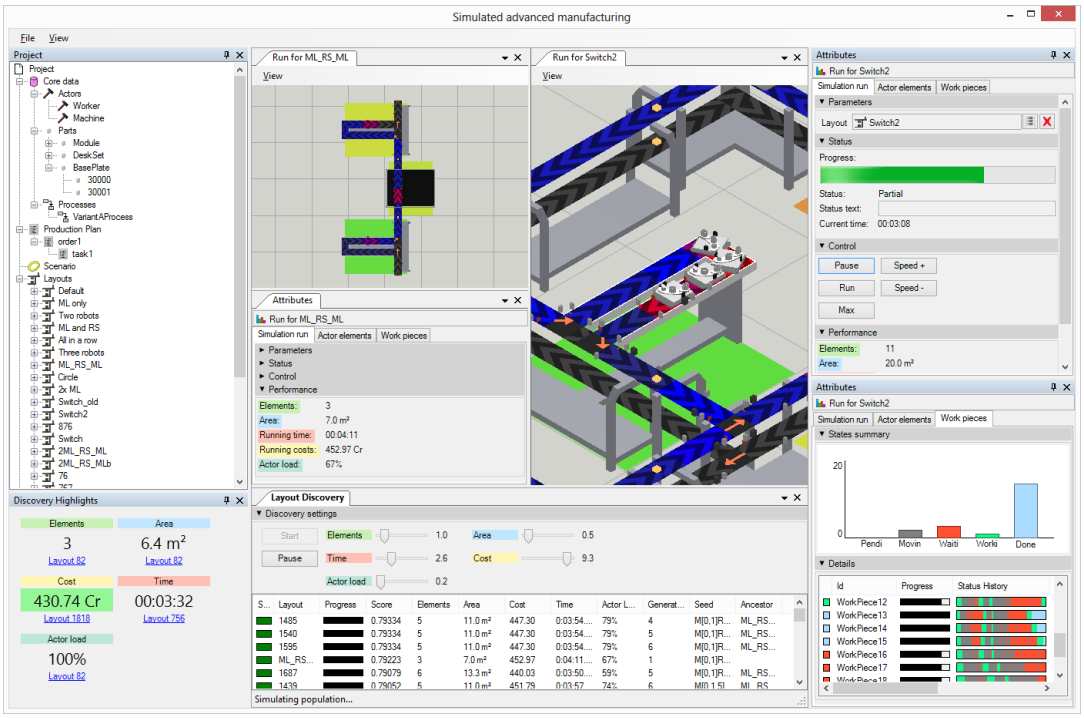


Figure 3.1: *Sam* ('Simulated advanced manufacturing'.)

fore not as dependent on a complete and formal description of the problem. With the human being a part of the decision making process, it can also increase the confidence in a proposed solution.

As a visual analytics system, *Sam* relies on human guidance through the parameter space and while it can provide an automatic analysis and evaluation of a given manufacturing layout, the ultimate assessment will have to be made by a human expert, who might take into account requirements that are not or cannot be formally expressed in the analysis model. In fact, there may not be a single value that is to be optimized but rather a whole set of targets that work against each other: An analyst may be looking for a layout that is both time- and cost-effective, yet there may be the option of adding another element that shortens the production time but increases the production costs. Deciding on whether or not to add this element is often not a matter of algorithmic optimization but of personal preference and might be subject to other, possibly implicit considerations.

Sam has been designed with a specific reconfigurable manufacturing system in mind. This system is a prototypical reconfigurable manufacturing system intended for teaching advanced manufacturing techniques. It was chosen because its modularity creates the flexibility sought for in future manufacturing systems and because, as a physical prototype platform, it can serve as common ground when communicating with domain experts.

3.1 The manufacturing domain

Manufacturing, in a very general sense, is the transformation of raw materials into products. A product may be sold to a consumer or it may be used as a part in another manufacturing process. In both cases, the finished product is more valuable than the raw materials used in its production. For centuries, manufacturing was a purely manual process (‘manu factus’ is Latin for ‘made by hand’). With the industrial revolution, machines powered by steam and later electricity were introduced into the manufacturing processes and greatly increased the productivity of many repeatable tasks. Along with the standardization of parts and interfaces, this made it possible to mass produce large numbers of virtually identical instances of the same product. This increased the importance of the efficiency of the manufacturing processes, as any delay or inefficiency was now multiplied across many process instances and even small improvements to a process provided possibly decisive advantages over the competition. It also required a high manufacturing precision, so that any parts produced met the defined standards. To address these requirements, Frederick Taylor (1856–1915) established what he called a scientific approach intended to separate the planning of the work from the actual execution. He conducted experiments to determine how to best execute a given manufacturing task and based on these findings, work planners designed very detailed assignments for each worker, broken down into elementary steps with precise instructions and time targets. Like Taylor, Henry Ford (1863–1947) optimized processes in his Ford Motor Company by dividing tasks into elementary steps, but while Taylor focused on manual labour, Ford made extensive use of machines and used conveyor belts to transport work pieces between workers and machines.

Optimizing manufacturing processes usually resulted in their strict linearization. With every work piece taking the same, predefined path through the factory, workers could specialise on a small set of highly repetitive tasks they had to perform and minimal delays were caused by having to prepare for a new task or decide what to do next. Additionally, replacing manual craftsmanship with machine work ensured that all products were manufactured equal and with high precision, manufacturers were able to provide generic replaceable parts that would fit any product and greatly simplified repairs.

However, while mass production made manufacturing large quantities much more efficient, it also introduced new limitations. Most notably, it was not easily possible to produce different variants of the same product while keeping the benefits of worker specialization and process efficiency. In his autobiography, Ford concluded:

> It is strange how, just as soon as an article becomes successful, somebody starts to think that it would be more successful if only it were different . . . Salesmen always want to cater to whims instead of acquiring sufficient knowledge of their product to be able to explain to the customer with the whim that what they have will satisfy his every requirement—that is, of course, provided what they have does satisfy these requirements.
>
> Therefore in 1909 I announced one morning, without any previous warning, that in the future we were going to build only one model, that the model was going to be 'Model T,' and that the chassis would be exactly the same for all cars, and I remarked:
>
> 'Any customer can have a car painted any colour that he wants so long as it is black.' (Ford and Crowther, 1922)

Customers may have gladly accepted these restrictions in Ford's days, seeing as it helped to make motor cars affordable for a much wider customer base. However, manufacturing enterprises exist in a complex and dynamic environment and customer expectations, along with society and technology, have seen drastic changes in the past one hundred years. Other dynamic influences include the market situation and laws and regulations. The market, more specifically the supply of raw material and the demand for the company's products, determines both the raw material prices and the product prices that can be realized. The relation of these two figures may decide whether or not a certain manufacturing process is economically feasible. Also, a drastic decrease in the demand for a product may mean a process is no longer viable if the economic considerations of this process assume a certain minimum number of products sold per time unit. Similarly, a change in government may bring a different political agenda that may influence the viability of manufacturing processes based on, for example, their environmental or social implications.

As a consequence, adaptability is a major topic in today's manufacturing industry, and the ability to adapt quickly and efficiently to changing requirements is seen as a central feature of future factories (Westkämper, 2007). In the turbulent environment of changing supply and demand, prices, legislature, and technology, a system is expected to be able to adapt to such changes and operate efficiently at the same time. This requirement was demonstrated very dramatically by the economic crisis that began in 2007. In less dramatic times, one important factor is that today, products are often available in many different variants. When buying a car, customers can now not only choose from a variety of colours, but can often select an engine, a chassis, interior materials, and various additional equipment from

automatic transmission to xenon headlights. As a result, there are modern production lines where one will rarely find two cars of the exact same configuration. This difficulty of predicting what customers will order in the near future leads to smaller batch sizes and shortens the time that can be planned ahead. The adaptability of a system, i.e. its ability to adjust to a changing environment, requires a flexible system structure, that creates potential for changes to the system in the first place.

So while Taylor's scientific approach, which also suggested raising wages, was initially received well among workers, it eventually faced criticism, not only because workers began to feel abused by the strict efficiency targets issued by theorists but also because of concerns that providing workers with preset plans on how to best perform a task would preclude any innovation from practical experience. In an ever-changing environment, however, there usually is not a single optimal process that can be designed once and then only needs to be executed over and over again. Instead, there is a need for manufacturing processes to adapt to and evolve along with their environment.

When the Japanese economy recovered from the Second World War, it adopted a philosophy called *Kaizen* (Imai, 1986) ('change for the better'). In contrast to Taylor's view of the worker being a workhorse that only executes plans devised elsewhere, Kaizen encourages a continuing education of the workers. It demands continuous improvement in all parts of the enterprise and a structure that allows suggestions to be put forth on all levels of the organization. Kaizen is closely related to similar paradigms such as the continuous improvement process, which recommends detecting opportunities for improvement and implementing improvements in repeating cycles of 'plan', 'do', 'check', and 'act', and lean manufacturing, which strives to minimize waste. 'Waste', in this sense, is any investment of resources that does not create value for which the customer is willing to pay. One method of identifying places where resources are wasted without generating value is to map not the material flow but rather the value flow through a manufacturing process—a technique called value stream mapping (Rother and Shook, 2003). In the 1980s Western manufacturers began to imitate these principles. They have become common practice since then.

In addition to organizational developments, manufacturing technology has evolved to allow for more process flexibility. Many of the machines used in industrial production are machine tools—machines that use some kind of tool to modify a work piece. They greatly increased the precision at which parts could be reproduced and thus were key factors in the development of mass production and replaceable parts. Early machine tools were operated by hand but restricted the movement of the work piece or the tool in a way that ensured a higher precision than could be achieved by freehand

operation. Manual control of the machine was replaced by automation with the introduction of numerical control (NC) machines. These were able to perform work steps according to a sequence of commands, originally stored on punched tape. Later, computers were integrated into the machines, creating computer numerical control (CNC) machines, which brought new possibilities for transferring command sequences via more convenient data storage devices such as USB flash drives or directly via a network connection.

Beyond the individual machine, there are different ways of organizing the process as a whole. A process may use several separate machines without any strict connection between them and with work pieces being transported manually from one machine to the next. This is a very flexible arrangement well suited for the construction of unique items such as prototypes or highly specialized customer orders. It is, however, very inefficient when producing large quantities of the same item. Such a scenario is better handled by a transfer line, which connects individual machines or work places by an automatic material transportation system such as a conveyor belt. While very efficient for highly repetitive processes, this approach is also very inflexible, and any problem along the belt may bring the whole system to a standstill. With the increasing demand for flexible, yet efficient processes for producing limited numbers of products in many similar variants, flexible manufacturing systems now try to fill the gap between these extremes. One way to increase the flexibility of a manufacturing system is to make it reconfigurable (ElMaraghy, 2005; Mehrabi et al., 2000). In this sense, a manufacturing system is reconfigurable if its elements—the hardware as well as the software—can be rearranged quickly and with little effort. This has significant implications on process planning: Whereas before, a transfer line was set up once and changed only very rarely, a reconfigurable system may be changed many times a day. The ability to regularly adapt the physical process layout to the task at hand, the order situation, short-term customer requests, and other elements of the system environment, greatly improves the adaptability of the manufacturing system as a whole but also creates a demand for new ways of planning the process, as planning layout changes for reconfigurable manufacturing systems is a complex task and requires capable tools. In contrast to traditional approaches, plans for reconfigurable manufacturing systems are not generally made weeks in advance. To benefit from the flexibility, a production planner needs to be able to make adjustments on short notice to manufacture a limited number of items, then adjust the process for the next task. Planning tools need to assist the planner in finding a good process layout, while also communicating the benefits, disadvantages, and risks of a layout. This is an iterative process that involves human judgement,

assisted by automatic evaluations, such as the calculation of key figures for certain layouts. Consequently, it is a promising application for the visual analytics approach as described in Chapter 2.

3.2 Related work

Klopper et al. (2011) presented a system that assists a dispatcher in deciding on a machine schedule in an adaptable manufacturing environment. Given a problem description in the form of a precedence graph of the various production tasks and their dependencies, the system computes possible machine schedules, calculates key figures, and presents them to the user. The user can preselect relevant schedules in a sorted list and compare them in a radar chart of their key figures. *Sam* focuses on the conveyor layout of a reconfigurable manufacturing system and extends the approach beyond supporting users in deciding on one of several precomputed solutions to supporting them in designing and adjusting process layouts in an iterative process. Also, *Sam* uses simulation and an evolutionary algorithm rather than a graph search. The simulation can optionally be played back in real-time, as Rohrer (2000) calls for the inclusion of visualization and animation in manufacturing simulations to help in verification and validation, understanding of results, communication of results, providing a feel for the accuracy and usability of the model, and achieving credibility. Regarding requirements for the visualization, he lists interactivity, realism (again to support the credibility of the model), performance, flexibility, and ease of use.

Dangelmaier et al. (2005) describe a concept for a discrete manufacturing system simulation which is supported by virtual and augmented reality to create an intuitive and understandable user interface. In *Sam*, users define their layouts by placing and connecting elements in a graphical representation. Other approaches exist. Schulze et al. (2000), for example, emphasize that modelling manufacturing simulations using a flexible simulation language is still a valid approach and provide an example of describing an assembly line simulation model using one such language.

Doleisch and Hauser (2002) presented a system that uses an interactive brushing tool for the specification of a degree-of-interest function in simulation data presented in both a 3D view and different kinds of diagrams, all linked by the degree-of-interest function. Section 2.5 already introduced *HyperMoVal* by Piringer et al. (2010) and *Vismon* by Booshehrian et al. (2012) as systems that perform visual analysis on implicit parameter spaces. *Sam*'s parameter space, however, is not continuous and the automatic sampling is partly replaced by a user-driven construction of layouts to be evaluated.

Figure 3.2: The iTRAME reconfigurable manufacturing system. (Image courtesy of IFF, University of Stuttgart.)

Aigner et al. (2011) provided an overview of various methods for visualizing time-dependent data, considering static visualization, dynamic representations and event-based visualizations. They state that the latter plays an important role in the field of simulation, because it can present the data for effective analysis and allows interacting with the simulation in real-time. Wenzel et al. (2003) presented a taxonomy of visualization techniques for simulation in production and logistics and gave examples on which kinds of visualization are suitable for data specific to manufacturing.

3.3 The iTRAME system

Sam is a prototypical implementation of a layout planning system for a reconfigurable manufacturing system. It can be used to plan and evaluate layouts for iTRAME processes. The iTRAME ('intelligent Transformable Assembly and Manufacturing Equipment') system (Riffelmacher et al., 2007) is an existing, prototypical realization of a reconfigurable manufacturing system (Figure 3.2). It has been developed by the University of Stuttgart's Institute of Industrial Manufacturing and Management (IFF) in collaboration with the industrial education company Festo Didactic. It is meant to act as a physical model factory and is part of a learning factory to be used for training manufacturing professionals. It is reconfigurable, because elements can be connected or disconnected easily using standardized con-

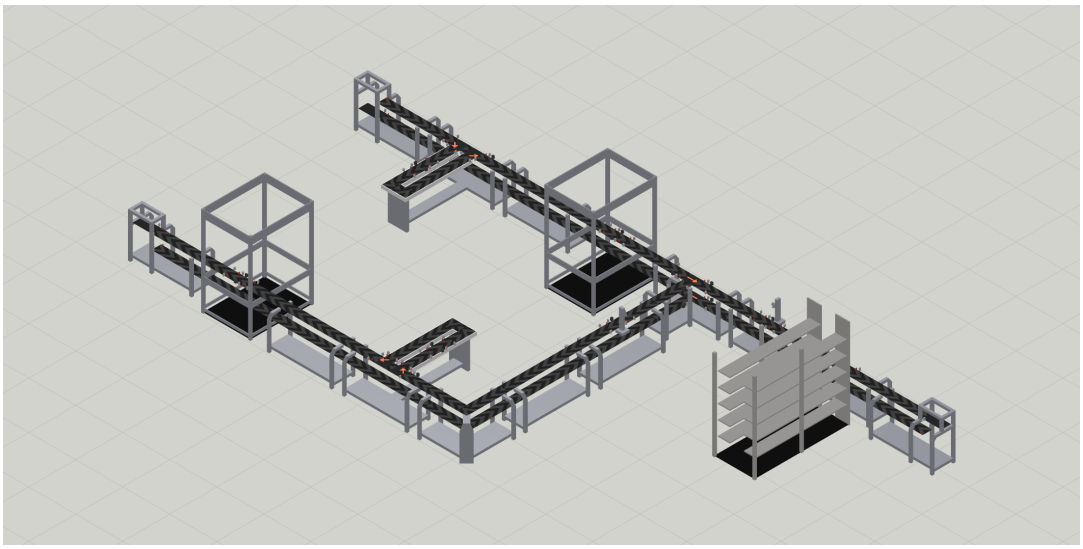


Figure 3.3: An example of an iTRAME layout as displayed by *Sam*. A two-level conveyor belt transports work pieces between elements.

nectors that distribute the necessary power, pressurized air, data, and control signals throughout the system. Also, the size and weight of the elements are such that a single person can move and rearrange them using a pallet jack. A central control computer detects the current layout based on the connectivity information transmitted by the elements. Figure 3.3 shows an example of an iTRAME layout as displayed by *Sam*'s user interface.

All iTRAME elements contain a conveyor belt segment that connects to those of neighbouring elements. These conveyor belts can transport work pieces from one element to the next, forming an automatic material flow through the manufacturing process. They also transport work pieces within the element, for example from the previous element to the position where a work step is performed on the work piece and from there to the next element. Each element has a second conveyor belt on a lower level that moves in the opposite direction. This makes any element connection bidirectional and while all actual work is done on the upper level, the lower belts can be used to transport work pieces in the opposite direction without adding additional elements, conserving space.

iTRAME layouts can be composed of a number of different element types (see Figures 3.4 and 3.5):

The **linear transport** element is the most basic of the iTRAME elements. It contains a conveyor belt section that transports work pieces along to the next element. The lower level transports work pieces in the opposite direction.

The **corner transport** element is similar to the linear transport element

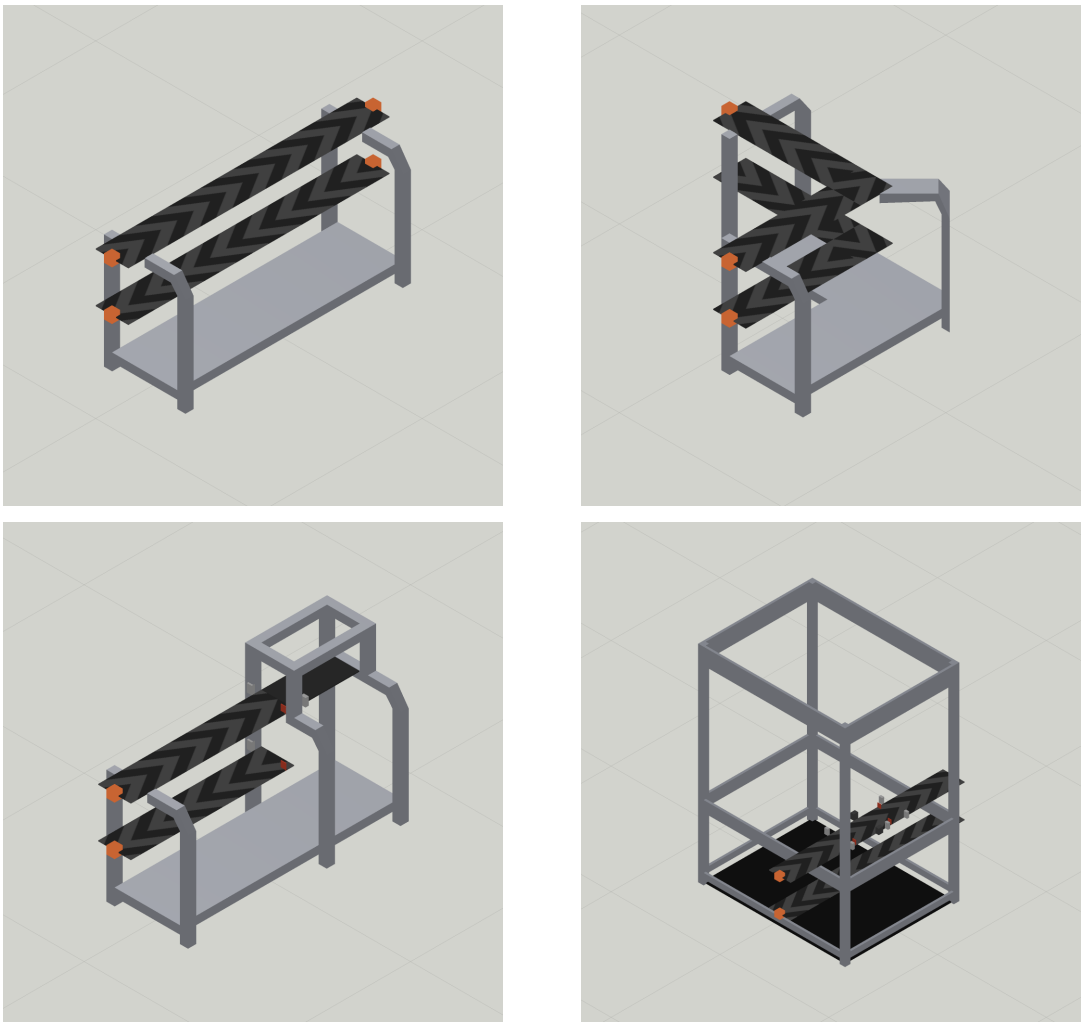


Figure 3.4: The linear transport, corner transport, lift, and robot station elements as displayed by *Sam*

but creates a turn in the conveyor belt.

The **lift** element transports work pieces between its upper and lower conveyor belt. It is the only way for a work piece to switch belt levels. Whether a lift element moves work pieces from the bottom to the upper layer or the other way around depends on the movement direction of its conveyor belts (which always move in opposite directions). It is also the only element with only one (two-layer) connection to a neighbouring element.

The **robot station** element contains a robotic arm in a safety cage. The upper conveyor belt moves work pieces into a defined work position where the robotic arm can perform assembly and disassembly tasks on the work piece, i.e. it can add or remove parts from the work piece.

The **manual labour station** element steers an element from the main

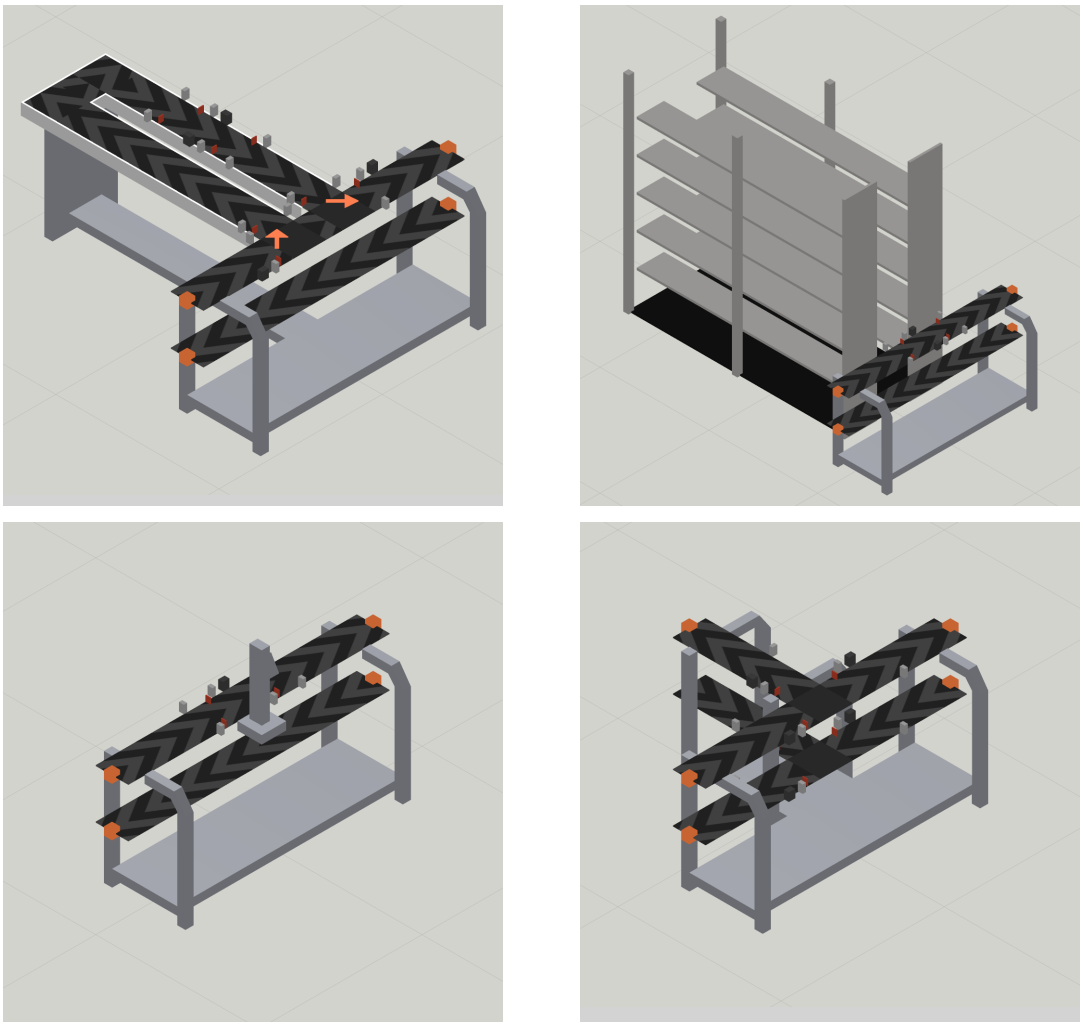


Figure 3.5: The manual labour station, automatic storage, vision station, and switch elements as displayed by *Sam*

conveyor belt onto a separate segment and presents it to a human worker. A screen attached to the station shows instructions on the tasks that need to performed on the work piece and ask for confirmation once these tasks have been completed. Upon release, the work piece moves back onto the main belt. The manual labour station can be used to introduce work pieces into the process (by adding a task to place a new work piece onto the conveyor belt) and to remove the finished products (by adding a task to remove the presented piece and store it). The separate segment on which the manual work is performed allows the entire element to be bypassed by work pieces that do not require manual work steps.

The **automatic storage** element can remove work pieces from its upper conveyor belt and place them on a shelf or take work pieces from the

shelf and place them on the belt. It can be used both to store the initial configurations of work pieces and to receive the completed products. It can also serve as a buffer for intermediary states.

The **vision station** element can visually inspect a work piece using a camera and determine whether it has been manufactured successfully or needs to be reworked or scrapped.

The **switch** element has a total of three connections to neighbouring elements and can represent a fork in the conveyor belt layout. The possibility of forking and joining conveyor belts has significant implications. An iTRAME progress is not necessarily a strictly linear sequence of elements. By combining multiple switch elements, complex networks can be formed, increasing both the potential and the complexity of the system. Switches can be used to place work stations in a parallel arrangement and distribute work pieces between them or for creating entirely different paths, possibly for manufacturing different variants of the same product. iTRAME demonstrates the flexibility to change not only the physical process layout but also the logical flow of work pieces on short notice. Once an iTRAME layout contains switches, the order in which work pieces are introduced into the process is no longer the only way to steer it. If multiple work stations are able to perform the next work step on a work piece, various strategies can be employed to decide which work piece goes where. These can take the current state of the process into account and might act based on the current work load of a work station, the current work piece density on different segments of the conveyor belt or the individual priority of a work piece.

Work pieces travel along the conveyor belt on carrier plates that are equipped with RFID tags. Some iTRAME elements contain RFID readers that can detect these tags and identify a work piece. The is used both by work stations to determine what work steps need to be performed on the piece and by switches to determine where a given work piece needs to go. When the conveyor belts move continuously, all work pieces travel at the same speed. Since there is the occasional need to stop individual work pieces while not affecting others—at the work position of a robot or manual labour station or at a switch segment while a work piece is being identified or while the switch is occupied—some iTRAME elements contain mechanical bolts that can be extended to block work pieces and hold them in place.

3.3.1 The model product

The creators of the iTRAME system also designed a model product to be produced in the manufacturing process. It is a desktop set meant to hold

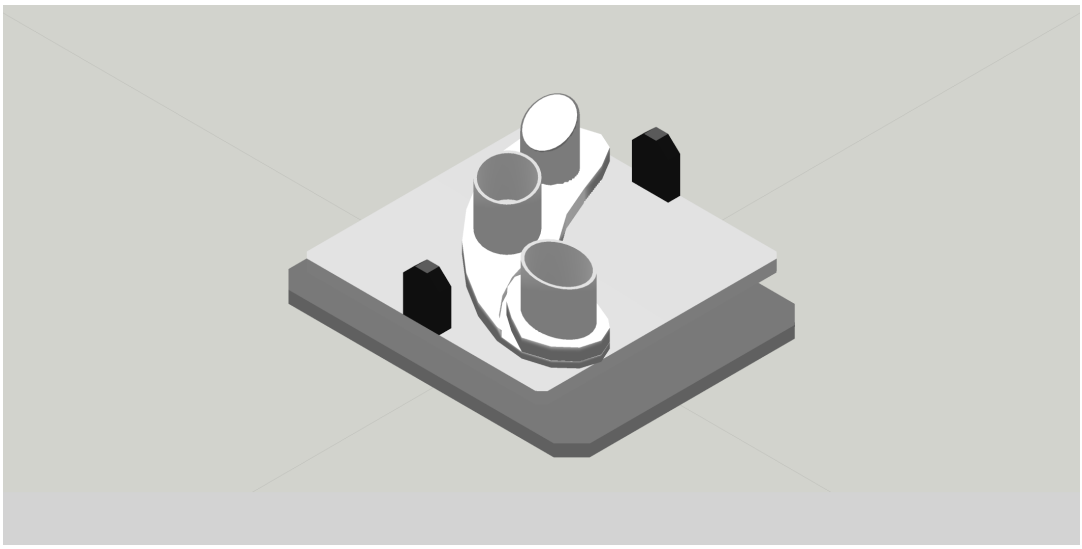


Figure 3.6: The desktop set that serves as an example product for the iTRAME manufacturing system on top of a carrier plate with RFID tags (black). The set in this image contains two cups and a clock.

pens or other small items (Figure 3.6). The set is assembled from a number of components that can be combined in numerous ways, creating a product with a large number of variants. This is in line with the general assumption that the factory of the future will have to handle production scenarios in which the number of product variants increase while the batch sizes decrease.

Each desktop set contains a sickle-shaped base plate with three fixtures, one of which can be swung out. Each fixture can hold one of a number of different components. There are 10 types of cups, 3 types of lamps, and 4 types of gauges. Additionally, there are two different types of base plates with the pivot point of the mobile fixture either towards the centre of the base plate or at the outer edge. If fixtures may be left empty, these combinations allow for $2 \cdot (10 + 3 + 4 + 1)^3 = 11{,}664$ product variants.

3.4 Modelling iTRAME processes

Sam aids the user in evaluating a given layout by providing key figures and a presentation of its operation. Both are obtained by computing a simulation of the layout. This simulation has a virtual representation of the layout execute a manufacturing process while tracking the movements and state changes of all work pieces. This requires a comprehensive and detailed model of the structure and behaviour of all involved elements and

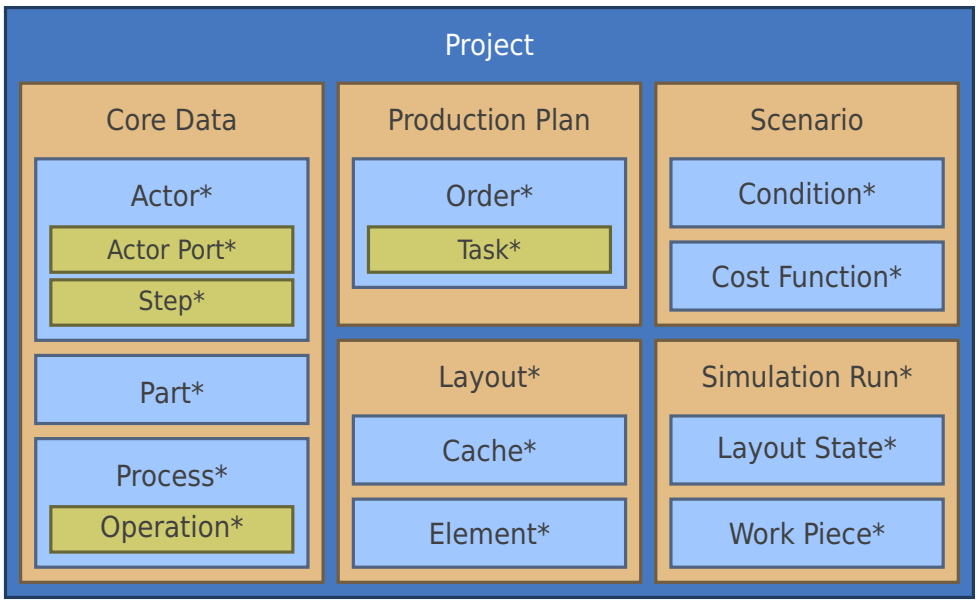


Figure 3.7: An overview of *Sam*'s data model. Asterisks mark elements that may be present multiple times.

the process. Figure 3.7 shows the general structure of the model *Sam* uses.

A **project** combines all information necessary to perform a layout optimization. This information is divided into five sections: The core data, the production plan, the scenario, the layouts, and the simulation runs.

3.4.1 Core data

The core data is a collection of generic information on the products and production processes. It defines the parts, which include both products and intermediate states, the processes, which describe how products can be manufactured, and actors, which describe how layout elements may contribute to a process.

A **part**, for the purpose of *Sam*'s simulation, is merely a certain state of a work piece. A work piece is defined in the simulation runs section and indeed does not exist outside the context of a simulation run. Process operations performed on a work piece may change its state. Whenever the current state of a work piece matches the definition of a part, the work piece can take the role of this part in a process operation.

Parts are defined in terms of attributes, slots, and constraints. An attribute is a named property of a work piece that can be set to any value. A slot is a named property that can be filled with another part. Constraints demand that an attribute has a certain value or a slot is filled with a part of a certain type. Parts can have a base part from which they inherit all

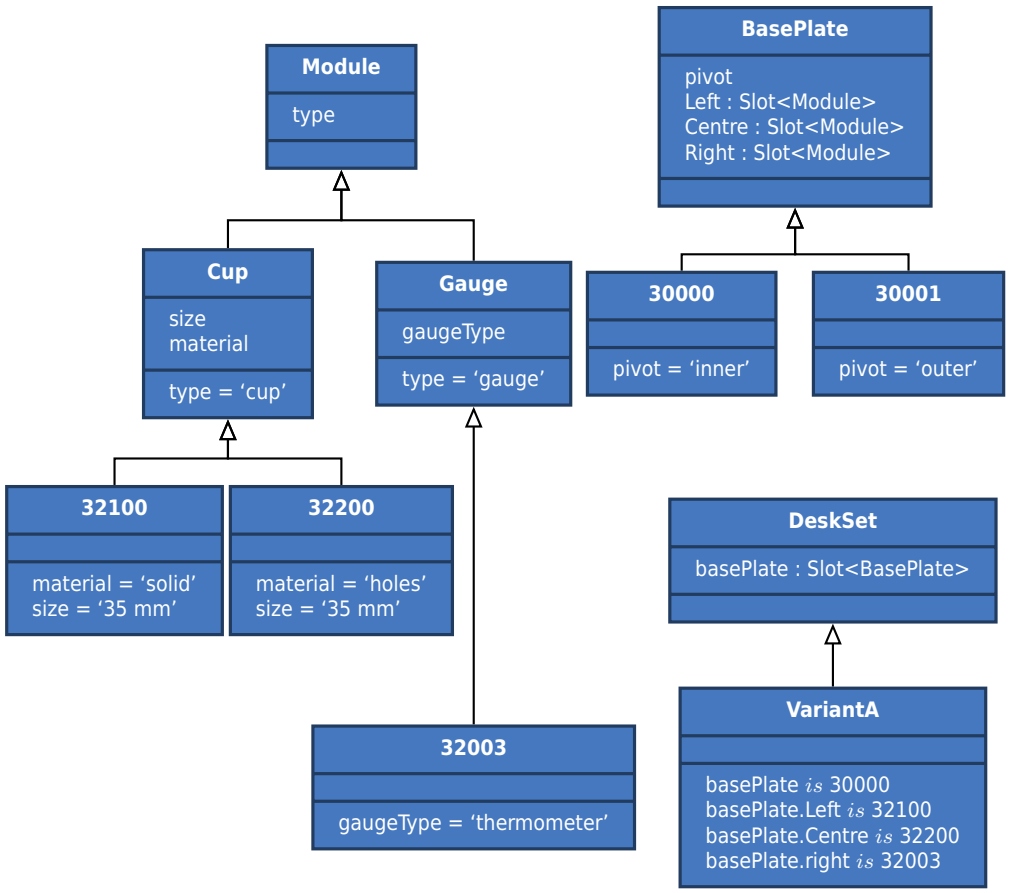


Figure 3.8: A definition of several component parts and a product called 'VariantA'.

attributes, slots, and constraints. A work piece matches a part definition if it has all the attributes and slots of the part and its attribute values and slot contents comply with the part's constraints.

Figure 3.8 shows an example of part definitions that make up definition of a product variant of the desk set model (Figure 3.6). There are *Modules*, which have an attribute named *type*. There are *Cups* and *Gauges*, which both are *Modules* with their *type* attribute set to 'cup' and 'gauge' respectively. *Cups* have the attributes *size* and *material* while *Gauges* have an attribute *gaugeType*. The part *32100* is a *Cup* of *material* 'solid' and *size* '35 mm'. The part *32200* is a *Cup* of *material* 'holes' and *size* '35 mm'. The part *32003* is a *Gauge* of *gaugeType* 'thermometer'.

A *BasePlate* is a part with an attribute *pivot* and three slots *Left*, *Centre*, and *Right*, which can hold *Modules*. *30000* is a *BasePlate* with *pivot* set to 'inner' and *30001* is a *BasePlate* with *pivot* set to 'outer'. The *DeskSet* is a part with a slot *basePlate*, that can hold a *BasePlate*. The product *VariantA* is a *DeskSet* that has a base plate which contains *32100* in its left slot,

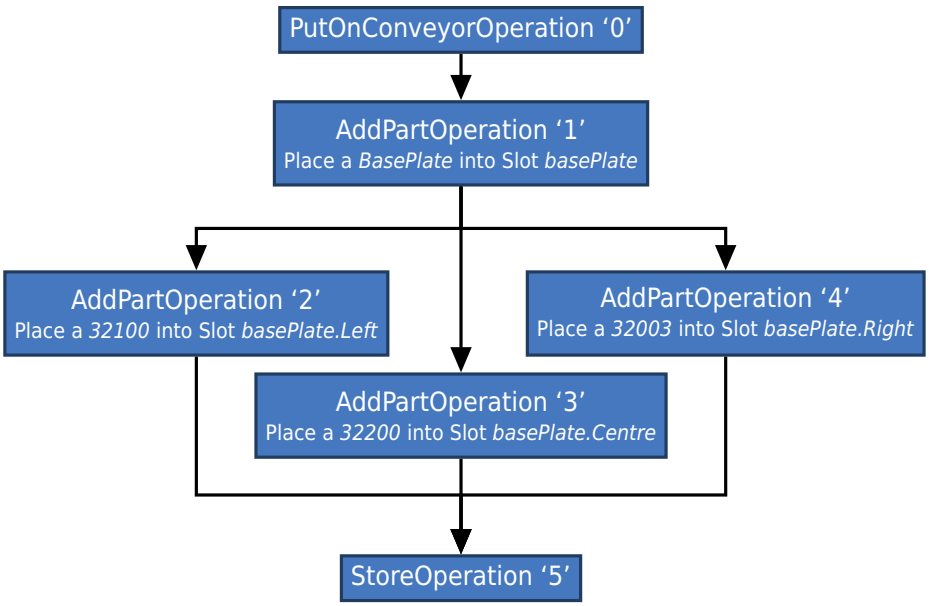


Figure 3.9: A process to manufacture *VariantA* from Figure 3.8.

32200 in its centre slot, and *32003* in its right slot.

A **process** is a set of **operations** that describe the manufacturing of a product, which is a certain, final state of a work piece and thus a part. In *Sam*, a process is made up of three types of operations:

- A *PutOnConveyorOperation* places an empty carrier plate on the iTRAME conveyor belt.
- An *AddPartOperation* combines two parts.
- A *StoreOperation* removes a part from the conveyor plate and places it in a storage container.

Processes are not a linear sequence of operations. Instead, each operation of a process defines a set of dependencies, operations that must have been completed before this operation can be performed. Figure 3.9 shows a process that can be used to manufacture *VariantA* from Figure 3.8. The *AddPartOperations* that fit the three modules into the slots of the base plate can be performed in any order once operation ‘1’, which places the base plate, has been completed. The *StoreOperation* ‘5’ requires that all other operations in the process have been finished.

An **actor** describes a role that a layout element such as a manual labour station or a robot station may play in the process. An actor defines both a set of actor ports and a set of steps. An **actor port** is an abstract place, where an actor may get, put, or operate on a part or work piece. This may

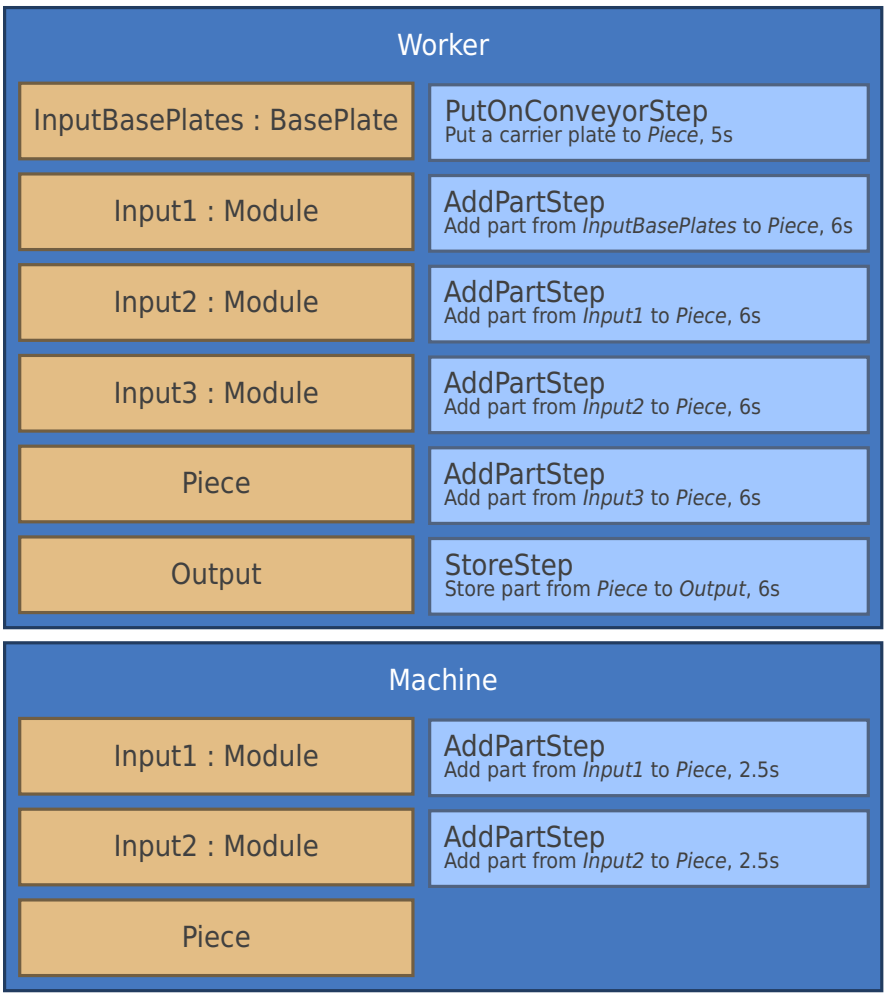


Figure 3.10: Two actors and their actor ports (left) and steps (right).

refer to a shelf or box, where parts are placed or the work position on the conveyor belt. A **step** defines a way for the actor to perform a process operation. There are three types of steps that correspond to the three types of operations:

- A *PutOnConveyorStep* places a new carrier plate at a specified actor port.
- A *AddPartStep* takes a part from one actor port and adds it to a part located at another actor port.
- A *StoreStep* takes a part from one actor port and places it at another actor port.

All steps also specify the time it takes this actor to perform the step. Figure 3.10 shows the two actors ‘Worker’ and ‘Machine’. Here, a ‘Worker’

is defined as an actor that has six actor ports and can execute six actor steps. The ‘InputBasePlates’ port can be used to access parts of type ‘BasePlate’ (as defined in Figure 3.8) whereas the ‘Input1’, ‘Input2’, and ‘Input3’ ports can supply parts of type ‘Module’. The port ‘Piece’ is used to access the work piece the worker is currently working on and ‘Output’ is where the worker stores finished products. These ports are referenced by the steps. The *PutOnConveyorStep* creates a new work piece by placing a carrier plate at the ‘Piece’ port. The *StoreStep* removes a work piece from ‘Piece’ and places it at ‘Output’. The four *AddPartSteps* take a part from one of the input ports and add it to the work piece at ‘Piece’. If a process layout connects ‘InputBasePlates’ to a source of *BasePlates* and the parts *32100, 32200,* and ‘32003’ are available at one or more of the other input ports, then this actor can complete the entire process for manufacturing the product ‘VariantA’ (Figure 3.9).

The second actor, ‘Machine’, has only two input ports and a port for the current work piece. It has no *PutOnConveyorSteps* or *StoreSteps* and can only add a part from either input port to the work piece at ‘Piece’, but it does this quicker than the worker (in 2.5 seconds as opposed to 6 seconds for the worker). It cannot complete an entire process for ‘VariantA’ on its own but might nevertheless make a worthwhile addition to a layout.

3.4.2 Production plan

The suitability of a layout depends on the manufacturing task that is to be performed. The production plan defines this task by specifying which products need to be produced. It contains a set of **orders**, which specify a priority and a due date as well as a number of tasks. A **task** defines the part to be produced and a quantity. The information in the core data can be used to determine which processes can produce this part and which actors can take part in these processes. The production plan also defines the general production strategy, which determines the order in which work pieces are introduced into the system and the reasoning behind routing work pieces through the layout.

3.4.3 Scenario

The scenario defines global conditions and restrictions for the layout optimization. This may include space constraints (the layout must fit into a given area) or fixed elements (such as an immobile container for finished products). It also determines the parts that are initially available as raw material. Most importantly, however, it defines the cost function that assigns a cost value to a specific layout operation. Domain experts may define actual

Element	Idle	Working
Transport	0.01 Cr/s	-
Corner	0.01 Cr/s	-
Lift	0.05 Cr/s	0.05 Cr/s
Switch	0.05 Cr/s	0.05 Cr/s
Manual Labour Station	0.85 Cr/s	0.85 Cr/s
Robot Station	0.01 Cr/s	0.15 Cr/s

Table 3.1: The cost function used for the prototype.

monetary values, but for the purpose of this prototypical demonstration, the cost function uses a fictional currency ‘Credits’ (Cr). Its coefficients have been chosen based on the assumptions that

- the cost that an element incurs depends on how long it has been operating,
- the transport and corner elements have the lowest operating costs,
- the more complex switch and lift elements have medium operating costs,
- the manual labour station has high operating costs,
- a robot station has a low operating cost when idle and a medium cost when working.

The resulting cost function used for the prototype is given in Table 3.1.

3.4.4 Layouts

When searching for a suitable layout, a user will usually create a number of different layouts that fulfil the conditions set by the production plan and the scenario to varying degrees. These layouts are collected in the layouts section of a project, where they can be accessed, simulated, compared, or deleted.

A layout defines the spatial arrangement and connectivity of its iTRAME elements and caches. **Elements** can be connected to other elements via their iTRAME ports. The lift element has one port, the switch element three. All other elements have two ports. Since all elements contain an upper and a lower conveyor belt, each iTRAME port represents two conveyor connection points.

Robot stations, manual labour stations, automatic storage elements, and vision stations are actor elements. These can take the role of an actor and perform actor steps and thus process operations. For each actor element, the layout specifies its actor (as defined in the core data) and links the actor ports to caches or the work position of the actor element itself. The work position of an actor element is the location on the conveyor belt where a work piece is held while the actor element performs an operation on it.

Caches are not part of the actual iTRAME system but represent a generic storage opportunity for parts (such as a shelf, a table, or a box). By linking the actor ports of actor elements to caches, the layout defines where an actor element gets the pieces it adds to a work piece and where it places work pieces it removes from the conveyor belt. Caches keep track of the material flow by managing an inventory of parts taken from them or placed into them. By restricting which parts are available in or accepted by a cache, one can influence which actor steps and consequently which process operations an actor element can perform.

Figure 3.11 shows an example of an actor element ‘ML1’ and its connections. Since its actor is ‘Worker’ (as defined in Figure 3.10), it has five actor ports. Its ‘InputBasePlates’ and ‘Input1’ ports are both connected to a cache named ‘PartsCache’. Its ‘Output’ port is connected to another cache named ‘ProductCache’ and its ‘Piece’ port is connected to its own work position. Considering that ‘PartsCache’ contains *BasePlate* and *32100* parts and that ‘ProductCache’ can receive *VariantA* parts, ‘ML1’ can perform four of the six actor steps of ‘Worker’. It cannot, however, perform the steps adding parts from ‘Input2’ and ‘Input3’ because these ports are not connected. If ‘ML1’ were to execute the process from Figure 3.9, it would be able to perform the process operations ‘0’, ‘1’, ‘2’, and ‘5’ but not ‘3’ and ‘4’ because no port used in one of its *AddPartSteps* has access to the necessary parts *32200* or *32003*.

3.4.5 Simulation runs

Sam’s primary means for evaluating a process layout is simulating it. A simulation run represents the simulation of a given production plan on a given layout using a given definition of core data and scenario and keeps track of the results. Multiple simulations can be run on the same layout and, if parameters or conditions change or stochastic elements are introduced, may produce different results. In addition to information on the state, progress, and results of the simulation, a simulation run contains states and work pieces.

Some of the items contained in a process layout are simulation objects. A simulation object is an item that has dynamic properties in addition to

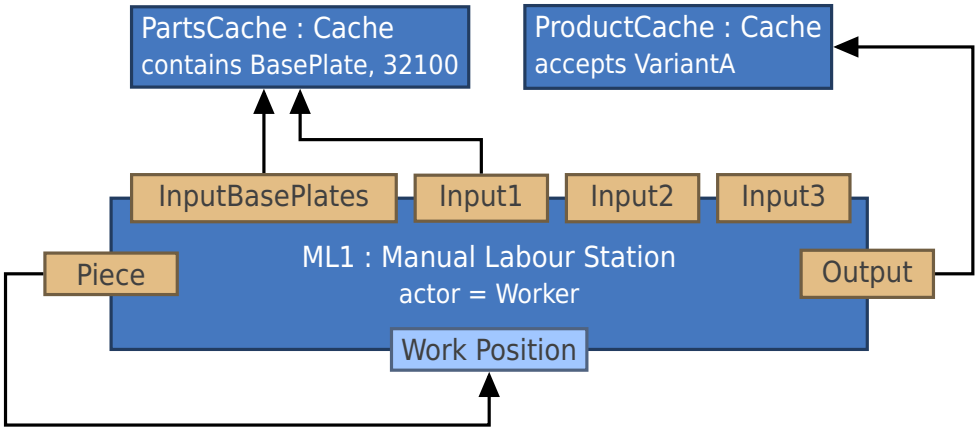
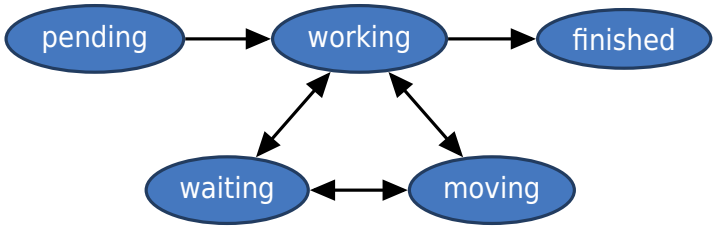


Figure 3.11: An actor element connected to two caches through its actor ports.

the static properties defined in the layout. The position and connectivity of an iTRAME element, for example, is a static property that does not change during a simulation run. The current progress of a process operation performed by an actor element or the current vertical position of the movable conveyor segment of a lift element, however, are dynamic properties whose values may change during a simulation run. The collection of a all dynamic properties of a simulation object is its **state**. A simulation run keeps a state for each simulation object in the associated layout. Because states are part of the simulation run and not the layout itself, multiple simulation runs can operate on the same layout simultaneously without influencing each other.

Work pieces are another set of dynamic items in a simulation run but unlike states, they have no static counterpart in the layout. Instead, a simulation run contains one work piece for every product requested in the production plan orders. Initially, a work piece is in a ‘pending’ state, meaning that its production has not yet started. In this state, a work piece is not a physical entity but merely an item on a to-do list. Once an actor element performs a *PutOnConveyorOperation* on the work piece, a physical counterpart (called a conveyor item) is created and the work piece alternates between the ‘working’, ‘moving’, and ‘waiting’ states. When it has completed all operations in its process (most likely ending with a *StoreOperation*), it enters the ‘finished’ state (Figure 3.12). A simulation run completes successfully when all work pieces have reached the ‘finished’ state. When a work piece cannot continue its process (because it cannot reach an actor element that can continue its process, for example), the simulation run fails.

Successfully completing a simulation run produces a number of key figures that characterize the performance of the layout. *Sam* currently uses



State	Descripton
pending	the work piece has not yet started its manufacturing process
working	an actor element is performing a process operation on the work piece
moving	the work piece is travelling on the conveyor belt
waiting	the work piece is standing still on the conveyor belt without being worked on
finished	the work piece has completed its manufacturing process

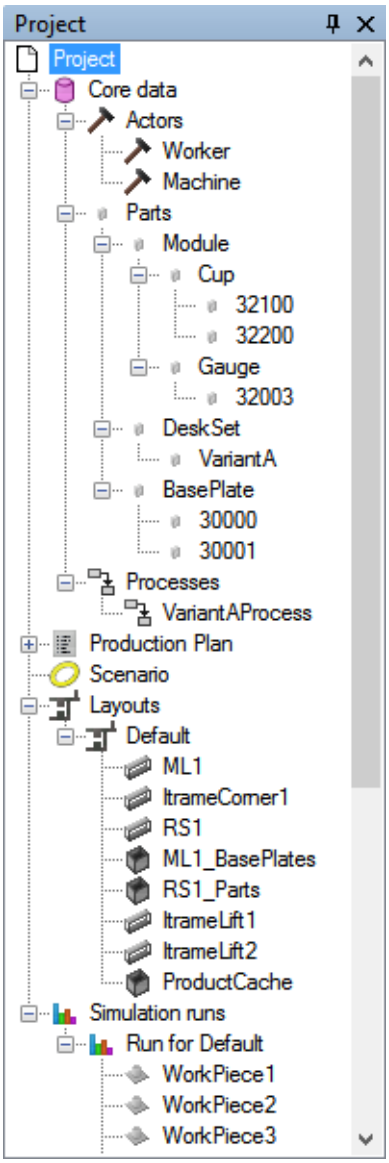
Figure 3.12: The five work piece states.

five different key figures:

- The number of iTRAME elements used by the layout.
- The area of the bounding rectangle of the layout.
- The total time it took this layout to complete the manufacturing task.
- The total cost this layout incurred while completing the manufacturing task according to the cost function defined in the project scenario.
- The average actor load. The actor load is the portion of the total time that an actor element spent in the ‘workiang’ state (as opposed to the ‘idle’ state, in which it waits for the next work piece to arrive).

3.5 The visual interface

The visual analytics approach combines automatic data analysis and interactive visual interfaces. *Sam*’s visual interface is its graphical user



► **Figure 3.13:** The project view displaying a project with the core data shown in Figure 3.8, and Figure 3.9, Figure 3.10, a layout, and a simulation run.

interface, which enables the user to perform all tasks required for planning and evaluating iTRAME layouts:

- managing analysis projects (defining the core data, a production plan, and a scenario),
- designing iTRAME layouts,
- evaluating and comparing layouts, and
- finding new, relevant layouts with automatic support.

Sam's user interface is made up of a number different views that can be arranged as the user sees fit. Available views are the **project view**, the

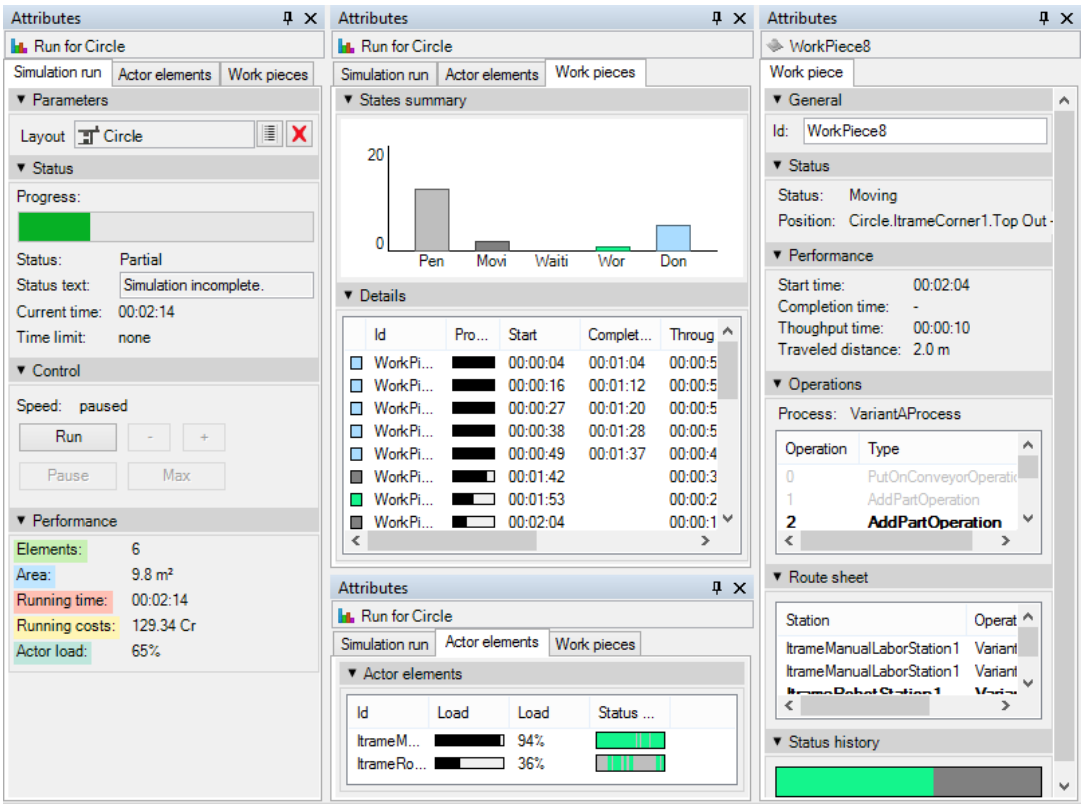


Figure 3.14: Four attribute views showing details on a simulation run (left), the work pieces (centre top) and actor elements (center bottom) of that simulation run, and a single work piece (right).

attribute view, the **layout view**, the **layout comparison view**, the **layout discovery view**, and the **discovery highlights view**. The project view shows the hierarchy of all elements in the current project. This includes the core data, production plan, and scenario definition as well as any layouts the user has designed and simulation runs that have been performed.

3.5.1 The attribute view

The attribute view shows the attributes or properties of an element. Multiple attribute views can be opened to show attributes of different objects or different aspects of the same object. Some attributes are purely informational, others can be modified from the view. For example, when viewing a manual labour station element, its actor and port connections can be set.

When displaying a simulation run, the attribute view shows three pages of information. The first page displays the overall progress and the performance (in terms of the key figures introduced in Section 3.4.5) of the simulation run. It also contains controls to pause the simulation or change

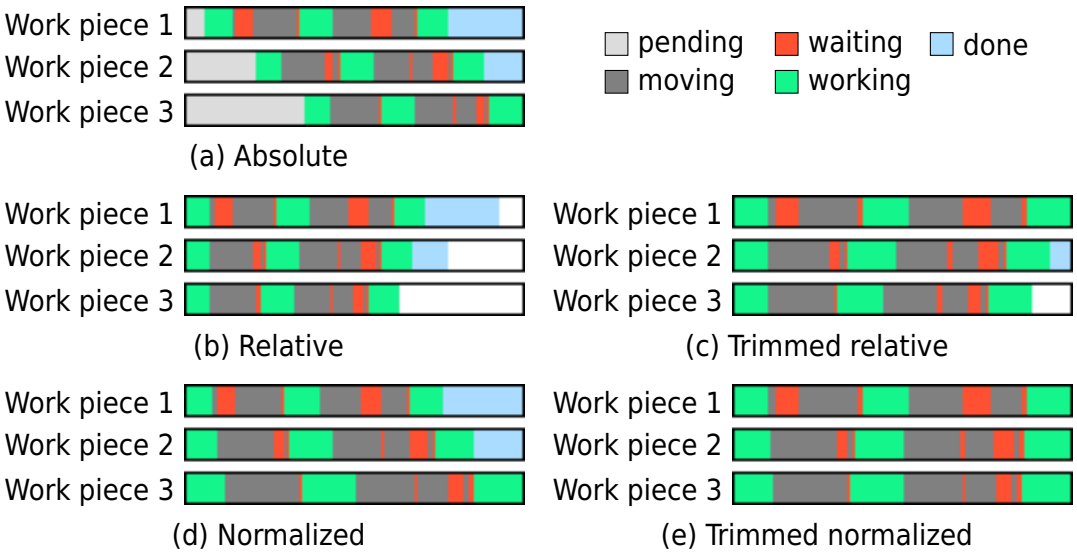


Figure 3.15: Five different representations of the same status history.

its speed. The second page shows an overview of the simulation run's actor elements and their current average load and their status history. The third page contains a list of all work pieces in the simulation run with their status, progress, start time, completion time, throughput time, order, task, product, and status history. This page also shows a bar chart of work pieces grouped by their current status ('pending', 'moving', 'waiting', 'working', 'done').

When displaying a work piece, the attribute view shows the status and performance of the work piece, its list of operations, route sheet, and status history. The list of operations contains all operations of the work piece's process and marks them as completed, executable, and pending. An operation that has not yet been completed is executable if all its dependencies are met and pending if that is not yet the case. The route sheet lists which actor elements have executed the completed operations and which actor elements are already scheduled to perform executable or pending operations.

Status history

The simulation records all state changes of work pieces and actor elements in what it calls a status history. Status histories are visualized in the form of coloured bars (Figure 3.15). Each coloured section of a bar represents a time period in which a work piece or actor object was in a certain state. These bars can be scaled in different ways:

In **absolute** mode (Figure 3.15 (a)), the bar shows the entire status history from the start of the simulation to the current simulation time. In Figure 3.15, work on work piece 3 started after about a third of the total simulation time as indicated by the grey ('pending') section in the left part of the third bar. Production of new pieces starts in regular intervals with all work pieces taking about the same time. Absolute mode is useful for comparing the overall development of a process, as it puts each work piece state in relation to the absolute simulation time.

In **relative** mode, status changes are displayed relative to when a work piece first changed its state from 'pending' to another state. Relative mode is useful for comparing the life cycles of different work pieces and can illustrate increases in throughput or waiting times more clearly than the absolute display, where work piece life cycles are not generally aligned. Figure 3.15 (b) shows that in this example, the throughput time slightly decreases from work piece 1 to work piece 3. By default, the scale of the relative display is the same as in absolute mode. As an alternative, **trimmed relative** mode scales the display relative to the longest work piece life cycle, not including time spent in the 'done' state (Figure 3.15 (c)).

In **normalized** mode, the display is scaled so that each individual work piece's history (not including time spent 'pending') fills the available area. **Trimmed normalized** mode also strips time time spent in the 'done' state. This gives an impression of the distribution of states relative to the throughput time of the work piece. Figure 3.15 (e) shows that work piece 3 spends more of its throughput time in the 'working' state than work piece 1.

Comparing the status histories of work pieces across a simulation run can provide valuable insight when judging the performance of a layout. Figure 3.16 shows a comparison of two different layouts. The status histories are displayed in absolute mode. The layout on the left processes work pieces in groups of four and throughput times remain stable for the entire duration of the process. In the layout on the right, pairs of work pieces are started simultaneously and there is a considerable build-up of waiting times as the process progresses.

3.5.2 The layout view

The layout view displays a visual representation of a layout. It can be used examine the spatial arrangement and connectivity of a given layout, to modify layouts, and to create new layouts. It can also show a simulation run, displaying the dynamic state of a layout with work pieces moving about, switches changing their positions, and lift elements transporting work pieces between the conveyor belt layers. This can be used to watch a simulation run in real-time (or fast-forward or slow motion) to better

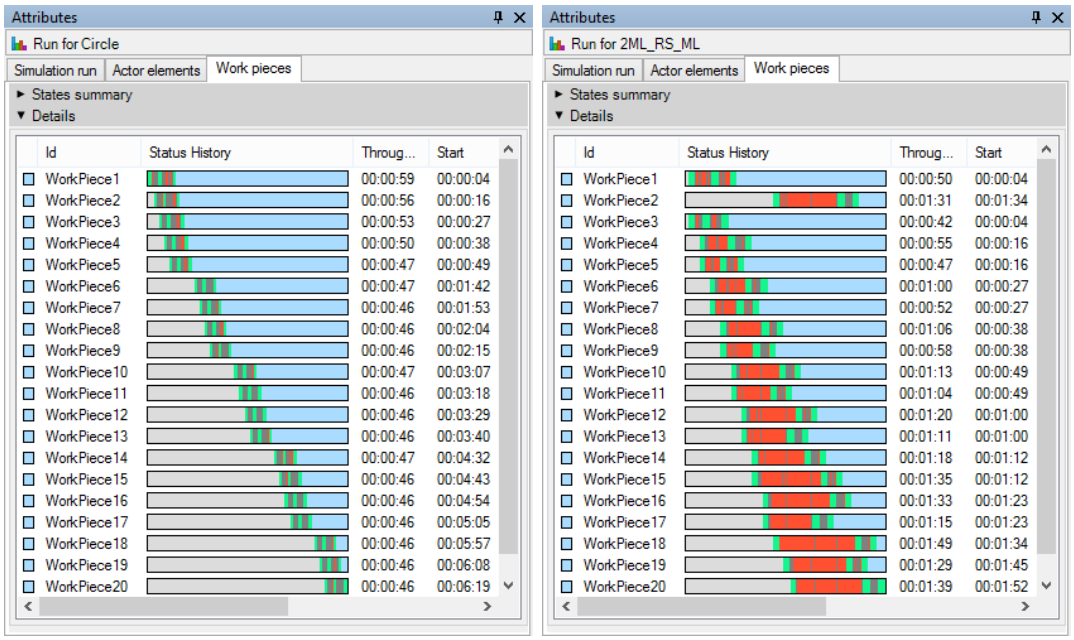


Figure 3.16: Two status history comparisons.

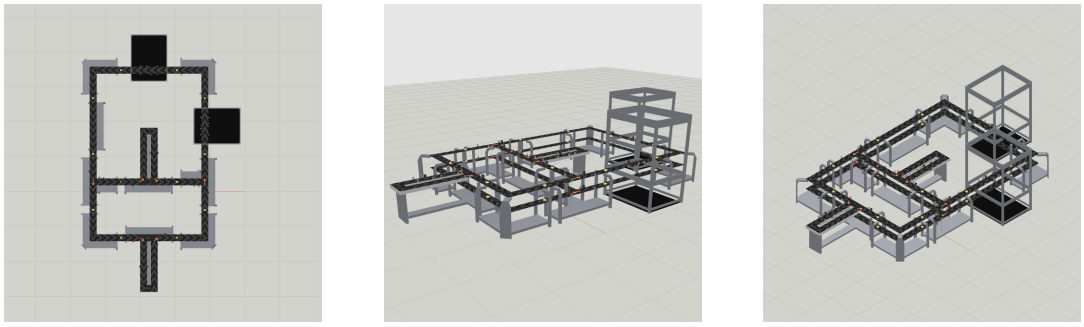


Figure 3.17: The top-down, perspective, and isometric projections.

understand the operation of the corresponding layout. Multiple layout views can be opened, for instance, to watch two simulations side-by-side or to have multiple views of the same simulation.

The user can switch a layout view between three camera projections:

- The top-down projection (Figure 3.17 left) is a parallel projection looking down onto the layout. It clearly shows the spatial layout of elements and prevents any overlap between different elements. However, only the top level of the conveyor belt is visible, as it completely occludes the lower level and any work pieces on it. The top-down projection supports panning, zooming, and rotating around the top-down axis.

- The perspective projection (Figure 3.17 centre) creates a realistic rendering of the scene with objects appearing smaller the further away they are. The user can freely move and rotate the virtual camera to show any detail of the layout. With an appropriately positioned camera, work pieces on both layers can be seen. The drawbacks of this projection are that the amount of element occlusion is highly dependent on the view angle and that it is not as easy to compare lengths and distances across the three-dimensional view. The perspective projection supports moving and rotating the camera in three dimensions.

- The isometric projection (Figure 3.17 right) is a parallel projection that (unlike the top-down-projection) projects the scene onto a plane that is not orthogonal to any of the coordinate axes. Instead, the isometric projection plane is chosen in a way that results in the scaling factors along all three axes being equal (hence the name). Since the scaling factors do not depend on the distance from the virtual camera, lengths and distances can be compared across the entire display. Also, while there is some occlusion in the isometric view, both the upper and lower conveyor belt are visible in most cases. The isometric projection supports panning, zooming, and rotating the view around the vertical axis in 90° steps.

Elements in the layout view can be moved by dragging them with the mouse. The location of the iTRAME ports is indicated by small coloured dots on the element. When one of the unconnected iTRAME ports of an element that is being moved comes near an unconnected iTRAME port of another element, it will snap into place. Upon releasing the mouse button, the two elements will be connected. This is indicated by a change in the colour of the iTRAME port. Once connected, elements can be moved as a group. Clicking an element in the layout view shows its properties in the attributes view.

Figure 3.18 shows a close-up view of a switch element. A switch element has three iTRAME ports (with two conveyor connection points each). One of this element's ports is connected to a neighbouring element, as indicated by its yellow colour. The other two ports are orange, marking them as unconnected. The texture on the conveyor belt indicates the direction in which the belt is moving. The user can reverse the belt direction of an element, which will automatically adjust the belt directions of all connected elements. For switch elements, the branch direction can be set independently. The branch direction is the belt direction of the short orthogonal segment. If it moves towards the centre of the switch, the switch joins

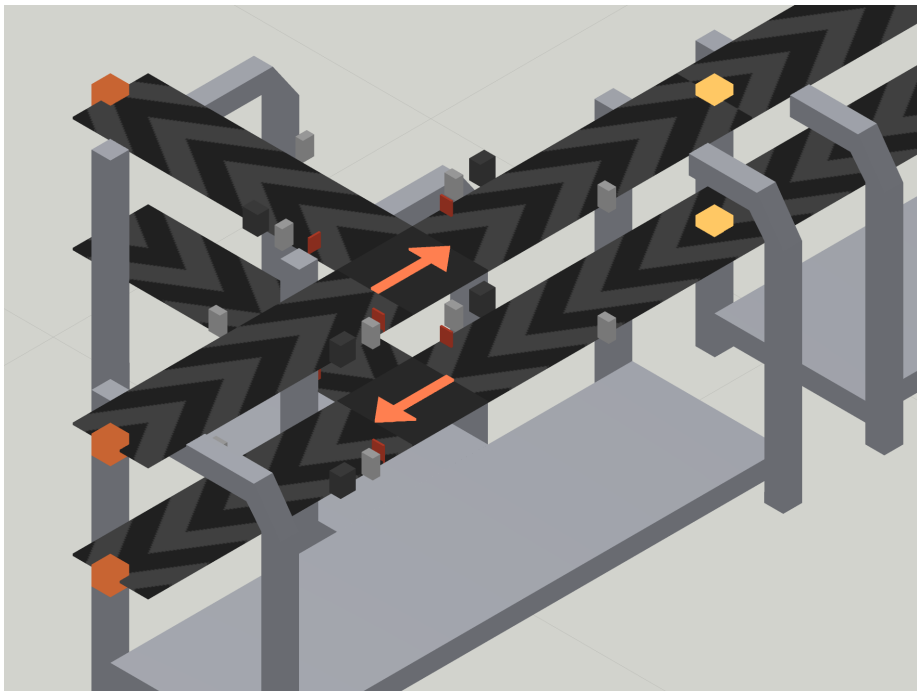


Figure 3.18: A close-up view of a switch element with one connected iTRAME port (yellow) and two unconnected ports (orange). The texture of the conveyor belt shows the movement direction of the belt. The orange arrows show the current position of the switch.

two segments of the conveyor belt. If it moves away from the centre (as in Figure 3.18), the switch splits the belt into two segments. Since the lower and upper belt always move in opposite directions, a switch that splits on its upper level will join on its lower and vice versa.

The orange arrows in the centre of the switch show the current position of the switch. In Figure 3.18, the arrows point straight ahead, so a work piece travelling across this switch would continue on to the neighbouring element in the top right of the image. The switch position is part of the state of the switch element and is only displayed when the layout view shows a simulation run (and not just a layout).

When displaying a simulation run, the layout view also shows the average work piece density as a colour map on the conveyor belt. Conveyor sections with a very low work piece density, such as those sections never used during a simulation run, remain dark grey. Medium work piece density is shown in blue and the most frequented sections are highlighted in red. In Figure 3.19, the work position of the two manual labour stations (top centre) is red, which is to be expected, as the manual labour stations continuously place work piece on the conveyor, add modules, or remove

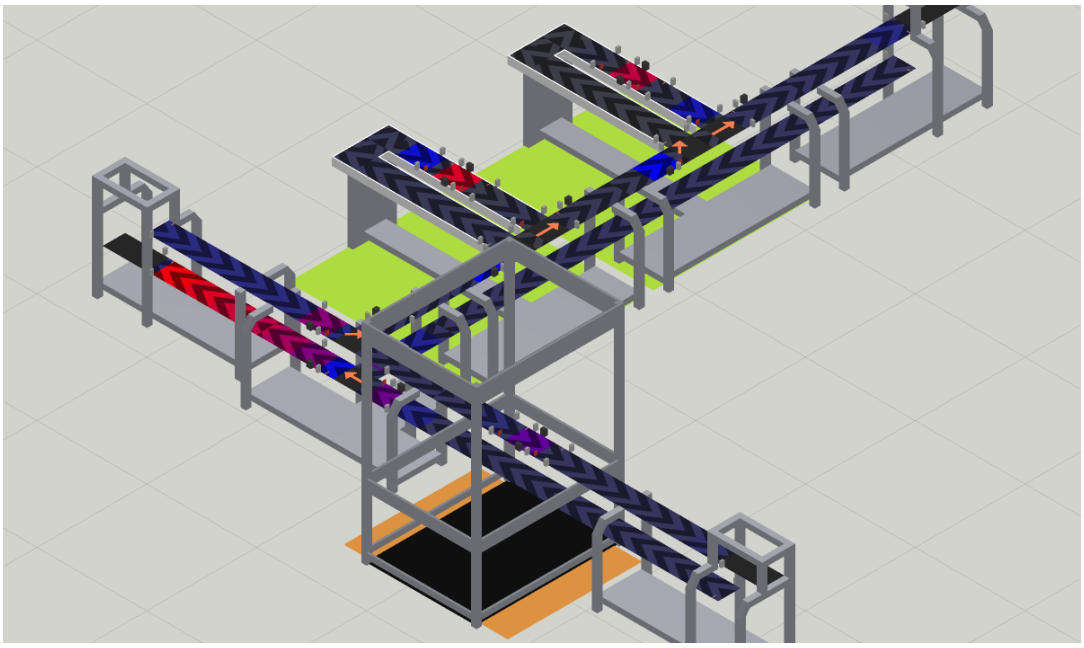


Figure 3.19: When displaying a simulation run, the layout view shows the average work piece density by colouring the conveyor belt. A coloured rectangle beneath each actor element indicates its load.

work pieces and store them. The lack of a red trail in front of these work position indicates that there is no significant build-up of work pieces waiting to be processed by either manual labour station. The lift element on the left, however, creates a pronounced red highlight on the conveyor section leading up to it. This indicates a significant work piece congestion on this conveyor section. Coloured rectangles beneath each actor element show the load of this element on a red–yellow–green scale. In Figure 3.19, the robot station at the bottom is highlighted in orange, indicating a less than ideal load of this actor element. The exact load values can be found in the attribute view for this simulation run.

Side note: Technical realization

The layout view renders a scene using the DirectX programming interface and thus benefits from graphics hardware acceleration. The layout is converted into a scene of triangles, which are transformed, projected, and finally drawn into the view by a set of ‘shaders’, small pieces of code executed on the graphics hardware. *Sam* does not aim for a highly complex and realistic visualization of the scene and uses the Blinn–Phong reflection model (Blinn, 1977) with a single, direc-

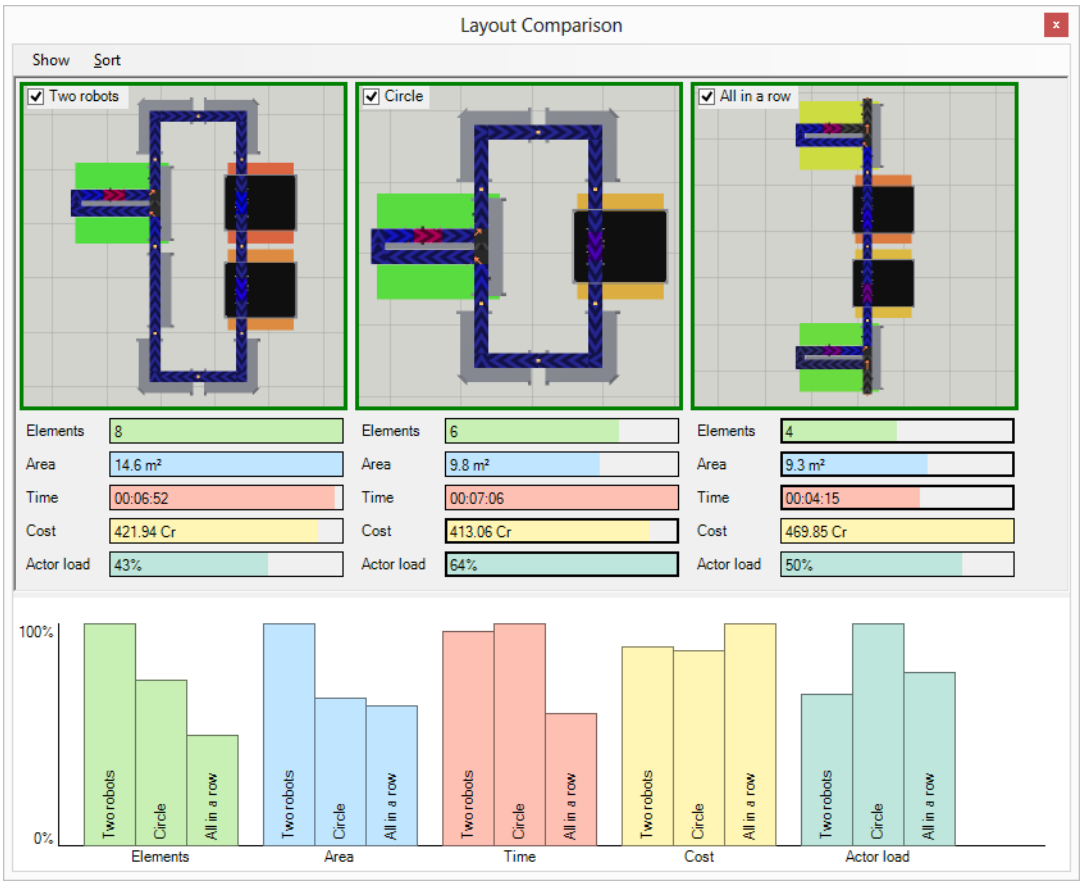


Figure 3.20: The layout comparison view.

tional light source and Gouraud shading (Gouraud, 1971). It supports different materials to set objects apart, but these are expressed only in terms of ambient, diffuse, and specular colour, as well as a specular component ('shininess'). As an exception, the triangles representing the conveyor belts are rendered using a texture map that indicates the direction of the belt.

3.5.3 Layout comparison

Promising layout candidates can be collected and compared in the layout comparison view (Figure 3.20). This is where the user creates and verifies a hypothesis on what might be a good layout to use. The top part of the view displays the layouts that have been simulated so far. Layouts with a green border have successfully completed the manufacturing task, whereas layouts with a red border have failed their task. The specific failure reason can be found in the attribute view of the corresponding simulation run. The space below the layout image lists the values of the key figures as

determined by simulating the layout. They characterize the performance of this layout in comparison to the other layouts. These are the five key figures defined in Section 3.4.5: number of elements (green), layout area (blue), total time (red), total cost (yellow), and average actor load (turquoise). Optionally, a score value (purple) provides a weighted combination of the other key figures. How the individual key figures are weighted to produce the score value is determined by the user in a separate window using slider controls. The result is immediately visible in the layout comparison view. The coloured bars visualize the key figures values in relation to the other layouts in the view. The highest value of all views is displayed as a full bar. Lower values produce proportionally shorter bars. For each key figure, the best value of all layouts displayed is drawn in bold, so one can easily recognize which layouts excel in which aspects. Lower values are better for number of elements, area, time, and cost. Higher values are better for actor load and score. The lower part of the view shows a bar chart of all selected key figures grouped by layout and, as above, normalized to the overall maximum value. In this view, a user can compare the advantages and disadvantages of different layouts. The user can configure the display to show only selected key figures or sort layouts according to a given key figure. At any time, a layout can be discarded and removed from the view. The upper and lower part of the view are linked in that hovering over a layout in the upper part will highlight the corresponding bars in the lower part and vice versa.

3.5.4 Layout discovery and discovery highlights

Sam supports the user's search for a good layout by automatically generating and evaluating layout variants in the background. This process can be watched and controlled from the layout discovery view (Figure 3.21 top). The top part of the view shows the current search priorities. The user can use the sliders to define an importance value for each key figure. The background process can be started and paused with the two buttons to the left of the sliders.

The bottom part of the view lists all layouts currently considered by the automatic discovery with the best layout at the top of the list. Each row shows a status indicator (pending, simulating, simulation failed, simulation succeeded), a name (automatically generated layouts are identified by consecutive numbers), the simulation progress, a score value (resulting from the key figures of the layout and the search priorities), and the key figure values for the layout. The layout discovery uses an evolutionary algorithm (see section Section 3.8 for details) and additional columns identify the layout's ancestor (i.e. the original layout that was the basis for this variant)

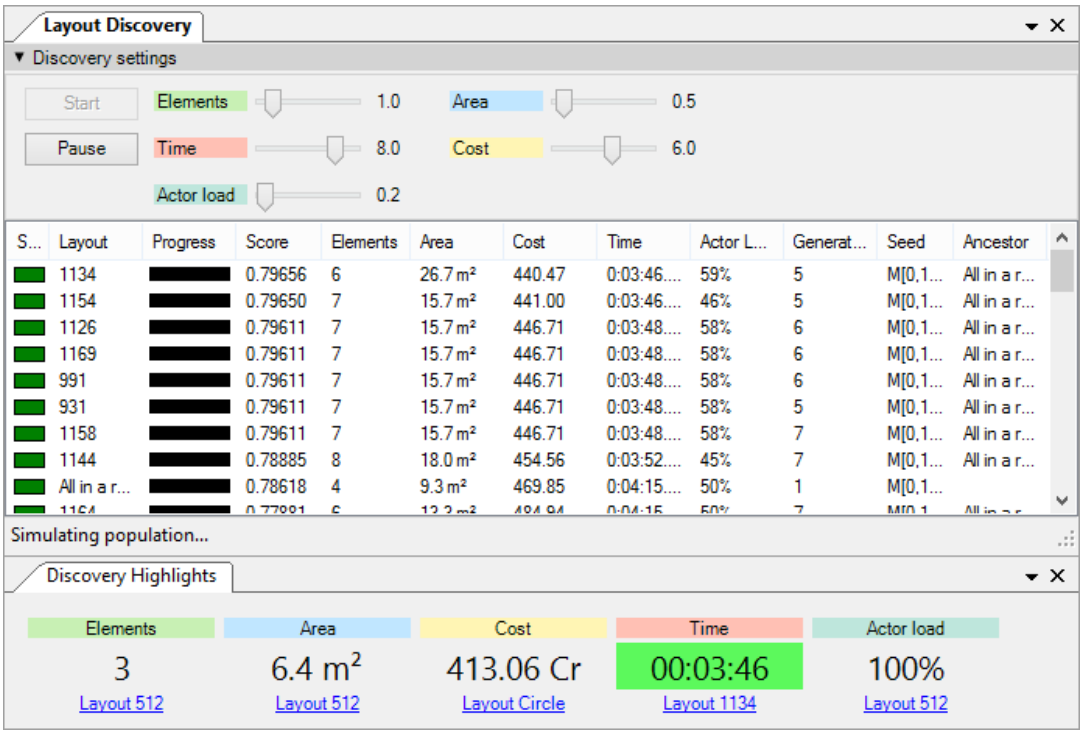


Figure 3.21: The layout discovery and the discovery highlights view.

and generation (i.e. how many modification steps were performed before arriving at this layout). The ‘Seed’ column lists the layout’s ‘genetic code’, which is mostly informative.

The discovery highlight view (Figure 3.21 bottom) gives a short summary of the discovery process by displaying for each key figure the best value of all layouts currently considered. When an improvement is made for one of the values, it is highlighted in green (such as the ‘Time’ value in Figure 3.21). The highlight fades away over time, so just by glancing at this view, the user can tell whether the automatic discovery has recently found an improvement. This makes the discovery highlight view suitable for being docked at the side of the main window to provide a progress indication in the user’s peripheral vision. Clicking one of the layout names below one of the key figures opens the corresponding simulation run for closer inspection.

3.6 Case study

This section demonstrates how a visual analysis of iTRAME layouts using *Sam* might progress by examining a specific scenario, detailing the individual steps, and presenting the results. These results were obtained in

an actual session with *Sam* (albeit using synthetic data), so even though a repetition of the process may not yield the exact same results (due to the probabilistic nature of the automatic layout discovery, see Section 3.8 for details), this case study is representative of how the work and knowledge of a human analyst and the automatic analysis can complement each other.

The case study uses the core data described in Section 3.4.1 and the cost function given in Section 3.4.3. For this example, an analyst is looking for a good layout to manufacture 20 items of the 'VariantA' product (as defined in Figure 3.8) using the process from Figure 3.9. This 'good' layout will have to be a compromise between time, cost, and other considerations.

Before designing a first layout, the analyst enables the automatic layout discovery and configures it to search for layouts using the weight factors $w(\textit{Number of elements}) = 1.0$, $w(\textit{Area}) = 0.5$, $w(\textit{Time}) = 8.0$, $w(\textit{Cost}) = 6.0$, and $w(\textit{Actor load}) = 0.2$.

3.6.1 Four initial layouts

According to the core data used, only the 'Worker' actor for the manual labour stations can place work pieces on the conveyor belt or remove them from there. Consequently, the first and final steps of the process will have to be performed by a manual labour station. The simplest iTRAME layout that can successfully complete the process, is a single manual labour station. The analyst starts by creating this layout, calling it 'ML only' (Figure 3.22 top). Manufacturing 20 items on this layout takes about 12 minutes and costs 611 Cr.

Robot stations are both faster and more cost-efficient than the manual labour station, so the first attempt to improve this layout is adding a robot station and have it perform the operations '2', '3', and '4' (adding the modules to the base plate). Two lift elements create a way for work pieces to travel back to the manual labour station to be stored. Figure 3.22 centre shows this second layout, 'ML and RS'. It reduces the time to 7 minutes and the costs to 435 Cr while obviously increasing the number of elements and area. The yellow highlight around the robot station in the layout view indicates that it is not utilized to its full capacity. The 'Actor elements' tab shows a load of 93 % for the manual labour station but only 35 % for the robot station.

Instead of the lifts, the analyst could have used corner elements to lead the work pieces back to the manual labour station, so the third layout, 'Circle' connects the manual labour station and the robot station with a circular conveyor belt (Figure 3.22 bottom). Simulating this layout reveals that this change has little effect on the time. It slightly reduces the costs (due to the lower operating cost of the corner elements) while increasing

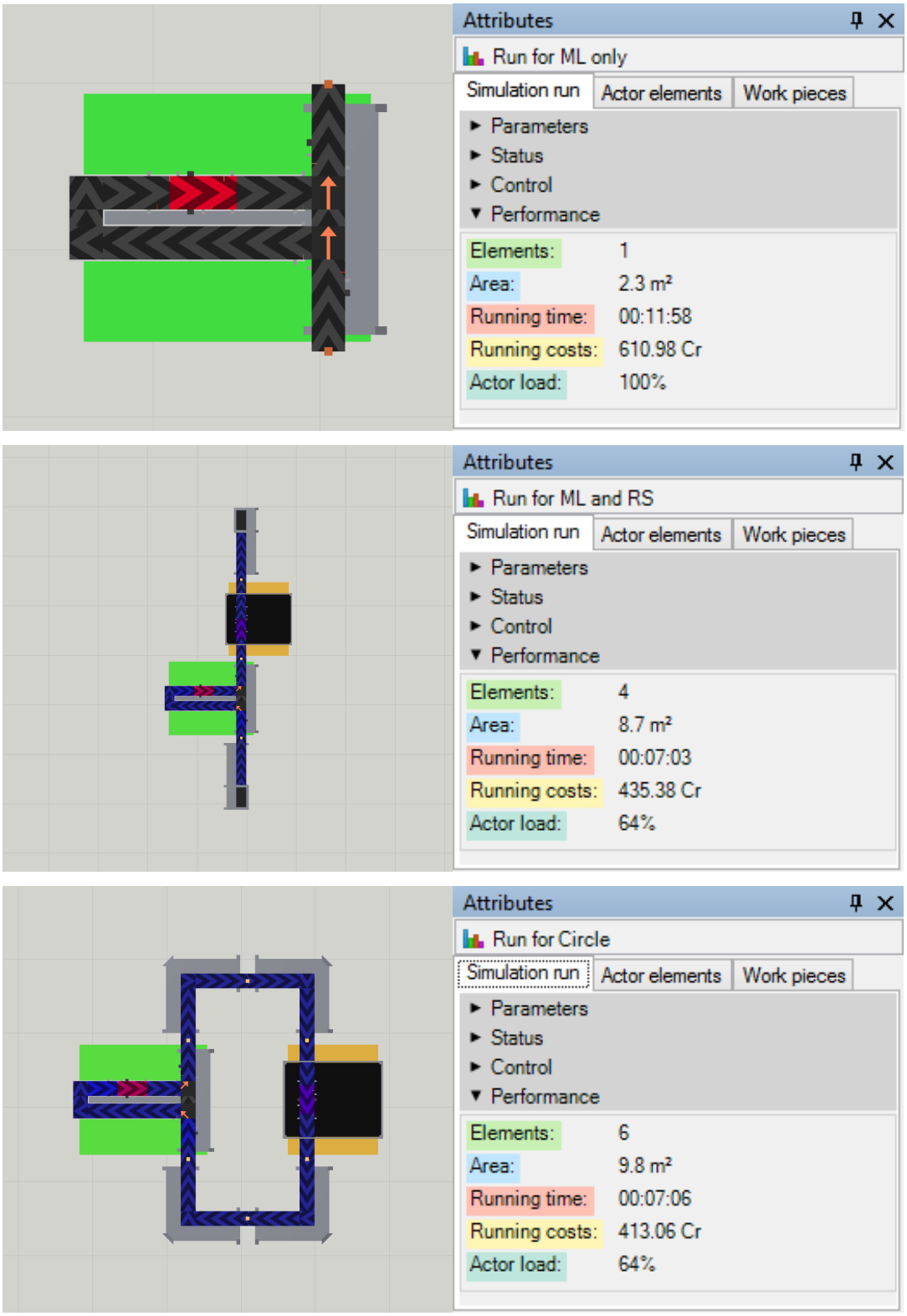


Figure 3.22: The ‘ML only’, ‘ML and RS’, and ‘Circle’ layouts and their performance.

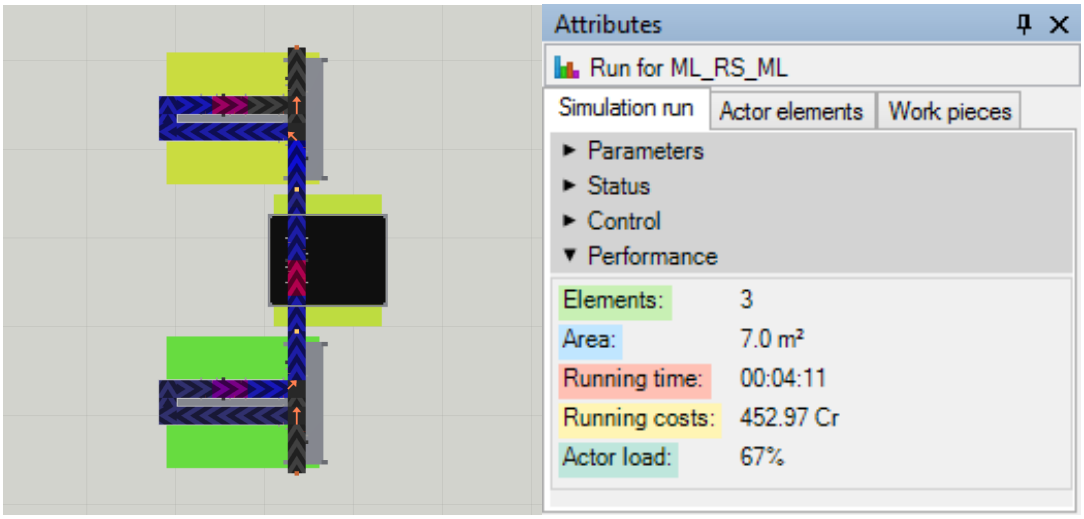


Figure 3.23: The ‘ML_RS_ML’ layout.

the total number of elements and the covered area. The actor load of the robot station is unchanged.

The analyst now considers adding a second manual labour station that performs the store operation. This way, work pieces need not travel back to the original manual labour station. Also, if the first worker does not have to perform both tasks of putting items on the conveyor and removing the finished products, this may reduce the idle time of the robot station. The resulting layout ‘ML_RS_ML’ can be seen in Figure 3.23. This arrangement greatly reduces the required time from 7 to a little over 4 minutes while its costs are not much higher than those of the ‘ML and RS’ and ‘Circle’ layouts. Additionally, the actor load of the robot station increased from 35 % to 59 %.

After designing these four initial layouts, the analyst opens the layout comparison view to compare the layouts examined so far (Figure 3.24). Applying the same weight factors as used for the automatic discovery confirms that ‘ML_RS_ML’ is the best layout, while ‘ML and RS’ and ‘Circle’ are about equal. ‘ML only’ is worst in both time and costs, so the analyst discards this layout by removing it from the layout comparison. Since ‘ML and RS’ and ‘Circle’ are so similar, ‘ML and RS’ is also removed, because the analyst feels the three second difference in time does not make up for the 20 Cr difference in cost.

3.6.2 Integrating automatic suggestions

Before moving on with the two candidate layouts, the analyst takes a look at the discovery highlights view (Figure 3.25). The automatic discovery has not

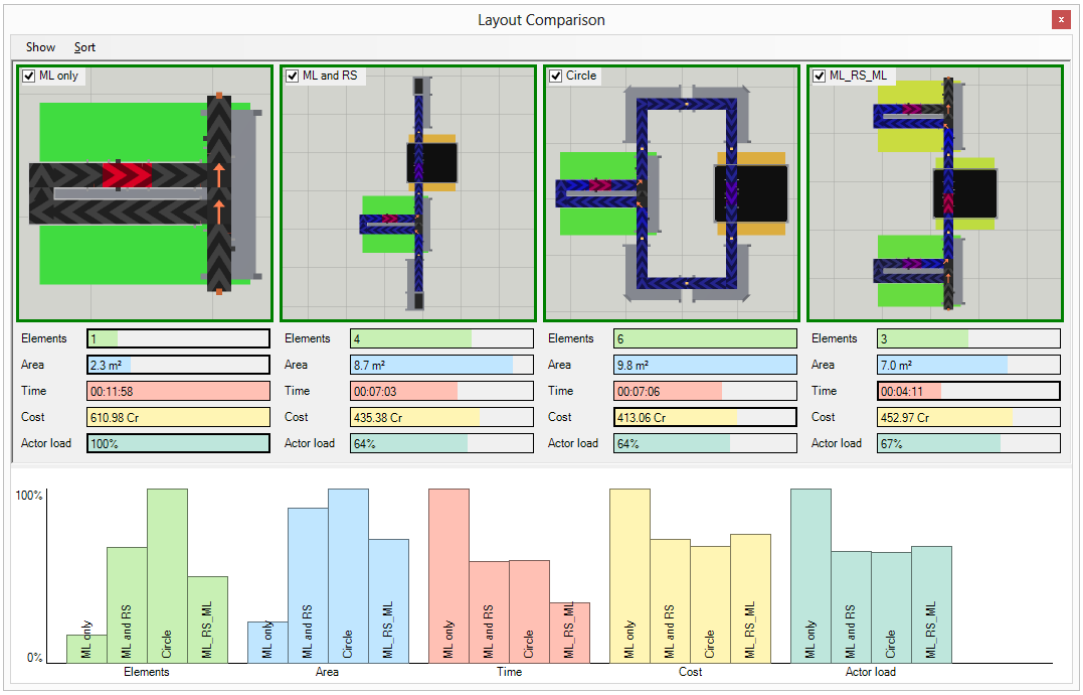


Figure 3.24: The layout comparison view comparing the four initial layouts.

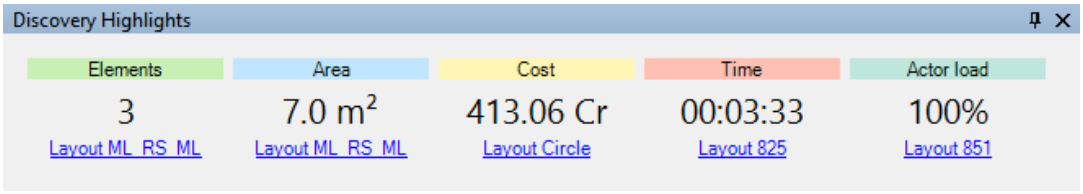


Figure 3.25: The discovery highlights view indicates that the automatic discovery has found a layout that completes the task in less time than any of the layouts considered so far.

found a layout that is more cost efficient than ‘Circle’ so far, but its layout ‘825’ (Figure 3.26 top) completes the task in just 3:33 minutes, significantly less than the current best value of 4:11 minutes for the ‘ML_RS_ML’ layout. The analyst opens this layout in the layout view and discovers that it extends the ‘ML_RS_ML’ layout with another manual labour station. This seems to improve the actor load of the robot station and is indeed faster than the layout designed by the analyst, but it is also more expensive. Also, the analyst immediately spots several flaws in this layout: All lift elements are unused and thus superfluous, as is the switch element, which routes all work pieces from the bottom manual labour station to the centre one. Reviewing the simulation in real-time also reveals that the bottom manual labour station only places the carrier plates and sends these to the centre

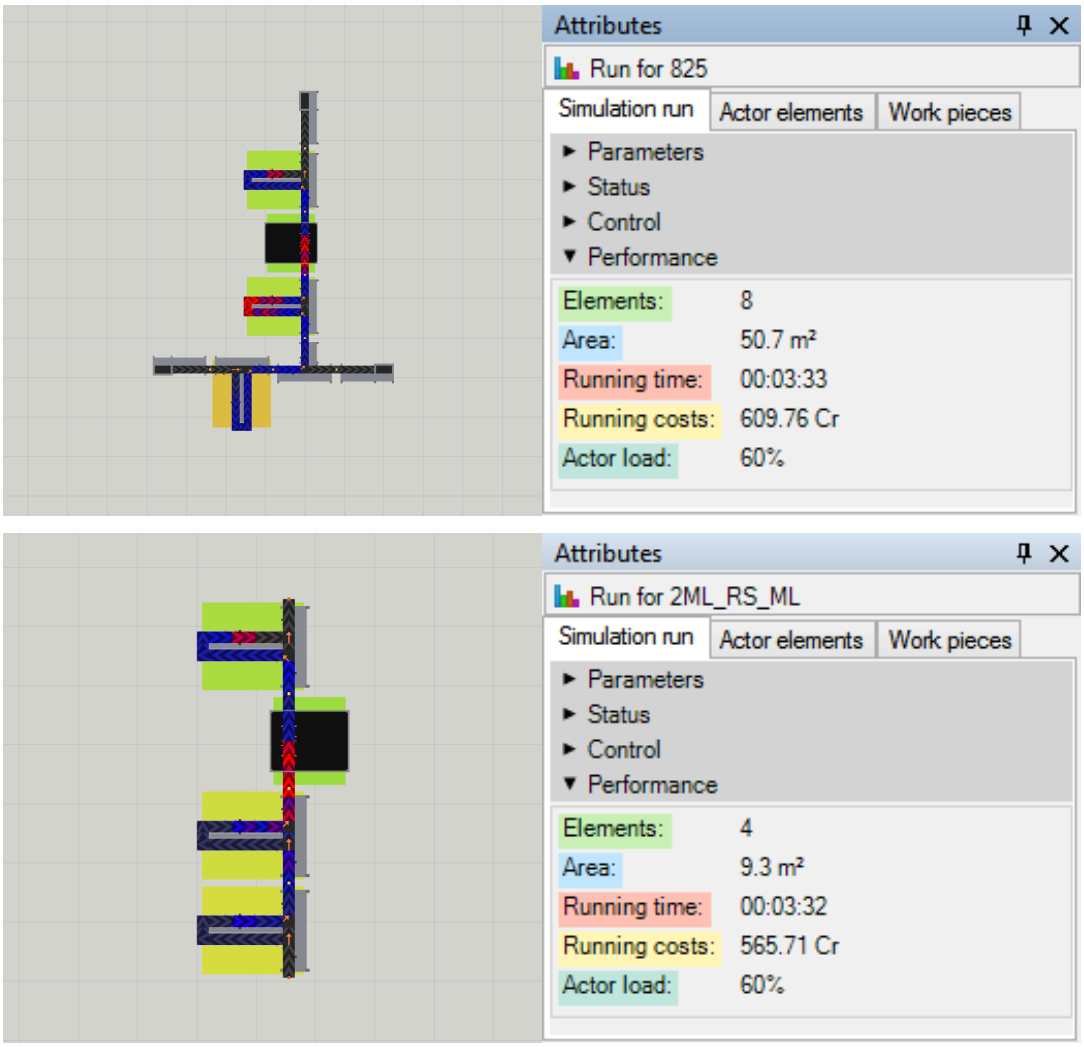


Figure 3.26: Top: The '825' layout, an automatically generated variation of 'ML_RS_ML', which is faster than the original, but contains some obvious flaws. Bottom: The '2ML_RS_ML' layout, which corrects these flaws.

station for adding the base plates. The analyst decides to correct these flaws to see whether without them, the layout achieves a competitive cost value. The resulting layout ‘2ML_RS_ML’ (Figure 3.26 bottom) improves every aspect of ‘825’ but, interestingly, the running time is only marginally better. The red conveyor highlights in the layout view indicate that this may be because the robot station is now the bottleneck in the process.

Adding ‘2ML_RS_ML’ to the layout comparison view reveals that it is currently the fastest layout, while ‘Circle’ is still the one with the lowest costs. ‘ML_RS_ML’ remains the best overall layout according to its score value. In an effort to improve the efficiency of layout ‘2ML_RS_ML’, the analyst now has the bottom manual labour station not only place the carrier plate and the base plate but also add two of the modules. This lessens the build-up in front of the robot station and improves the time to 3:23 minutes. However, ‘Circle’ is still cheaper and ‘ML_RS_ML’ still has a better overall score. Figure 3.27 shows the layout comparison view comparing ‘Circle’, ‘ML_RS_ML’, ‘2ML_RS_ML’, and the new variant ‘2ML_RS_MLb’.

The analyst likes the speed of ‘2ML_RS_MLb’ but is concerned about its costs, especially given the alternative of ‘ML_RS_ML’, which saves almost 100 Cr. It is also true, however, that ‘ML_RS_ML’ takes almost one and a quarter of the time ‘2ML_RS_MLb’ needs. While the analyst ponders the situation, the automatic layout discovery reports finding a noteworthy variant of ‘2ML_RS_MLb’. The layout ‘2852’, while neither the quickest nor the least expensive, is faster than ‘ML_RS_ML’ and less expensive than ‘2ML_RS_MLb’. In fact, at 436 Cr, it is even less expensive than ‘ML_RS_ML’. ‘2852’ might be a good compromise layout and indeed, the layout comparison view confirms that even though it covers the largest area yet, its score value is slightly higher than that of ‘ML_RS_ML’, making it the best overall layout so far.

The analyst opens ‘2852’ in the layout view to examine how it achieves these improvements (Figure 3.28). The third manual labour station has been removed and ‘2852’ looks a lot like ‘ML_RS_ML’ with added lift elements at the top and bottom. However, it inherited the idea to have the manual labour stations place some of the modules from ‘2ML_RS_MLb’. In ‘2852’, both manual labour stations place carrier and base plates and the top station also places two of the modules. It then sends its work pieces over the lifts to the robot station, which adds the third module. For work pieces arriving from the bottom manual labour station, the robot station adds all the modules. Finally, all work pieces arrive at the top manual labour station to be stored. This is a rather effective way of sharing the work, as is reflected in the average actor load value of 78 %, exceeding both the 67 % of ‘ML_RS_ML’ and the 68 % of ‘2ML_RS_MLb’. Satisfied with these results, the analyst chooses ‘2852’ as the layout to be implemented for the process.

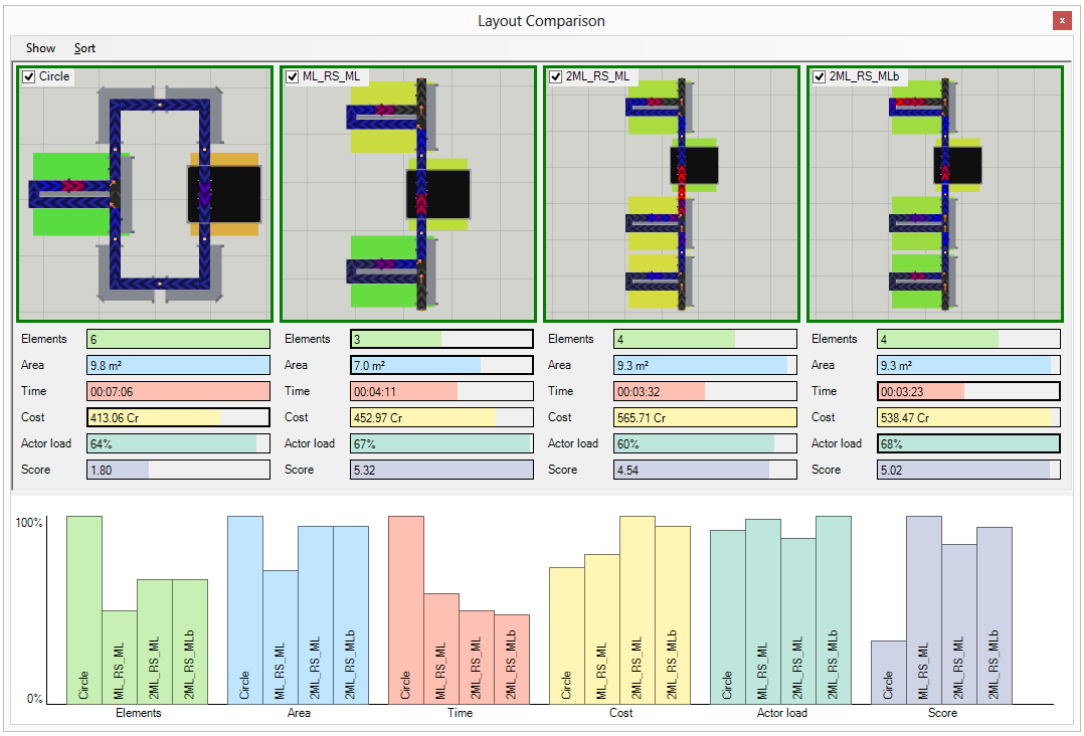


Figure 3.27: The layout comparison view comparing the four initial layouts.

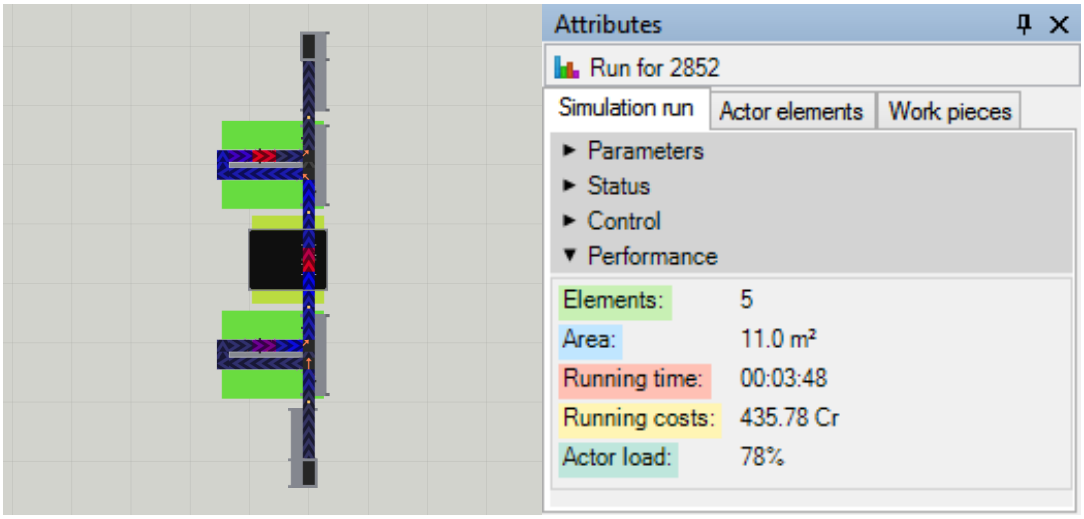


Figure 3.28: Layout ‘2852’ is the layout ultimately chosen by the analyst. It is an automatically generated variant of the analyst’s ‘2ML_RS_MLb’ layout.

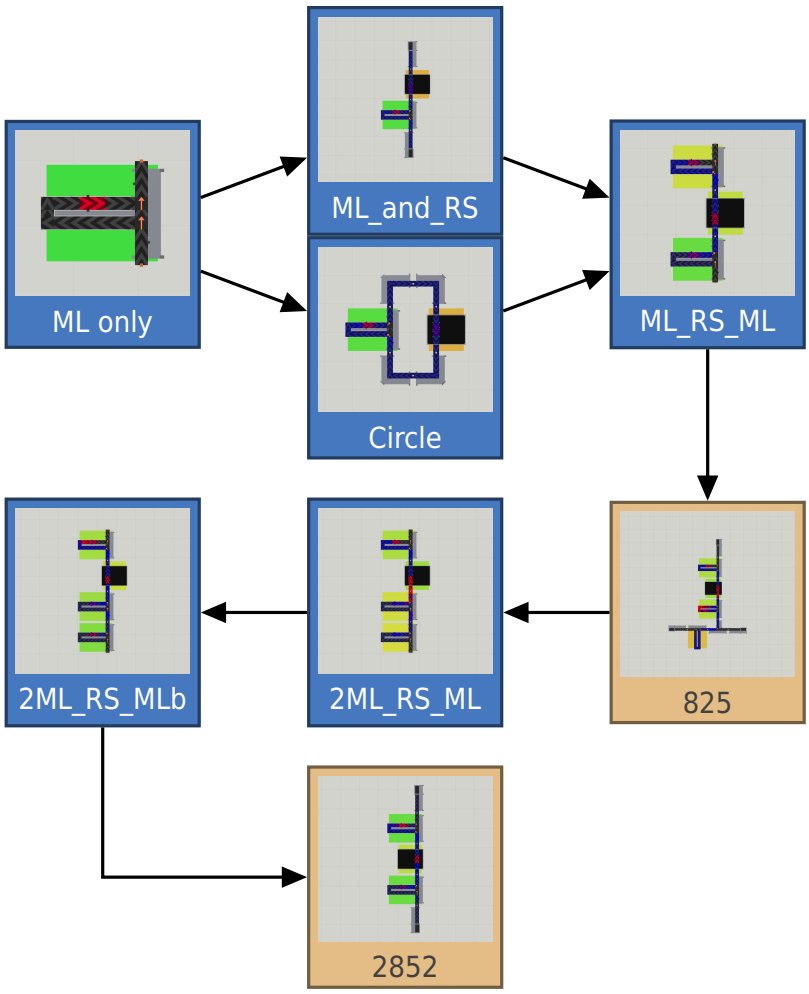


Figure 3.29: Layouts designed manually (blue) and found by the automatic discovery (brown) in the course of the case study. Arrows visualize the logical flow of thought.

3.6.3 Conclusion

This case study demonstrated how *Sam* can be used to design process layouts with support from visual analyses, automatic candidate evaluation, and automatic candidate suggestion. While the data is synthetic, the analytic process is a concrete example of the implict data space exploration described in Section 2.5 and Figure 2.5 and of solving an optimization problem with visual analytics as described in Section 2.6 and Figure 2.6. In the course of the case study, the analyst first created a set of initial layout candidates (‘ML only’, ‘ML_and_RS’, ‘Circle’, ‘ML_RS_ML’), which the automatic discovery used as starting points for an automatic search of the parameter space. One of the results of the automatic sampling, the

layout ‘825’, was used by the analyst to create and evaluate the layout variants ‘2ML_RS_ML’ and ‘2ML_RS_MLb’. These in turn were picked up by the automatic sampling, which found the layout ‘2852’, a variant of ‘2ML_RS_MLb’. ‘2852’ was inspected by the analyst and ultimately chosen as a reasonable solution to the optimization problem at hand. Figure 3.29 summarizes this process.

3.7 Simulating iTRAME processes

Sam supports the analyst in evaluating a given iTRAME layout by simulating it. The simulation produces several key figures as well as a visual and dynamic representation of the layout’s operation. Both can be used to assess the general merit of a layout, its strengths and weaknesses, and potential problems or opportunities for improvement. Simulation runs and their results are the basis for much of the information presented in *Sam*’s various views as described in Section 3.5.

As stated in Section 3.4.4, *Sam* models an iTRAME layout as a connected graph of iTRAME elements and caches. However, this view is much too coarse for a detailed simulation of work piece movements and actor element operations. When the user adds and connects iTRAME elements in the layout view, *Sam* creates a complex arrangement of conveyor segments, conveyor edges, and conveyor objects that model the structure and functionality of these elements in much greater detail.

3.7.1 The conveyor graph

Whenever a work piece is put onto the conveyor belt, *Sam* creates a **conveyor item**, which acts as the physical representation of the work piece while it is on the conveyor. Unlike the work piece (which begins its existence in a non-physical ‘pending’ state), a conveyor item has both a logical position in the conveyor graph and an absolute position in 3D space. A conveyor item travels along the conveyor graph by traversing **conveyor edges**, which start and end at **conveyor ports**. Conveyor ports are similar to iTRAME ports in that they can be connected to another conveyor port to model a physical connection between two conveyor edges. Conveyor ports connected in this way are said to be neighbouring ports. A conveyor item travelling on a conveyor edge e_1 can continue onto another edge e_2 if the start port of e_2 is a neighbour of the end port of e_1. Connecting two iTRAME ports, which contain one conveyor port for each level of the conveyor, results in each conveyor port of one iTRAME port being connected to its partner on the same level in the other iTRAME port.

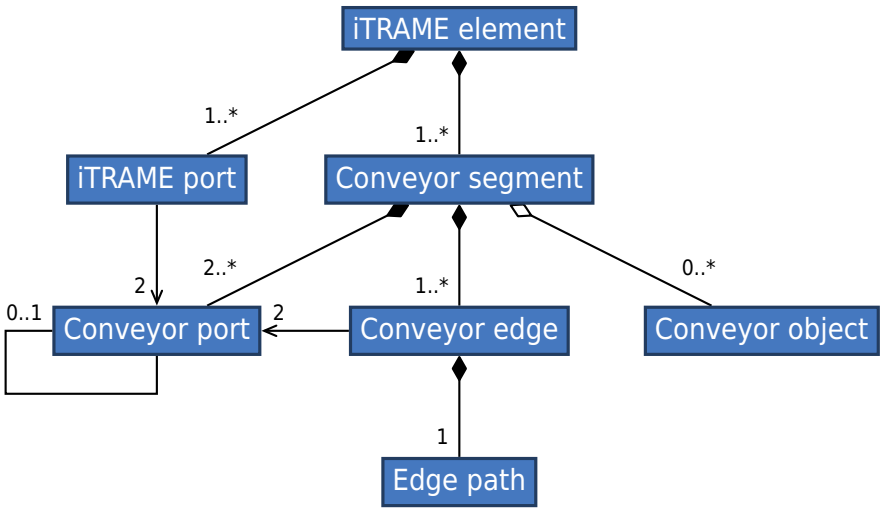


Figure 3.30: The inner structure of an iTRAME element.

Conveyor edges are directed, so there is a clear notion of which is the ‘source’ and which is the ‘destination’ port of the edge. However, every edge is still considered a two-way connection and, depending on the movement direction of the conveyor belt, may be traversed backwards, from destination to source. A conveyor edge defines a **conveyor edge path**, which specifies the geometric shape of the edge as a sequence of linear and arc elements. Conveyor ports and edges are combined into **conveyor segments**, which make up the structure of an iTRAME element (Figure 3.30). In addition to ports and edges, conveyor segments can contain **conveyor objects**, which are located on a conveyor edge and interact with passing work pieces.

Side note: Node–node–edge–node–node

From a mathematical point of view, conveyor ports and conveyor edges do not constitute a proper graph because a graph is defined as a tuple $G = (V, E)$ where V is a set of vertices, E is a set of edges, and an edge is a set of two elements from V. In the conveyor graph, however, edges link ports, but ports can also be connected to each other by being neighbours.

This can be remedied by defining this neighbouring relation to be a special kind of edge and stating that in addition to the conveyor edges, E contains a ‘neighbour edge’ for each pair of neighbouring ports. In graph theory, a graph that differentiates different types of edges is called an edge-coloured graph and the conveyor graph could be defined

as $G = (V, E, f)$ where $f : E \to \{\text{conveyor}, \text{neighbour}\}$ maps an edge to one of these 'colours'. When traversing the graph, conveyor items would then have to follow a 'properly edge-coloured path', in which any two successive edges differ in colour (Abouelaoualim et al., 2008). This requires them to alternate between conveyor and neighbour edges. Since only conveyor edges represent a path in space while neighbour edges merely model transition points, the path may be thought of as consisting of vertex pairs (neighbouring conveyor ports) with conveyor edges connecting these pairs and the traversal rule being that a pair may only be exited via the vertex that was not used to enter it. As a consequence, unpaired, single vertices are dead-ends that cannot be exited.

This view has some merit in the modelling of traffic networks, as it allows an implicit specification of turn restrictions (Figure 3.31) and simplifies the adaption of pathfinding algorithms to respect these restrictions. It has been used for modelling railway track topology, where it is referred to as a double vertex graph (Montigel, 1992). *Sam* does not currently depend on turn restrictions because it assumes the conveyor direction of a conveyor segment does not change during a simulation run making all edges one-way connections (the notable exception being the movable part of lift elements). The port/edge model was chosen because it relates spatial paths to edges (instead of vertices) and allows iTRAME elements to define their inner structure in terms of edge paths and outer end points (ports), to which other iTRAME elements may connect their own end points.

Conveyor objects

Sam simulates iTRAME elements as state machines that interact with work pieces by reacting to events reported by conveyor objects and by invoking operations on or changing the state of conveyor objects. Conveyor objects can be placed on an edge and set to act on work pieces that traverse the edge in forward, backward, or either direction. There are three elementary and three compound conveyor objects (Figure 3.32).

The **stopper** object can be raised or lowered. When raised, it stops work pieces from moving past it, holding them in place even though the conveyor belt itself never stops moving. They are essential in that they keep work pieces from entering occupied stations or stop them in front of a switch that is currently changing its position. In the physical iTRAME system, stoppers are realized by extensible, mechanical bolts.

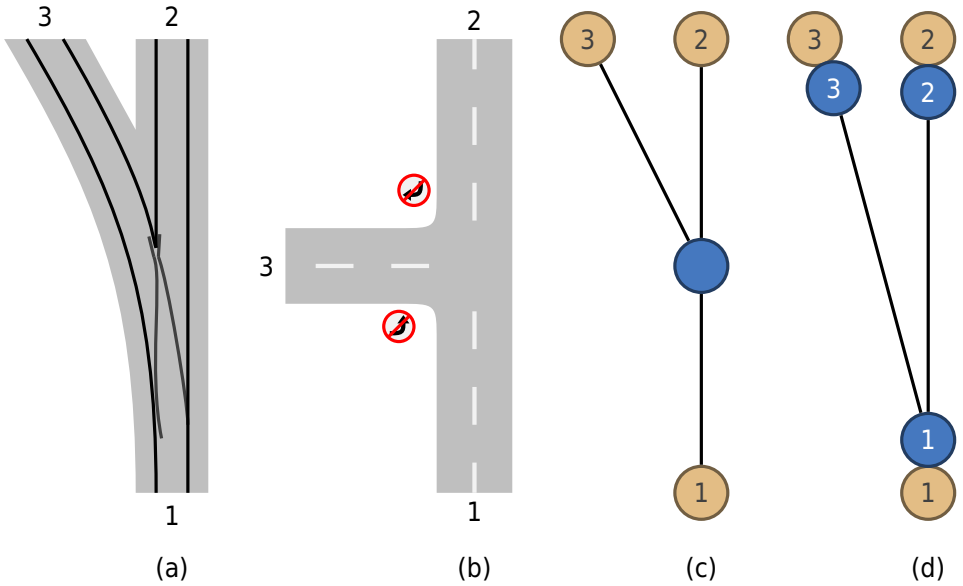


Figure 3.31: A railway switch (a) and a road intersection (b) that allow traversal on $1 \leftrightarrow 2$ and $1 \leftrightarrow 3$ but disallow it on $2 \leftrightarrow 3$. A vertex/edge graph (c) cannot model this restriction while the vertex pair model (d) enforces it via the rule that a vertex pair must be entered and exited on different vertices.

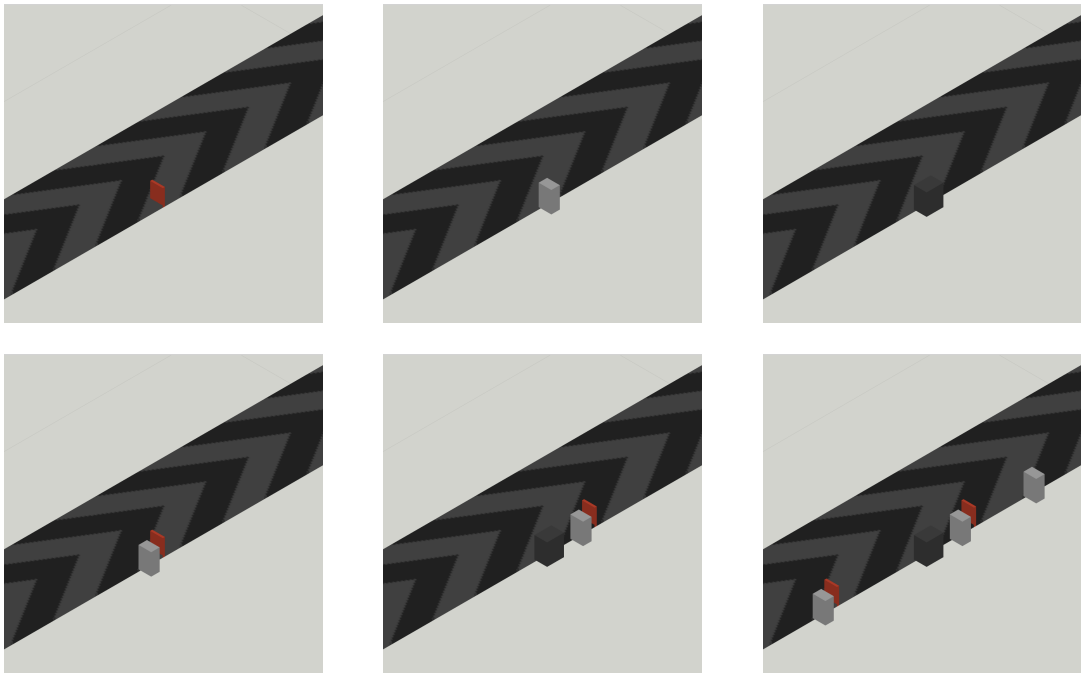


Figure 3.32: Top: The three elementary conveyor objects: stopper, detector, and reader. Bottom: The three compound conveyor objects: detector/stopper, scanner, and single item scanner.

The **detector** object detects passing work pieces. It monitors a specific position on the conveyor belt and reports whether or not this position is currently occupied by a work piece. It cannot, however, determine the identity of that work piece.

The **reader** object identifies work pieces. For a work piece to be identified successfully, its RFID chip must remain near the reader object for a certain time. A reader object will therefore usually be combined with a stopper object that keeps the work piece in the right position for the duration of the identification.

The **detector/stopper** object is a combination of a stopper object and a detector object placed just in front of the stopper. Through the combination of these elementary objects, it can check whether a work piece is currently waiting at the stopper. It can also let a single work piece pass the stopper, ensuring its separation from any work piece immediately following it.

The **scanner** object is a combination of a reader object and a detector/stopper object. The stopper is positioned so that it keeps a work piece in the correct position to be identified by the reader. The scanner object can stop a work piece, identify it, and—when ordered to—release it, stopping and identifying the following work piece.

The **single item scanner** is the most complex conveyor object and combines a scanner object, a detector object, and a detector/stopper object. It represents a typical work position of an actor element, in which the detector/stopper first separates work pieces, letting only one work piece at a time proceed to the scanner. The scanner identifies the work piece, which may prompt the actor element to perform a process operation on it. Upon its release, the detector object determines whether the work piece has cleared the scanner area, which will then prompt the detector/stopper to admit the next work piece into the scanner.

Conveyor segments

Within an iTRAME element, conveyor ports and edges are contained in conveyor segments. There are only two types of conveyor segments: linear and switch.

A **linear conveyor segment** contains two conveyor ports and a conveyor edge that connects these ports. The segment is linear only with respect to its connectivity—a conveyor item entering the segment at one port can only leave it via the other. The shape of the edge (as defined by its edge path), however, can be any combination of linear and arc segments.

A **switch segment** contains three conveyor ports and three conveyor edges between them. At any time, only one of the edges is active and conveyor items may only move on the active edge. Which edge is active

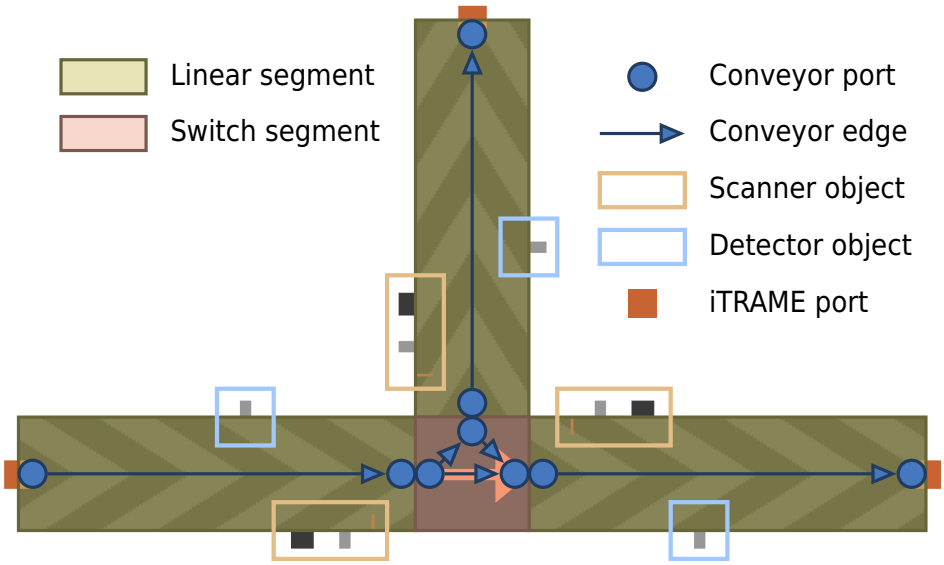


Figure 3.33: The components of a switch element.

may change during a simulation run. This enables the switch segment to route work pieces to different neighbouring segments. In the layout view, an orange arrow shows the current position of a switch segment.

The topology of an iTRAME element type is not defined in terms of conveyor ports and edges but in terms of interconnected conveyor segments (which indirectly defines a port/edge topology). Figure 3.33 shows the conveyor segments and conveyor objects that make up the upper level of a switch element. The lower level is structured similarly, with all conveyor directions reversed. In the switch element, three linear segments connect to one switch segment. Each of the linear segments contains a scanner object positioned on the edge leading to the switch segment. This scanner segment stops and identifies any work piece moving towards the centre of the switch. Once a work piece has been scanned, the switch element determines where it needs to go and, if the switch segment is clear of other work pieces, sets the active edge of the switch segment accordingly before releasing the work piece from the scanner object. The linear segments contain a detector object on the edge leading away from the centre. The switch element uses these detectors to determine when a work piece has cleared the central switch segment.

Conveyor and Cartesian coordinates

The position of conveyor objects and conveyor items is expressed in conveyor coordinates. A conveyor coordinate is a tuple (e, d) where e is an

edge of the conveyor graph and d is a distance from source port of e. It is possible to describe the movement of a conveyor item along the graph by regularly increasing the value of d and, when it reaches the length of the edge e, resetting it and replacing e with the edge starting at the conveyor port neighbouring the destination port of e.

This coordinate system is well-suited for computing conveyor item movement and the interaction of conveyor items with other items or conveyor objects. It is not suitable, however, for visualizing items and objects in the layout view. This requires their position and orientation to be known in three dimensional Cartesian coordinates relative to an origin point. These can then be transformed to screen coordinates according to the current layout view projection (top-down, isometric, or perspective).

A transformation from conveyor to Cartesian coordinates can be defined by selecting one conveyor port from the graph and assigning a 3D position and orientation to it. Following the edge paths of its edges, one can then assign a position and orientation to the other ports of the conveyor segment. Neighbouring ports have the same position and opposite orientation, which gives the position and orientation of one port of each connected conveyor segment. Continuing this process, one can determine the position and orientation of all conveyor ports that can be reached from the initial port. Finally, the Cartesian position and orientation of any conveyor coordinate (e, d) can be determined by combining the position and orientation of the source port of e with the relative position and orientation that results from following the edge path of e for the distance d.

3.7.2 Routing and pathfinding

Routing is the action of determining where a work piece should travel to next. Pathfinding is the action of determining how a work piece can get to its destination. Routing a work piece involves determining its executable operations, compiling a set of candidate actor elements, and selecting one of these elements as next destination of the work piece. A work piece is associated with a process from the core data, which defines the operations necessary to create the respective product. The executable operations are those that have all their dependencies fulfilled. The candidate actor elements are those that can perform at least one of the executable operations and can be reached from the current position of the work piece. Selecting one of these elements to actually perform the next operation on the work piece may be as simple as just picking any of the candidates or the result of a sophisticated evaluation that considers many different factors. This is the routing strategy defined in the production plan. The strategy used for the case study selects an actor element based on its distance from the work

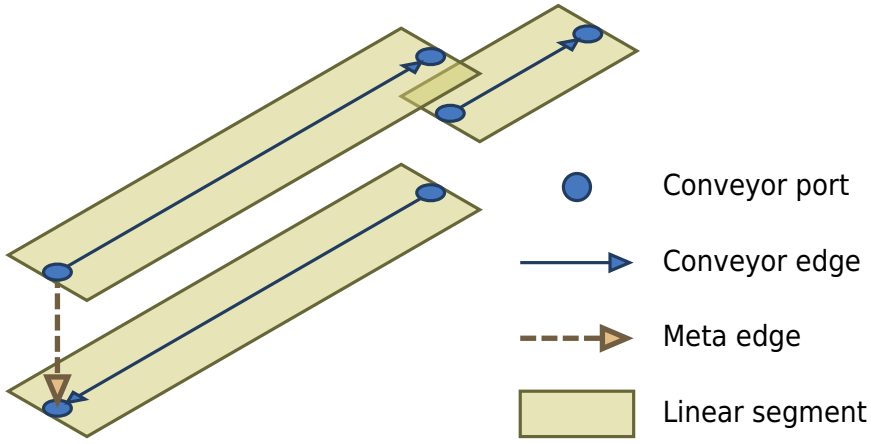


Figure 3.34: The lift element requires a meta edge to express that there is a path from the upper to the lower belt even if the moveable segment (to the right) is never connected to both of the other segments at the same time.

piece and the number of work pieces already en route to it while never routing a work piece past an actor element that could continue its process.

Pathfinding is done using Dijkstra's algorithm (Dijkstra, 1959), slightly modified to account for the vertex pair graph and meta edges. A meta edge is a special kind of edge required to successfully find paths across lift elements. A lift element is made up of three linear conveyor segments. Two of them are fixed and connect to a neighbouring iTRAME element. The third segment can move vertically and transport conveyor items between the two conveyor levels by dynamically connecting and disconnecting itself from the two other segments. When transporting items from the upper to the lower belt, for example, it is connected to the upper segment while waiting for an item, not connected to any other segment while in transit, and connected to the lower segment while waiting for the conveyor item to leave. As a result, there is never a continuous edge path between the upper and the lower segment and a pathfinding algorithm would have to conclude that there is no way of reaching the other conveyor level.

To indicate that there is, in fact, a way of reaching the lower level—one that involves waiting for the connectivity of the conveyor graph to change—the lift element contains a meta edge linking its two outermost conveyor ports (Figure 3.34). A meta edge is essentially a list of conveyor edges that can be passed in sequence, even if they may not all be connected at this point in time. The pathfinding algorithm can use this information to construct a path across the lift element and to determine the distance travelled when following the meta edge.

3.7.3 Simulation

Following the classification used by Law and Kelton (1999), *Sam*'s simulation is dynamic, deterministic, discrete, and event-based.

A **dynamic** simulation, as opposed to a static one, simulates the behaviour of a system over time. A static simulation is one in which the passage of time is disregarded or not relevant. *Sam* requires a dynamic simulation to properly model how work pieces may influence each other while many of them use the conveyor belt system at the same time.

In a **deterministic** simulation, as opposed to a stochastic one, repeating a simulation run with the same parameters and constraints will produce the same results. A stochastic simulation, on the other hand, contains random elements that may lead to slightly (or very) different results when repeating the simulation. There are no random elements in *Sam*'s simulation, so repeating the same simulation run will indeed return the same key figures, work piece density, etc. This has the advantage that different layouts can be compared without any stochastic influence which might just by chance make a layout look better or worse. On the other hand, real manufacturing processes are subject to stochastic influences. For example, it is unrealistic to assume that a human worker will take the exact same amount of time for each instance of a repeated task. In fact, how sensitive a given layout is to stochastic fluctuations in the duration of various activities may be a significant information for an analyst. This is not currently examined by *Sam*, but could be implemented by replacing the fixed durations for actor steps, lift movements, switch position changes, etc. with stochastic values and computing several simulation runs for the same layout, watching for significant fluctuations in the outcome. This might of course be supplemented by an automatism that provides statistical indicators such as the minimum and maximum values or standard deviation of key figures.

In a **discrete** simulation, as opposed to a continuous one, the state of the system changes only at specific points in time and is assumed to be constant between them. In a continuous simulation, the state of the system is a continuous function that can be evaluated at arbitrary points in time, which often involves modelling the system state as a differential equation. Many state changes in *Sam* layouts are inherently discrete (such as starting or completing an actor step or unblocking a conveyor item at a stopper object) whereas the work piece movement models a continuous process. However, expressing the movements of numerous work pieces and their interactions with other work pieces and conveyor objects would result in a very complex equation with no simple analytic solution. *Sam* therefore moves work pieces in regular, small, and discrete time steps, which is essentially a numerical approach to modelling their continuous movement.

Being **event-based**, *Sam*'s simulation keeps track of events that occur in the simulated world and may prompt a change of the system state. Some elements have the ability to both raise events and react to events raised by other elements. These so-called 'simulation objects' include some of the iTRAME elements and the conveyor objects. The simulation regularly generates time step events that prompt the movement of work pieces and other time-dependent activities. Other events include an actor element having completed an actor step or a scanner object having identified a work piece. When moving a work piece, the simulation can determine the distance a work piece travels during a time step by multiplying the conveyor speed and the duration of the time step. Traversing the conveyor graph starting at the current position for the distance travelled gives a path from the old to the new conveyor position. This path can then be checked for obstacles such as other work pieces or raised stopper objects and be truncated accordingly.

During the simulation run, work pieces and actor elements record all status changes along with the time of the change in their status history. Work pieces can change their status between 'pending', 'working', 'moving', 'waiting', and 'done'. Actor elements change between 'working' and 'waiting'. A visual representation of the status histories can be viewed in the attribute view of the simulation run (Figure 3.15). The status histories of the actor elements can be used to calculate the actor load and cost key figures. The simulation ends successfully when all work pieces have reached the 'done' state. It fails if at one point a work piece cannot continue its process because it cannot reach any actor element that can perform one of its executable operations. The simulation run may also fail after reaching a predetermined time limit.

3.8 Automatic layout discovery

Sam's automatic layout discovery component assists users in planning a reconfigurable manufacturing layout by offering suggestions for layout modifications. While the user works with the system and designs and evaluates layouts, the automatic discovery creates and evaluates alternative layouts in the background. Figure 2.6 on page 23 illustrates this parallel execution of manual and automatic sampling of the parameter space. Both methods base their decisions on the automatic evaluation provided by the simulation. Although the two branches are largely independent, feedback can be provided in either direction: If the automatic sampling discovers a layout whose evaluation is in one or more aspects better than other layouts seen so far, it can bring this to the user's attention via the discovery

highlights view (Figure 3.21 on page 59). The user implicitly provides continuous feedback by designing and evaluating layouts. Assuming that the layouts designed by a human analyst are generally better than a randomly chosen layout, these layouts are used as starting points for the automatic search of the parameter space. The user can also interact explicitly with the automatic discovery by viewing its state and the layouts it currently considers and by specifying search priorities with respect to the key figures calculated by the simulation. This follows the visual analytics paradigm in that the user and the automatic discovery essentially share their findings, with each party providing feedback and suggestions to the other.

3.8.1 Evolutionary algorithms

Sam's automatic layout discovery uses an evolutionary algorithm. Evolutionary algorithms are a class of algorithms that mimic the processes of evolution to compute solutions to optimization problems. They commonly contain a probabilistic element and do not guarantee to find an optimal or even acceptable solution. Well configured, however, evolutionary algorithms may produce results that converge to a sensible solution much more quickly than algorithms that eventually return an exact solution. Consequently, they can be very useful when the time it takes to produce a result is valued higher than the optimality of the result. This is often the case for problems whose size and complexity preclude the computation of an exact solution in an acceptable time frame. The approach can be adapted to a wide range of optimization problems.

Natural evolution

Evolutionary algorithms copy some of the processes of natural evolution, albeit in heavily simplified form. The theory of evolution is a widely accepted scientific explanation for the complexity and variety of life on Earth. It gained widespread attention with the publication of Charles Darwin's *On the Origin of Species* in 1859, promoting the assumption that all forms of life on Earth evolved from a single common ancestor by applying natural selection over the course of almost four billion years. The process involves a population of individuals with finite living spans, which produce offspring that differ slightly from their immediate ancestor. Interaction with their environment and competition over limited resources applies selection pressure to the population, meaning that some individuals are more likely to survive and reproduce than others. If the rate of survival and reproduction correlates with a specific trait, then this favourable trait will, in the long term, become more common in the population. Disadvantageous

traits, on the other hand, which reduce an individual's ability to survive and reproduce, will become less frequent or—quite literally—die out.

This process is called (natural) selection. The term fitness is used to describe the chance an individual has to be selected, i.e. to survive and reproduce. This is what the catchphrase 'survival of the fittest' (coined by British philosopher Herbert Spencer in 1864) refers to. It is important to note, however, that an individual's 'fitness' in this sense does not necessarily correspond to its physical shape or strength. It is rather a measure of how well an individual fits its immediate environment. Also, a high fitness is by no means a guarantee for survival or reproduction, as there is always an element of chance involved. Fitness is a statistical advantage, which may or may not apply to a specific individual. Selection acts both as sexual selection, which selects individuals for reproduction, and as ecological selection, which selects individuals for survival, and an individual's fitness may be different in both regards. The extent to which fitness influences selection is called selection pressure and may vary with the environment. In harsh environments, only very specialised individuals may survive, whereas when conditions are favourable and resources abundant, a wider variety of individuals may be able to contribute to the next generation.

The generation of similar yet slightly different variants of existing individuals is achieved through mutation, mostly random alterations introduced during the duplication of an individual. Such alterations may be caused both by imperfections in the physical duplication process and by external influences. The extent of these differences is determined by the mutation rate, which is an optimization problem in itself: If the mutation rate is too low, this will essentially disable evolution for the individual and its descendants, possibly putting them at a disadvantage with other lines that continue to evolve. If the mutation rate is too high, offspring may be so radically different not just from their ancestor but from any successful individual that they are unable to live at all, let alone reproduce. In general, more complex organisms require a higher precision in their duplication. This precision and thus the mutation rate can be controlled by including various redundancies and error detection mechanisms in the genetic information.

Mutation, as an almost inevitable side effect of reproduction, may be supplemented by recombination. Recombination occurs during sexual reproduction: Two individuals of a population generate a common offspring, which inherits a combination of traits from both parents. Populations that reproduce sexually are commonly divided into two distinct sexes (male and female) and only pairs of opposite sexes can reproduce. Often, only one of the sexes actually produces offspring while the other merely contributes genetic information. As an obvious disadvantage, the non-productive half of the population nevertheless consumes resources. Also, sexual selection

may favour traits that reduce an individual's ability to survive (such as a flamboyant appearance that attracts mating partners and predators alike). In spite of these apparent drawbacks, the vast majority of species living today reproduce sexually, which suggests that sexual reproduction provides benefits that outweigh these disadvantages. One benefit may be found in the increased genetic diversity of the population, which may reduce the change of a population being eradicated completely by external influences. For example, it is interesting to note that a small percentage of Europeans carry a genetic mutation that makes them naturally immune to the dreaded HI virus (Liu et al., 1996).

Individuals of a population may be regarded in terms of their genotype or their phenotype. The genotype is the genetic code the individual carries. The phenotype is the individual's observable appearance and traits that result from the genotype. In animal and plant life, the genotype is encoded in DNA molecules, long strands that form sequences of four types of nucleobases: cytosine, guanine, adenine, and thymine. Through complex biological processes, these basic building blocks eventually determine the appearance and traits of the phenotype. Mutation and recombination act only on the genotype and thus affect the phenotype only indirectly. An individual's fitness, on the other hand, is determined by its phenotype and only indirectly affected by the genotype.

Genetic algorithms

Evolutionary algorithms encompass a number of different approaches inspired by one or more of these natural processes. *Sam* uses a genetic algorithm. A genetic algorithm manages a population of individuals, in which each individual represents a possible solution for the optimization problem. Individuals are represented by their genotype, a compact representation of the solution such as a tuple or a vector. The algorithm repeatedly applies selection, mutation, and recombination operators on the population until a termination condition is met. The general procedure is:

```
generate an initial population;
while termination condition not met do
    select individuals for reproduction;
    generate children by mutation and recombination;
    select individuals for survival, remove the rest;
end
output the best individual in the population;
```

An actual implementation will have to make various design decisions:

- A suitable representation of the solution genotype has to be defined, as well as an operation that transforms this genotype into an actual solution (the individual's phenotype). Section 3.8.2 describes *Sam*'s genotype representation.

- A fitness function has to be specified, which assigns higher fitness values to solutions that more closely match the optimization goals. For complex goals, the fitness function may have to weigh a variety of aspects. *Sam*'s fitness function is a combination of a layout's key figures. The weight of each key figure (and thus the search priorities of the optimization) can be defined by the user.

- The size of the initial population and the method for creating it have to be determined. Typically, a number of individuals are randomly generated, but more specific ways of generating the population are possible. *Sam*'s initial population consists of 25 of the known user-created layouts. If there are less than 25 user-created layouts, the remaining slots will be filled with random layouts.

- The algorithm needs a termination condition that determines when to stop the evolution and return a result. This can be a time constraint or a limit on the number of generations. An algorithm may also terminate when a solution satisfies certain criteria or when the fitness improvement between two generations falls below a given threshold. Since *Sam*'s evolutionary algorithm is interactive, constantly reporting the current results in the discovery view, its evolution continues to run until it is stopped by the user.

- It has to be decided whether sexual selection, ecological selection or both are to be used and how the selection is to be done. Selection can be deterministic, such as always selecting the individuals with the highest fitness, or probabilistic, such as drawing a random sample biased by fitness. *Sam* generates 25 children per generation by selecting 25 (not necessarily distinct) pairs of parent individuals in a biased random selection (favouring individuals with higher fitness), recombining them and mutating the result to produce one child individual per parent pair. Of the resulting 50 individuals, the 25 with the lowest fitness are discarded.

- If mutation is to be used, a mutation operator has to be defined. This operator takes a genotype and returns a slightly different genotype. The actual mechanics of the operator depend both on the

genotype representation and the problem domain. Section 3.8.2 describes *Sam*'s mutation operator.

- If recombination is to be used, a recombination operator has to be defined. This operator takes two genotypes (parents) and returns one or two derived genotypes (children). Section 3.8.2 describes *Sam*'s recombination operator.

3.8.2 Implementation

Sam's genetic algorithm operates on manufacturing process layouts, therefore the phenotype of population individuals are layouts and thus complex data structures. The genotypes of the individual is a simplified representation of the layout as a sequence of steps $L = (s_1, s_2, \ldots, s_n)$, where a step is a tuple (t, e, l, O). The layout can be constructed by executing these steps in sequence. A step is executed as follows:

1. Create an element of type t (where t is one of { transport, corner, manual labour station, robot station, lift, switch }).
2. Find this element's port identified by index e (where $0 <= e < n$ with n being the number of ports of this element type).
3. In the layout constructed so far, find the port identified by index l (where $0 <= l < n$ with n being the total number of ports in the current layout).
4. Connect the element's port identified by e to the layout port identified by l, making it part of the layout.
5. If the element is an actor object (manual labour station or robot station), add the operations O to the set of operations it can perform.

For the very first step, 2., 3., and 4. do not apply and the values of e and l are irrelevant.

This genotype representation is essentially a linearization of the layout graph. The resulting layouts are guaranteed to be connected. However, not all possible genotypes correspond to a valid layout:

- For each step, l must identify a layout port that is free, i.e. not yet connected to any other port.
- The execution of a step must not cause layout elements to overlap.

Mutation

A genotypes can be transformed into another genotype representing a similar layout by applying *Sam*'s mutation operator. The mutation operator performs one or more of the following operations:

- Insert a new, random step into the genotype.
- Remove a step from the genotype.
- Change the element type t in a step.
- Change the element port e in a step.
- Change the layout port l in a step.
- Add an operation to a step's set of operations O.
- Remove an operation from a step's set of operations O.

Some of these operations are difficult to perform directly on the genotype representation. For example, changing the layout port of a step, which reconnects the corresponding element to another element in the layout, may require a reordering of steps to ensure that elements are constructed only after the element they connect to has been defined. To simplify such operations, *Sam* transforms a genotype into a connectivity graph, computes the mutation on this graph, and transforms the result back to a genotype representation. Figure 3.35 shows these three representations of an individual side by side.

Recombination

There are two obvious ways of combining two layouts:

- Layout genotypes can be recombined by selecting a number of steps from the beginning of one genotype and a number of steps from the end of the other and combining these steps to form a genotype for the new layout.
- Layouts connectivity graphs can be combined by selecting a port in each graph, disconnecting these ports from their current neighbours, connecting them to the selected port in the other graph, and finally discarding those parts of the resulting graph that can no longer be reached from the selected ports.

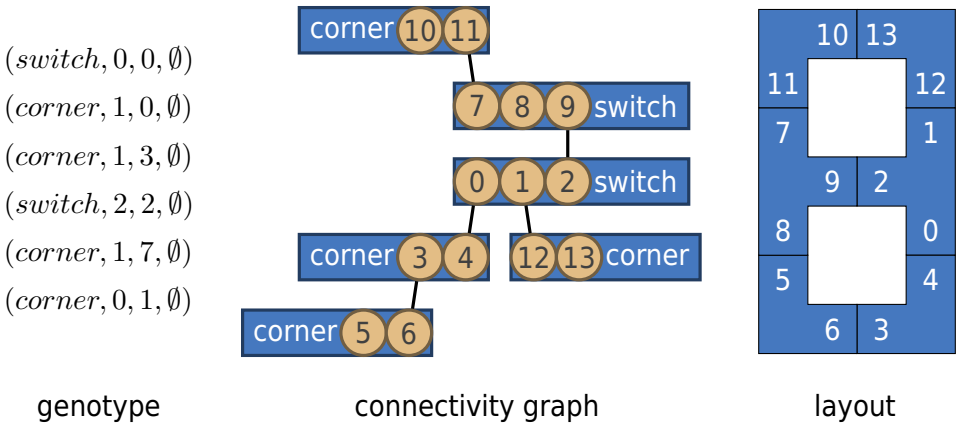


Figure 3.35: Three representations of the same layout. Left: genotype (as a sequence of steps), centre: the corresponding connectivity graph, right: the resulting layout. The numbers indicate the layout port indices referenced by the third step component.

The second method requires both layouts to be transformed into the connectivity graph representation but ensures consistent port numbers and can therefore be expected to produce fewer invalid genotypes than a simple recombination of steps. The evolutionary algorithm contains a strong probabilistic element, which makes its results partly depend on chance, but informal experiments (Figure 3.36) suggest that graph recombination is generally superior to using no recombination and in the majority of cases it performs better than genotype recombination. It is therefore selected as the recombination operator used by *Sam*'s layout discovery.

3.9 Future perspectives

While *Sam* has been developed with the iTRAME system in mind, it models the various iTRAME elements in terms of linear and switch segments, ports, stoppers, and item detectors. Any other reconfigurable manufacturing system that uses elements which can be modelled using the same building blocks can be integrated into *Sam* without major changes. Systems with more fundamental differences may require more effort, but as long as one can define a simulation model that produces appropriate key figures, any manufacturing system can be used as basis for designing, evaluating, comparing, and discovering layouts using *Sam*.

When multiple experts are involved in the layout planning process, *Sam* can benefit from large displays such as the one in Figure 3.37. Because the different views can be arranged freely and views on different layouts or

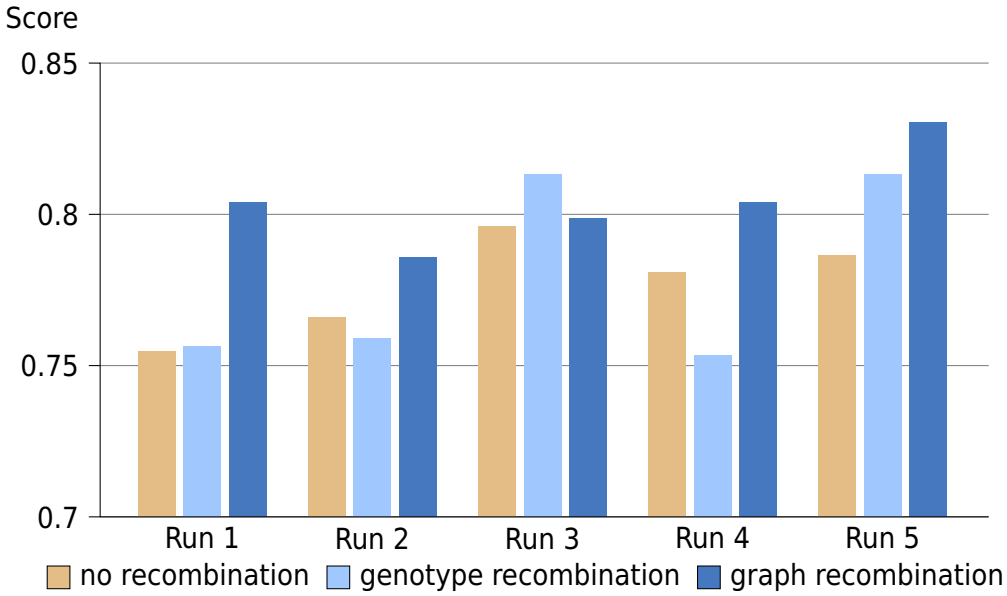


Figure 3.36: A comparison of the best score values achieved by the three recombination methods after 15 generations. Each run used a different initialization value for the random number generator.

different views on the same layout can be placed side-by-side, *Sam* can efficiently use the available display space.

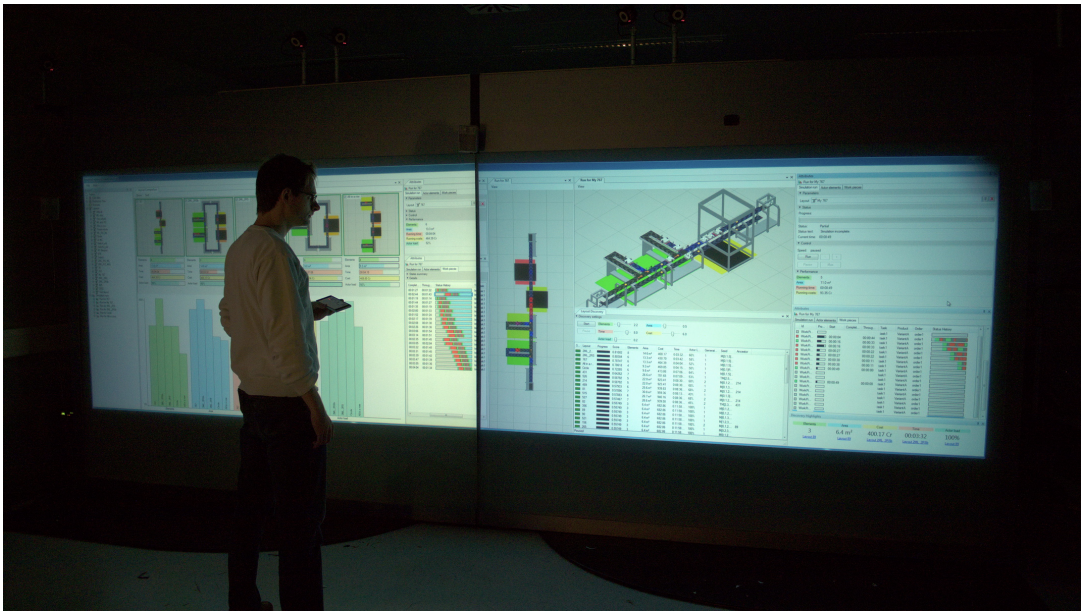


Figure 3.37: *Sam* running on a powerwall installation, where it benefits from ample space for arranging the various views.

4

Machine Condition Classification

Machine tools are important elements of manufacturing processes. A machine tool is a device that uses some kind of tool to cut or drill a metal work piece. It usually follows a computer controlled path to shape a raw metal block according to a CAD created design. They are high precision devices, made of many parts, equipped with numerous sensors, and controlled by complex electronics and software. They are also very expensive, so anyone investing into a machine tool has high expectations regarding its availability and reliability. Consequently, the precise diagnosis of actual and potential problems is an important issue for machine tool manufacturers, both for checking new machines before shipping them to the customer and for diagnosing machines during maintenance. Diagnosing a machine involves one or more tests, in which the machine performs certain tasks while recording various sensor readings. Viewing visualizations of these sensor readings, an expert will usually be able to judge whether certain parts and elements behave as expected or need to be repaired or replaced.

Visual analytics can make this process more effective by offering a more systematic and partly automatic approach. It can also enable non-experts to perform the diagnosis based on pre-configured automatic analysis steps. This problem statement is different from the one in Chapter 3 in that here, the data to be analysed is explicitly given in the form of files containing the

Parts of this chapter have previously been published in:

Wörner, M., Metzger, M., and Ertl, T. (2013), 'Dataflow-based Visual Analysis for Fault Diagnosis and Predictive Maintenance in Manufacturing', in *EuroVA 2013: International Workshop on Visual Analytics* (Leipzig: Eurographics Association), 55–59.

recorded sensor readings. Also, the goal is to classify record sets into ones indicating functional and ones indicating defective conditions rather than finding an optimal entity in a data set.

Monitoring the condition of a machine has always been an important topic in the manufacturing domain. There are sophisticated approaches for fault diagnosis and predictive maintenance. These include wavelet transformations (Wang et al., 2012) and machine learning techniques such as principal component analysis and support vector machines (Tang et al., 2010). Some methods do not require the installation of dedicated diagnostic sensors but can operate on signals a machine generates during its normal operation (Verl et al., 2009). Effort has also been put into including diagnostic algorithms on a chip in the machine (Ong et al., 2010). These methods strive for an automatic detection and diagnosis of problems. They do not require human intervention and have no need for an interactive visual interface. However, in an extensive review of condition-based maintenance, Jardine et al. (2006) note that in spite of the availability of advanced maintenance techniques in the literature, it is still common in the industry to either simply maintain machines on a regular schedule or wait until a breakdown occurs.

Both strategies have specific drawbacks: A maintenance schedule will usually have to be conservative enough to prevent most or all maintenance-related breakdowns. This, however, might result in a regular repair of things that were not broken, such as the replacement of parts which were due for replacement but still in good shape. Waiting until a breakdown occurs, on the other hand, makes sure only necessary maintenance is conducted, but has the obvious drawback of frequent and generally unpredictable standstills. A third alternative is predictive maintenance, monitoring the condition of a machine and initiating maintenance operations only when sensor readings indicate they are necessary.

Jardine et al. list several possible reasons for the industry preferring the first two strategies, including a ‘lack of efficient communication between theory developers and practitioners’. Visual analytics might help to establish this efficient communication by offering automatic classification of diagnostic data while integrating the human expert into the process. Using a combination of visual and automatic analysis may allow experts to judge visual data representations and determine patterns and correlations that can then be used by automatic methods to generalize these findings to other measurements.

This chapter describes a visual analytics system for the classification of machine data records. The implementation of this system was done by Michael Metzger as part of his diploma thesis.

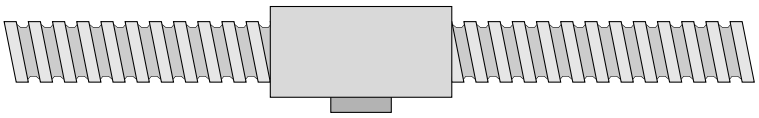


Figure 4.1: Schematic view of a ball screw.

4.1 Machine data

The data set used in this chapter was provided by a machine tool manufacturer. It contains data collected during machine test runs. In a test run, an operator performs a defined task on the machine while the machine continuously records the values of several sensors. In the data used, this task was a ball screw test. A ball screw (Figure 4.1) converts the rotational movement of a rod into a precise linear movement, for example to position the tool at the work piece. The machine can measure the current position of the ball screw in two ways: Directly, using markings on a measuring rod, and indirectly, counting revolutions of the motor driving the ball screw. Over time, wear will reduce the precision of the positioning and thereby the quality of the parts manufactured by the machine. The analyst's task is to determine whether the data of a given test run indicates a defect and thus the necessity to replace the ball screw.

Performing a test run involves moving the ball screw head forth and back over its entire range at a constant speed. Six sensor readings are recorded during this test. These include the nominal position, the difference between the nominal and the actual position, the velocity of the movement, and the difference between the direct (measuring rod) and indirect (motor revolutions) position measurements. There were 40 runs for this test. Every sensor was sampled at 167 Hz and recorded a total of 8192 numeric values (spanning about 49 seconds). The machines sample all sensors simultaneously, but there is typically a gap between the start of the recording and the start of the test as both are triggered manually.

4.2 The analysis system

In this scenario, an analyst's goal is to find a method to automatically estimate the state of a ball screw given its test run data. This method constitutes a classifier, which classifies test run data into 'good', 'bad', and possibly intermediate, borderline cases. It is not meant to replace the analyst but to assist, for example by facilitating a quick dismissal of clear-cut cases and allowing the analyst to focus on the more intricate cases, which require a much more careful examination. Here, designing a

classifier involves transforming the original data sequences into suitable metrics, provide a human pre-classification for several test runs, and then use machine learning to create a classifier that classifies this test set—and hopefully other, as of yet unknown cases, too—correctly.

The manufacturer's current solution to the classification problem involved an expert looking at and judging the data. Consequently, there was no clear indication and formal definition of which metrics could be used for machine learning. The implemented system therefore supports an explorative approach, in which an analyst combines different processing steps into a data flow graph to devise a method of loading, transforming, and evaluating data and trains this classifier by specifying a test set of pre-classified test runs. This graph-based representation of an analysis process has previously been used in generic analysis tools such as KNIME (Berthold et al., 2009) or RapidMiner (Mierswa et al., 2006). Using this classifier on another set of test runs and comparing the results with a manual evaluation, an analyst can assess and verify its usefulness. Once a domain expert has designed a graph for a particular analysis problem, non-expert users may be able to reuse this graph to perform analysis tasks on their own. Experts would only have to update the graph when the analysis task changes (such as when there is a modification to the machine or sensor design) or users report incorrect classifications.

The system's main window (Figure 4.2) is divided into three parts. The left part shows the analysis nodes that can be used to build the analysis graph. The central part contains the interactive visualization of the graph, which the user can edit by adding, moving, removing, and connecting nodes. Each node represents an analysis component and has a certain number of input and output ports. The colour of a port indicates its data type, and ports of the same colour can be connected to form a dataflow through the graph. A legend in the lower left corner of the window lists the meaning of each colour (Table 4.1). Hovering over a node port will show a tooltip that describes the data this port provides or expects. All nodes have a settings button in their upper right corner, which can be used to specify parameters for this particular analysis step. Some nodes also show a visualization button in their lower right corner, which will open a visual representation of relevant information. To the right of the graph view, there is a table of all test runs that are currently loaded. The first column shows the machine identification and the date of the run. The second column contains the run's manual pre-classification as 'good' (green tick), 'bad' (red cross), or 'indetermined/borderline' (grey circle). Graph nodes can add columns to this table to display relevant information. As an example, the third column in Figure 4.2 displays a numeric result computed by the 'MinMaxAvg' node and the fourth column displays the automatic classification of the 'Result'

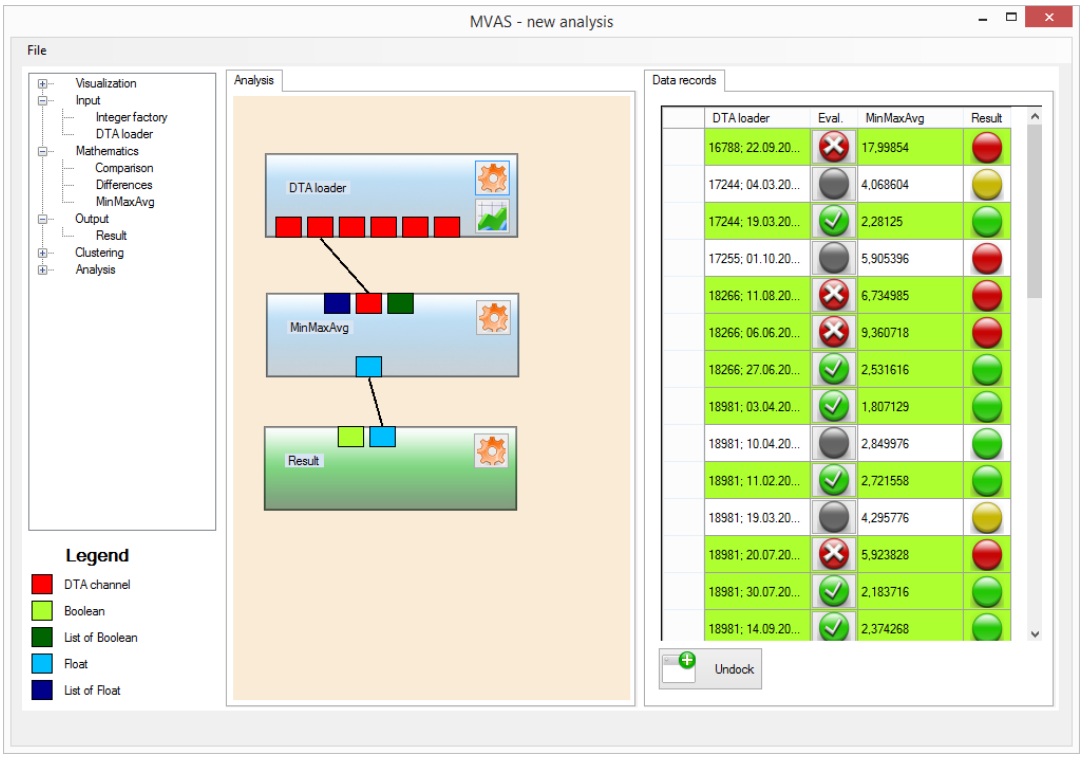


Figure 4.2: The main window of the prototype implementation: Available components on the left, analysis graph in the centre, test runs and component results on the right. Green rows indicate cases in which the manual classification (second column) and the automatic classification (fourth column) match.

node as a coloured circle.

4.2.1 Analysis components

The user combines nodes into an analysis graph, which the system traverses once for each test run. Figure 4.3 shows the available nodes.

The **DTA loader** node ('DTA' being the file format of the test run data) loads data files and provides one data channel output for each recorded sensor. A data channel contains the time series of values recorded by a single sensor in the current test run. The node has no input ports and acts as a data source in the analysis graph. Its parameters define where the data files are located and which test runs are to be read from these files. The node has a visualization button, which will open a plot of the loaded data (see Figure 4.5 for an example). Here, the user can examine the various channels and measurements or compare plots of different machines. These diagrams are drawn using the open source graph drawing library ZedGraph

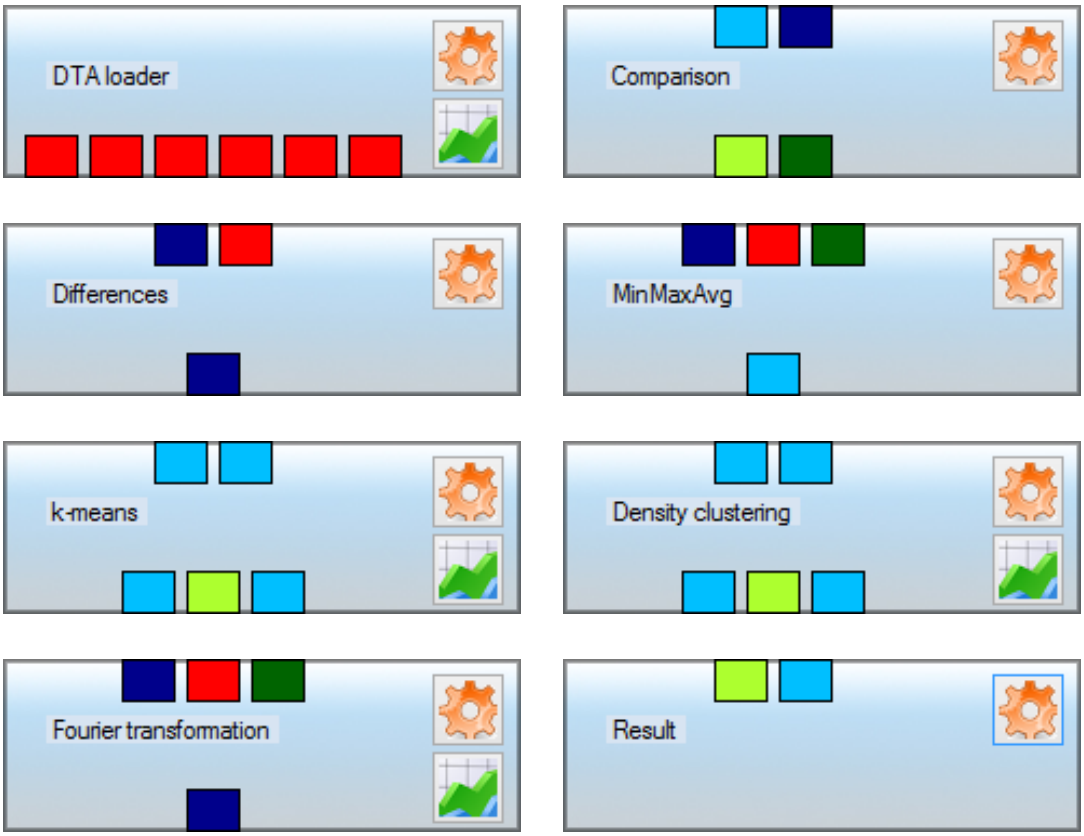


Figure 4.3: The different analysis components: DTA loader, Comparison, Differences, MinMaxAvg, k-means, Density clustering, Fourier transformation, and Result.

Colour	Data type
red	data channel
light green	Boolean value
dark green	list of Boolean values
light blue	floating point value
dark blue	list of floating point values

Table 4.1: Colours indicate the data type of node ports.

(Champion et al., 2012).

The **Comparison** node compares an incoming floating point value or list of floating point values to a specified constant and outputs a Boolean value (or a list of Boolean values) that states whether the input was greater or less (depending on the component's settings) than that constant.

The **Differences** node computes the differences between subsequent values in a list of floating point values or a data channel and outputs them as a list of floating point values.

The **MinMaxAvg** node takes a list of floating point values, a list of Boolean values, or a data channel and computes their minimum, maximum, or average value (depending on its settings).

The **k-means** node computes a two-dimensional k-means clustering (MacQueen, 1967) and, for each test run, adds the two input floating point values to the value set to be clustered. The number of clusters is given in the component's settings. The output ports give the cluster index along with an evaluation (as a Boolean and a floating point value) of this test run. The evaluation is created by labelling clusters that contain pre-classified test runs to match the pre-classification and apply this classification to all other test runs in the same cluster.

The **Density clustering** node is similar to the k-means node but uses the DBSCAN algorithm (Ester et al., 1996) to compute a clustering without a preset number of clusters.

The **Fourier transformation** node computes a fast Fourier transform (Cooley and Tukey, 1965) on a list of floating point values or a data channel to transform them into the frequency domain. The visualization of this node shows a graph of the computed spectrum (Figure 4.9).

The **Result** node has no output ports and acts as a data sink. Its input port accepts a Boolean value for each test run, classifying it as 'good' or 'bad'. A second input port accepts a floating point value instead to allow for a fuzzy classification of test cases. Two threshold values k_0 and k_1 can be specified in the node's settings and the node will classify a test run with an input value x as 'bad' if $x \leq k_0$ and as 'good' if $x \geq k_1$. For $k_0 < x < k_1$, the test case will receive a fuzzy classification between these extremes (Figure 4.4). It is possible to have the Result node determine these threshold values automatically based on the pre-classified test runs. The Result node adds a column to the test runs table that shows the classification result as a circle coloured green (good) or red (bad). If a fuzzy classification is used, intermediate shades are also possible. When a test run has both a manual pre-classification and an automatic classification provided by a Result node, the background colour of the corresponding table row indicates whether these two evaluations match (green) or disagree (red).

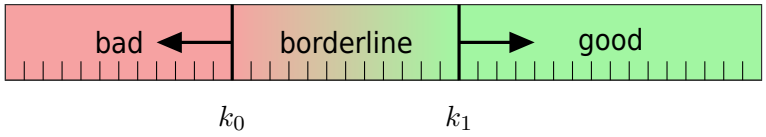


Figure 4.4: The result node classifies a test run based on an incoming value x and two threshold values k_0 and k_1. A test run is 'bad' if $x < k_0$ and 'good' if $x > k_1$.

4.3 Case study

The following case study demonstrates how an analyst might construct an automatic classification method using the implemented visual analytics system. While the actions are described from the point of view of a hypothetical analyst, the data used for the study is real, as are the pre-classifications provided by a domain expert.

The analyst starts by adding a DTA loader node to the analysis graph. This node loads the test runs and lists them in the table in the right part of the main window. The next step is to provide a pre-classification for some of these runs by using the visualization of the DTA loader node to view diagrams of the individual data channels, judging the state of the machine, and entering this assessment by marking the corresponding table entry as 'good', 'bad', or 'indetermined'. Comparing DTA loader diagrams of good and bad cases, the analyst notices that the deviation between the nominal and the actual ball screw head position is generally larger in the bad cases (Figure 4.5). Consequently, the average (absolute) value of this deviation might be a useful indicator value. The analyst adds a MinMaxAvg node, connects it to the DTA loader output port corresponding to the position difference channel, and sets it to compute the average of the absolute input values. In the test run table, a new column appears listing the average absolute position deviation for each test run.

Noticing that 'good' test runs indeed seem to have a lower average position difference, the analyst adds a Results node to create a classification of the test runs based on these values. This new node gets connected to the output of the MinMaxAvg node and, based on the available pre-classifications, automatically learns that the greatest average value of a 'good' test run is 2.72, whereas the least average of a 'bad' test run is 5.92. Figure 4.6 shows the resulting analysis graph. In the test run table, all rows with pre-classifications turn green, confirming that for these runs, all classification results match the expert assessments. Values between 2.72 and 5.92 are mapped to a gradual progression from 'good' to 'bad', and among those test runs without a known expert classification, the analyst

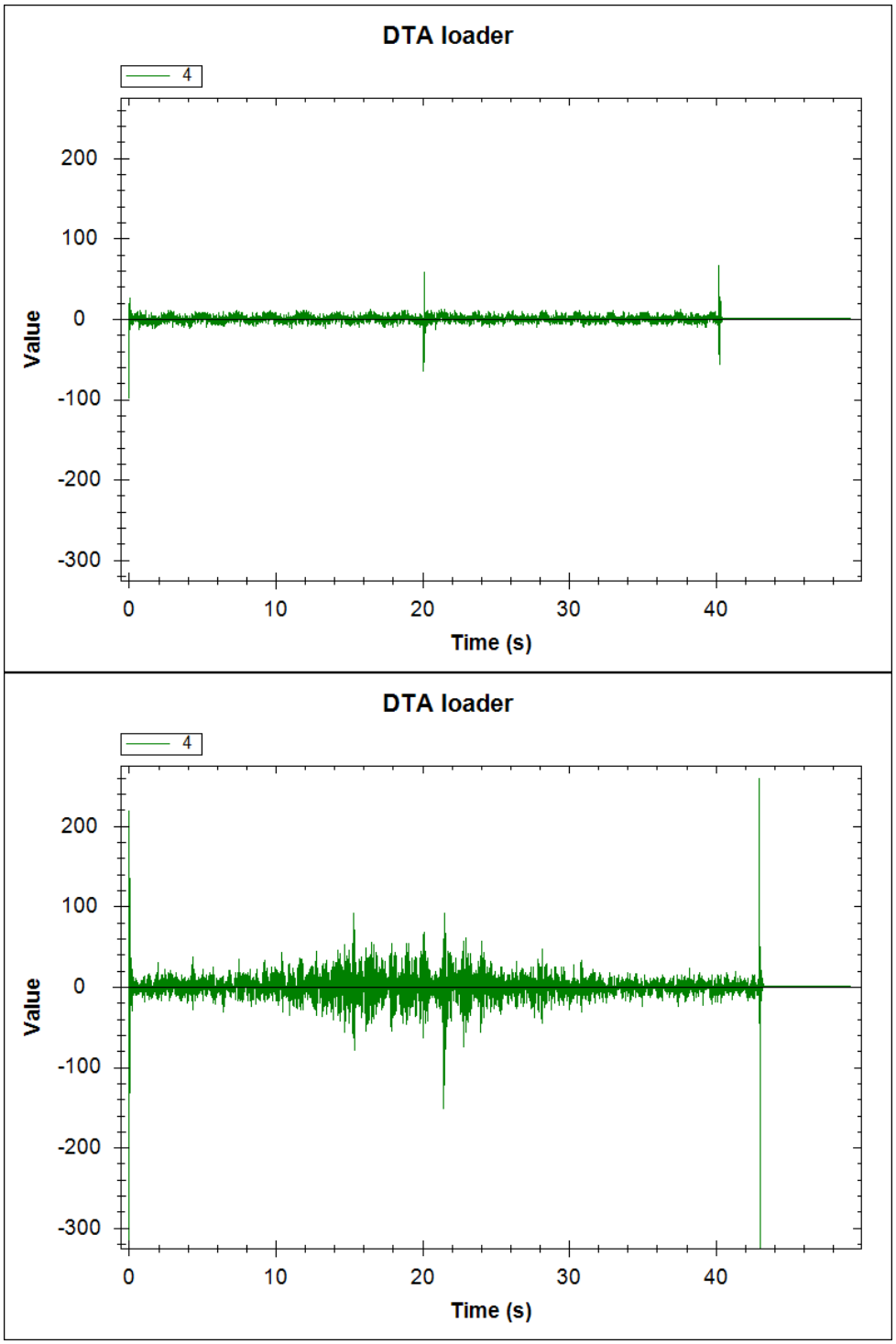


Figure 4.5: The data channel recording the difference between the nominal and the actual ball screw head position in cases pre-classified as good (top) and bad (bottom) as seen in the data visualization view of the DTA loader node.

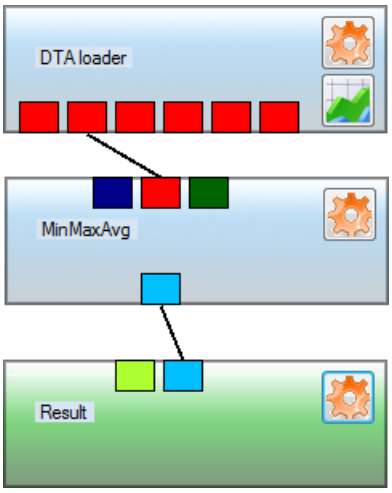
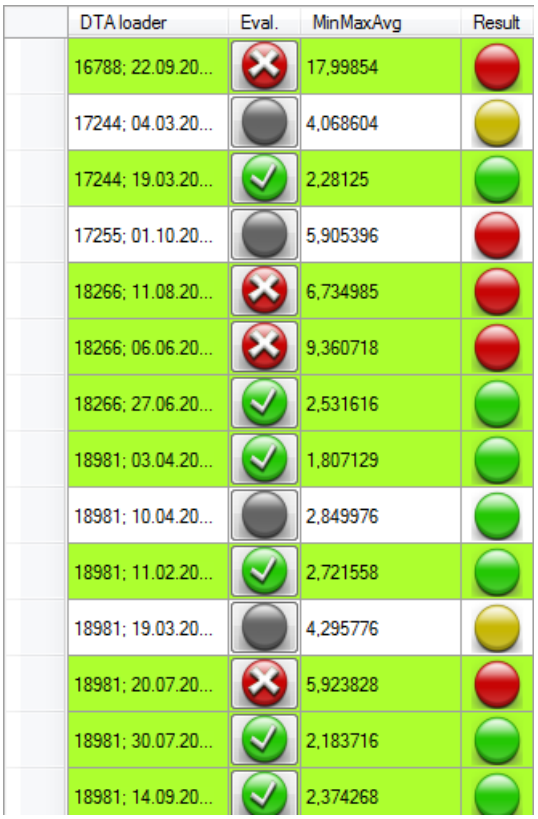


Figure 4.6: Using the first data channel to create a mask that limits the analysis to periods of steady head movement.

can now identify some that are deemed borderline cases according to this classifier (Figure 4.7).

While the average position deviation appears to be a good indicator, the plots show that even in the 'good' cases, there are noticeable peaks (as in Figure 4.5 top). These usually occur at the beginning, in the middle, and at the end of the measurement. With the necessary domain knowledge, an expert can tell that these peaks are caused by the starting, stopping, and turnaround motion of the ball screw head. They are an expected occurrence in every test and do not indicate a defect. With this knowledge, the analyst decides these points should best be ignored when calculating the average deviation.

The data channel containing the absolute nominal head position can be used to determine the time periods in which the head is meant to accelerate or decelerate and exclude these from the further analysis. To model this in the analysis graph, the analyst adds a Differences node and a Comparison node. The Differences node calculates the difference between two consecutive measurements in the nominal position data channel and the Comparison node is configured to produce a list of Boolean values that state whether the corresponding absolute position difference (and thus head velocity) indicates a steady movement rather than an acceleration or deceleration. This list of Boolean values can then be connected to the MinMaxAvg node, acting as a mask that limits its calculation to these time periods. Figure 4.8 shows the resulting analysis graph. Once the new connections have been made, the MinMaxAvg component recalculates the averages and the Results node updates its classification limits. It now



	DTA loader	Eval.	MinMaxAvg	Result
	16788; 22.09.20...	✗	17,99854	●
	17244; 04.03.20...		4,068604	●
	17244; 19.03.20...	✓	2,28125	●
	17255; 01.10.20...		5,905396	●
	18266; 11.08.20...	✗	6,734985	●
	18266; 06.06.20...	✗	9,360718	●
	18266; 27.06.20...	✓	2,531616	●
	18981; 03.04.20...	✓	1,807129	●
	18981; 10.04.20...		2,849976	●
	18981; 11.02.20...	✓	2,721558	●
	18981; 19.03.20...		4,295776	●
	18981; 20.07.20...	✗	5,923828	●
	18981; 30.07.20...	✓	2,183716	●
	18981; 14.09.20...	✓	2,374268	●

Figure 4.7: The green rows in the test runs table confirm that the current graph has classified all pre-classified test runs correctly. White rows have no pre-classification.

classifies test runs as ‘good’ if their average value is not greater than 2.99 and as ‘bad’ if it is not less than 6.87. Compared to the first analysis without the mask, the separation between good and bad increased from 3.20 to 3.88.

4.3.1 The frequency domain

Although the average value of the position difference appears to be well suited to classify these test runs, it completely ignores any temporal component of the signal. The plots of bad machines show that defects often do not simply increase the average value, but show time-dependent anomalies. An example is the data series in Figure 4.5 bottom, which reaches greater values near the turning point in the middle. There are other cases in which values are greater in the second half of the measurement or values are generally within the limits except for a short period with extreme oscillations. To include the temporal component of the time series in the analysis,

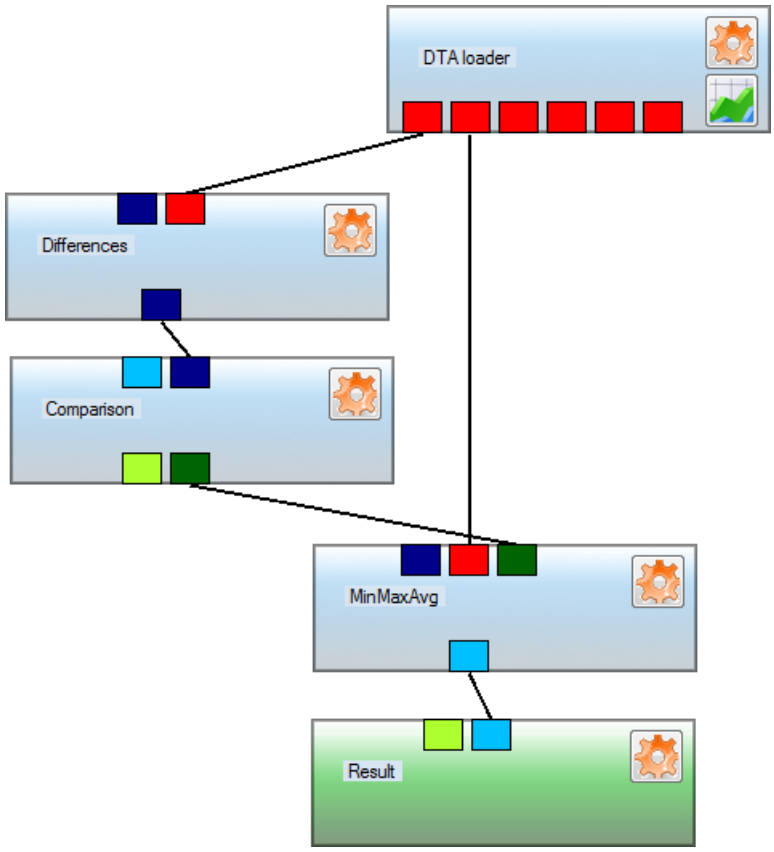


Figure 4.8: Using the first data channel to create a mask that limits the analysis to periods of steady head movement.

the analyst adds a Fourier transformation node. This node computes the frequency spectrum of a data channel and can thus detect temporal patterns in the data. The visualization of the Fourier node shows a graph of the spectrum (Figure 4.9). Comparing several spectra, the analyst notices that 'bad' test runs typically exhibited significant oscillations at around 20 Hz, which the 'good' test runs do not, and concludes that limiting the average calculation to a frequency band around 20 Hz might further increase the robustness of the classification. The resulting graph can be seen in Figure 4.10. As before, a Differences and a Comparison component limit the analysis to periods of steady ball screw head movement. A Fourier transformation node calculates the spectrum of the remaining section of the position difference channel. The settings of this node limit the spectral analysis to a range of 16–32 Hz. The result is aggregated into an average value by the MinMaxAvg node and the Result node finally classifies the results. As before, all pre-classified test runs are classified correctly and the analyst feels this classifier is now ready for a more extensive evaluation.

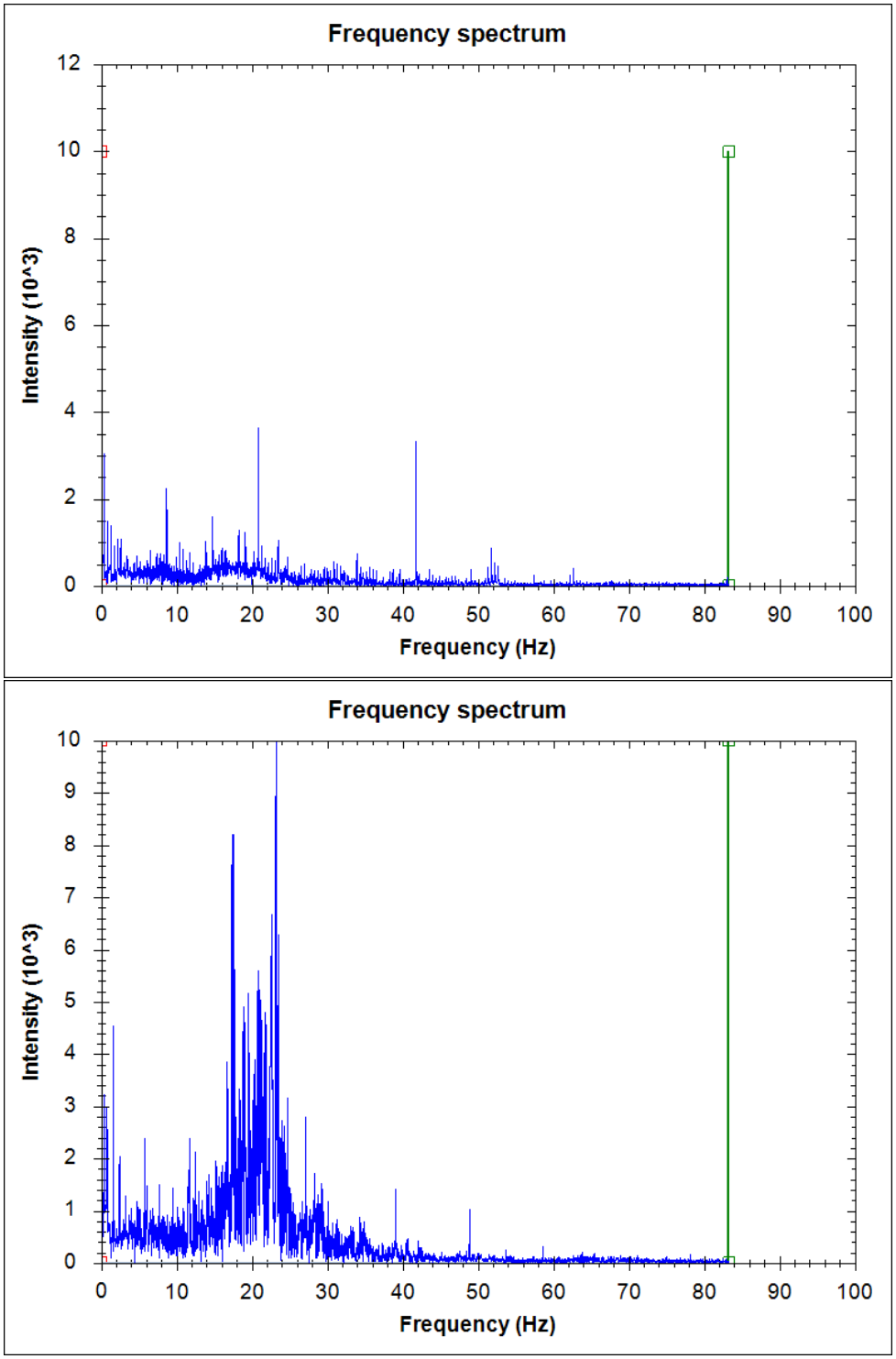


Figure 4.9: A spectral representation of the two signals in Figure 4.5 as seen in the visualization of the Fourier transformation node. The bad case has considerably more energy around 20 Hz.

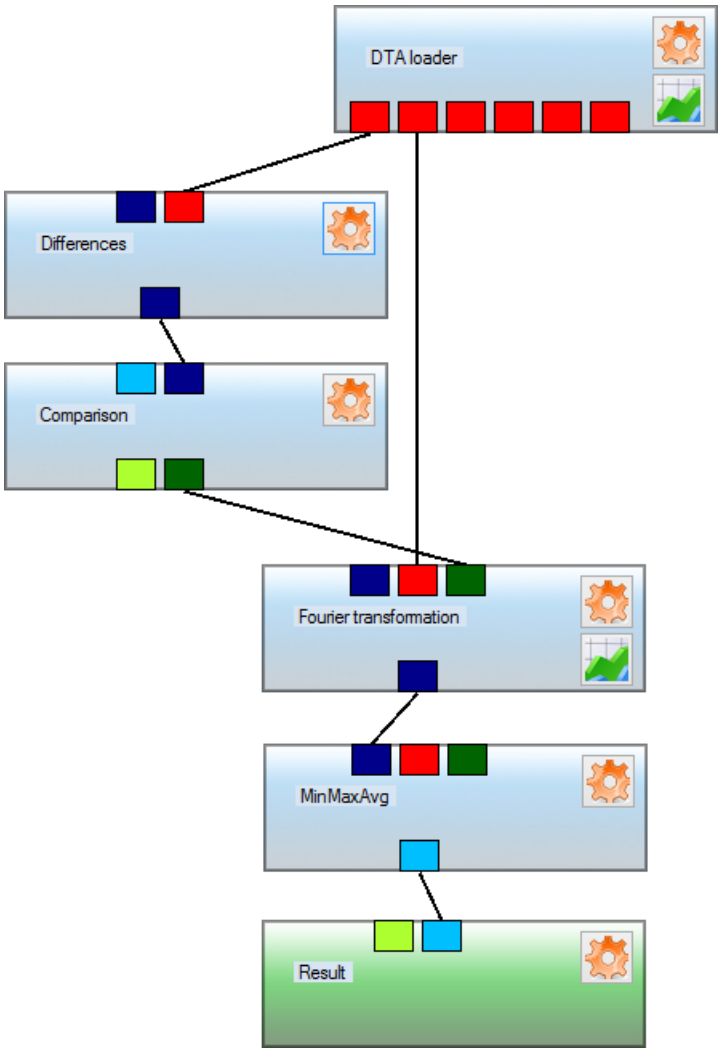


Figure 4.10: The final analysis graph, classifying test runs according to the average energy in the 16–32 Hz range of the position deviation, not regarding starting, stopping, and turnaround movements.

4.4 Results

The final analysis graph of the case study (Figure 4.10) was taken to a field test at the machine tool manufacturer. A first test used 40 test runs from the same ball screw scenario that was used in the test set of the case study. A domain expert provided pre-classifications for 20 of these test runs and the Result node selected its classification limits accordingly. After adding and classifying the remaining 20 test runs, the expert agreed with 19 of the automatic classifications. In the remaining case, the classifier marked a test run as ‘bad’ while the expert considered the plot to show only a minor problem. Looking at this test run’s spectrum revealed a high intensity in the 28–40 Hz range, which was not found in the other test runs, possibly indicating a difference in the configuration of this particular machine or an issue with the ball screw that is different from the defects seen in the other machines.

The manufacturer provided a second set of 100 test runs involving another ball screw component moving perpendicular to the first. After adjusting the classification limits in the Results node to account for a slightly different value range, the graph classified all of these test runs correctly. Additionally, one machine, which the expert expected to fail very soon, was scored at 0.1, another, which the expert said was having a minor problem, was scored at 0.9. This indicates that the fuzzy classification of borderline cases can indeed reflect both the fact that a case belongs to neither class and a tendency to be closer to one class than the other.

After completing this analysis, the domain expert was asked for an assessment of the utility of the approach, especially when compared to the current method of mostly manual evaluation. He stated that his impression was ‘rather positive’ and that the implemented prototype ‘illustrates an interesting way of how this kind of data can be analysed in the future’. He also said that ‘more experiments are necessary to visualize and analyse a much larger number of measurements’ and that he can imagine his company analysing about 1,000 measurements per year on 10 machine components with this approach.

5

Transportation Data Exploration

Chapter 3 discussed visual analytics on an implicit data space while the system in Chapter 4 operated on an explicit data set. This data set was small enough to not require a sophisticated data handling scheme. However, visual analytics specifically targets applications in which the data volume is well beyond what can be handled manually, so ways must be found to efficiently handle very large data sets in an interactive system. On the one hand, this requires the visualization and analysis components to use an out-of-core approach, which does not require all of the data to be in memory at all times. On the other hand, special care must be taken when designing the user interface, as interactivity tends to suffer when the size of the data being processed leads an application to exhaust the available system resources. This chapter discusses how a visual analytics system can approach both of these issues on a massive real-world data set from the public transport domain.

In *Illuminating the Path*, Thomas and Cook (2005) listed five major scale issues to be addressed by visual analytics systems: Information scalability (extract relevant information from a large amount of data), visual scalability

Parts of this chapter have previously been published in:

Wörner, M. and Ertl, T. (2012), 'Visual Analysis of Public Transport Vehicle Movement', in *EuroVA 2012: International Workshop on Visual Analytics* (Goslar: Eurographics Association), 79–83;

Wörner, M. and Ertl, T. (2014), 'Retaining Interactivity in a Visual Analytics System for Massive Public Transportation Data Sets', in *Proceedings of the 47th Hawaii International Conference on System Sciences – HICSS 2014* (IEEE), to appear ©2014 IEEE.

(effectively display a large amount of data to the user), display scalability (scale from mobile devices to room-filling screens), human scalability (handle multiple users), and software scalability (provide software that can handle large amounts of data in an interactive way). This chapter addresses the first and the last of these scale issues.

5.1 Requirements for interactivity

One way to ensure that a system handling large volumes of data remains interactive is providing sufficient hardware. However, when the cost of additional hardware is prohibitive or the analysis is considered an auxiliary task that does not warrant the necessary investment, a system will not scale and processing times will increase with the amount of data to be processed. If the user interface is tightly integrated with the analysis or visualization processes, this will result in a system that stops responding whenever a complex analysis or visualization task needs to be performed. Leaving the user without visual feedback or information on the progress of the task greatly reduces the usability and usefulness of a system. Visual analytics often involves following a certain line of thought and instead of regularly blocking the user from following this line, a system should allow continuous interaction, with the user interface acting as the user's agent that passes along commands and requests and reports back with status reports and results. More specifically, if real-time processing is not possible, four central requirements need to be addressed to ensure the usability of a visual analytics system:

1. Any time-consuming tasks should be computed in the background and not interfere with the responsiveness of the user interface. The user should be able to work with the system while waiting for an operation to complete.

2. A user should be able to view intermediate results of an analysis or visualization if these can serve as an early approximation of the final result. These preliminary results can be used to assess early on whether a task is likely to produce the intended final result. If it is not, cancelling the task or changing parameters can save the user from waiting a long time for an undesired result. To avoid any confusion, intermediate results should be clearly marked as preliminary (and therefore possibly incomplete, inconsistent, or incorrect).

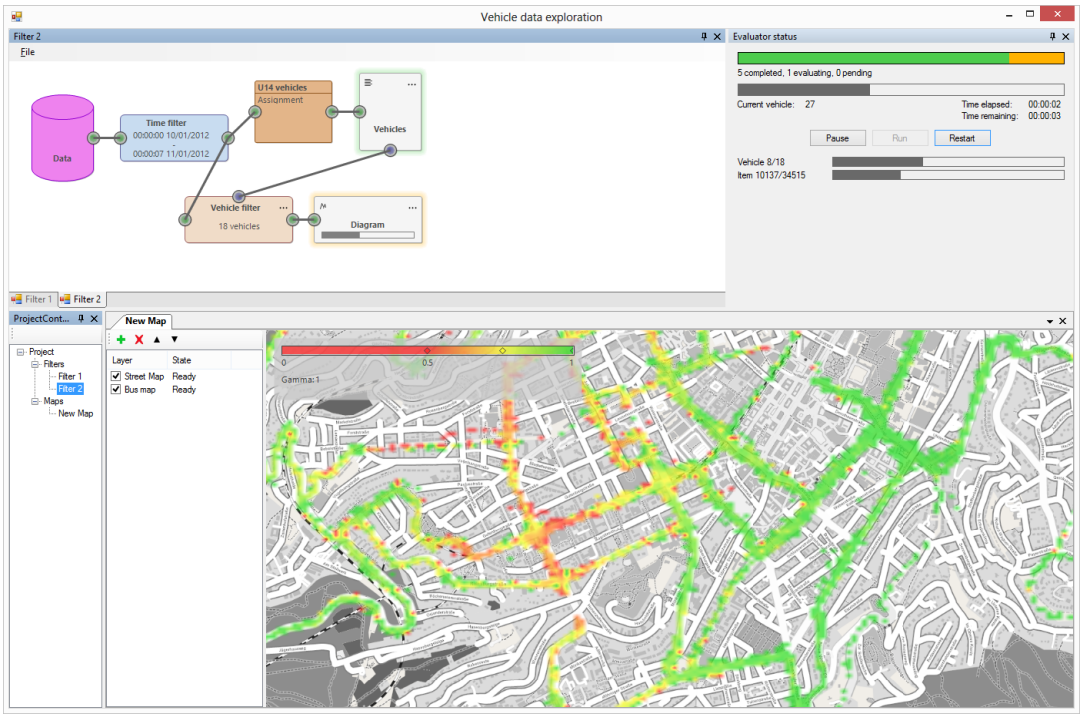


Figure 5.1: The implementation of the *Frida* framework analysing a public transport vehicle data set.

3. Where an estimation is possible, a user should be able to view a progress indication to decide whether it is worthwhile to wait for the completion of the operation. In combination with intermediate results, an analysis question might have been answered with reasonable confidence even before all data items have been evaluated. If the operation progresses very slowly, a user might choose to change parameters to simplify the query.

4. Available resources on the machine should not limit the total size of the data volume that can be analysed, even though they may limit the size of any (intermediate) results and thus how detailed an analysis may be.

This chapter introduces a framework and implementation for visual analytics applications having to handle large data volumes (Figure 5.1). Section 5.4 describes how this framework ensures that these requirements are met during all steps of the visualization pipeline (Chi, 2000; Haber and McNabb, 1990): data analysis, filtering, mapping, and rendering. Section 5.5 provides more details on the implementation and Section 5.6 describes the application of the framework in a visual analytics system for the analysis of a large vehicle data set from the public transport domain. For

easier reference, the framework will be referred to as *Frida* ('framework for interactive data analysis') in the following.

5.2 Related work

Frida is designed to manage various different analysis and visualization methods. In this aspect, it is similar to generic and modular visualization tools such as COVISE (Wierse, 1995) and ParaView (Cedilnik et al., 2006). ParaView's focus is on its client/server architecture, not on continuously providing intermediate results and progress reports. Instead, it renders a less detailed preview during user interaction and a full detail image when interaction stops.

SimVis (Doleisch, 2007) is a system for the interactive visual analysis of data from a fluid dynamics simulation using multiple data views. Its users can interactively assign degree-of-interest values to parts of the data and can interactively and iteratively define features they are interested in using either a feature definition language or direct manipulation of the specification. *Frida* focuses on a more explicit separation of analysis and visualization.

ATLAS (Chan et al., 2008) is a visualization tool for temporal data that maintains interactivity when panning and zooming through large data sets by predicting possible user actions and fetching data required to react to these actions in advance, hiding the actual latency between requesting data and seeing the visual representation. While this approach requires that the number of probable user actions is limited, some views in visual analytics system fulfil this requirement, and prediction interactions might be a valuable tool in improving the overall system responsiveness.

Piringer et al. (2009) presented a multi-threaded architecture to address the need for responsive visual interfaces when dealing with complex visualizations that potentially incur a significant latency. In their architecture, the thread handling user events may only perform operations that can be completed quickly enough to not cause a noticeable delay. The actual visualization work is done in separate threads (one thread per view) and the event handling thread communicates with the visualization threads using state variables. The visualization threads are expected to support early thread termination, stopping and restarting their work shortly after any significant parameter change. *Frida*'s approach is similar, but it assumes a visual analytics system with a separate representation of analysis and visualization. A user can model the required analysis steps, watch their individual progress, and request visualizations of the (possibly partial) results. Users are free to change the model while the analysis is in progress.

There is a central evaluation thread that handles communication with the user interface thread, reading the data required for the current analysis, and distributing the data items to a set of threads processing the data. At any time, the user can start visualization threads to view (possibly partial) analysis results.

Young and Shneiderman (1993) have previously advocated the approach of using a graph representation to create a user-friendly interface for what are basically Boolean expressions. Like the system in Chapter 4, *Frida* uses a graph-based model of the analysis process similar to that of KNIME (Berthold et al., 2009). KNIME uses a generalized, table-based data model and evaluates data paths one node at a time, passing data tables along from one step to the next. Nodes in parallel paths can be evaluated simultaneously. *Frida* uses a streaming model and passes individual data items along a path. Unless analysis nodes require a complete result set, they can generate intermediate results and visualizations before previous steps have been completed. It also aggregates filters along the filter path to simplify and accelerate the initial data query. When generating visualizations, KNIME does not offer intermediate results, progress indications or a way to cancel the generation. The user can continue to work with the analysis graph while the visualization is being generated, but when changes are made in the path leading to the visualization node, the application will not accept any more user input until the visualization is complete. In *Frida*, visualizations can be created as soon as the evaluation has been started and will regularly update themselves until the analysis is complete or cancelled.

Section 5.6 demonstrates the framework in an analysis system for the public transport domain. Andrienko and Andrienko (2013) published a comprehensive overview of visual analytics approaches for movement data and divide research in this area into the categories of looking at trajectories, looking inside trajectories (i.e. movement characteristics along trajectories), having a bird's-eye view on the movement, and investigating movement in context. von Ferber et al. (2009) analysed the public transport infrastructure of major cities in graph representations, but did not regard the movement of individual vehicles or intended to develop an interactive system. On a much larger scale, visual analytics has also been applied to intercontinental shipping routes (Lundblad et al., 2008).

5.3 Vehicle data

Frida was created to analyse a massive vehicle data set provided by Stuttgarter Straßenbahnen AG. The corresponding data specification is courtesy of Trapeze Switzerland GmbH. While the framework is not limited to the

public transport domain, the data set influenced some basic assumptions made on the data to be processed: *Frida* assumes the data to be a set of entities, each with one or more time-based sequences of multivariate states and specific events. In the transportation case, these entities are vehicles (buses or trams). While in operation, a vehicle continuously records performance figures such as its current speed, odometer value, and GPS position once every couple of seconds (the multivariate states). It will also record events such as stopping at a station, opening or closing doors, and sending or receiving a radio transmission.

Additional data tables contain infrastructure data specifying the location of all stations and the geometry of the rail track network. Also, there are route pattern tables, which state which sequence of stops make up a route pattern, and route tables, which state which patterns make up a route. The vehicle data covers about 350 active vehicles, serving 1,000 stations on 90 routes. The recording spans one month and amounts to about 380 million data items. This data is stored in just over 19,000 compressed text files with a total size of about 7 GiB. The uncompressed size of the data cannot easily be measured because some of files are corrupt and decompress to huge files of mostly random data. Based on the compression ratio of the working files, however, the total uncompressed size can be estimated to be about 44 GiB.

There are other inconsistencies in the data. A considerable number of files appear to be truncated and the checksum test of the compression format fails. For most of these files, however, the extracted content still ends with a ‘data collection complete’ event, so no relevant data is missing. In addition to invalid lines of data caused by an incomplete compressed file, some correctly extracted lines specify invalid dates from the 1980s or invalid vehicle numbers, most likely caused by an invalid configuration or missing initialization of the vehicle. All in all, 89 % of the data files passed an automated consistency check.

5.4 The framework

Similar to the system in Chapter 4, *Frida* uses a data flow graph to model an analysis task. Figure 5.2 shows the main components of the framework. A **data source** supplies data items to the **filter nodes** of the data flow graph. These forward data items (or possibly only a selection or a transformed version) to **consumer nodes**, which create analysis **results**. The user can create different **visualizations** of these results. The **evaluator** component controls the evaluation of the data flow graph. The following sections describe these elements in more detail.

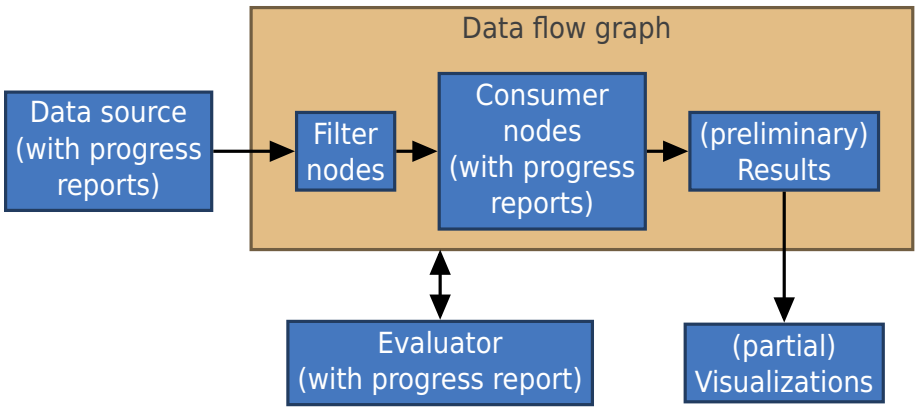


Figure 5.2: An overview of the framework.

5.4.1 Handling the data

Frida targets typical workstation machines. Therefore, reading a massive data set into memory in its entirety before starting the analysis is not an option. Instead, a disk-based out-of-core solution for efficient access to the data is required. Fekete has discussed different approaches at data management for visual analytics (Keim et al., 2010). Working off the original files would be the flat files approach. This would allow analysing the data without requiring any kind of preprocessing but has obvious drawbacks such as having to either decompress on demand or invest a lot of disk space, having to parse a lot of textual number representations, and having to read through a lot of data to find, for instance, a given point in time.

A more sophisticated approach is to import the data into a relational database system. These systems are highly optimized for the fast evaluation of data queries and handle large data volumes. They also support the creation of indices for a more efficient search for certain attribute values. However, relational databases are generally optimized for inserting, updating, and querying interrelated data items in a dynamic data set, whereas visual analytics, as in the public transport case, often operates on a static data set and therefore depends on reasonable import and querying performance but does not generally update data items nor add new items after the initial import. Also, *Frida* assumes a data set with no significant interrelations other than all data items being related to a specific entity and point in time.

The NoSQL (‘not only SQL’) approach may be more suitable for this kind of data. NoSQL systems are non-relational databases commonly employed when a large volume of data, not necessarily following a strict structure, is to be primarily queried with only occasional changes to the data set (Cattell, 2011). They are typically optimized for retrieving data by a key

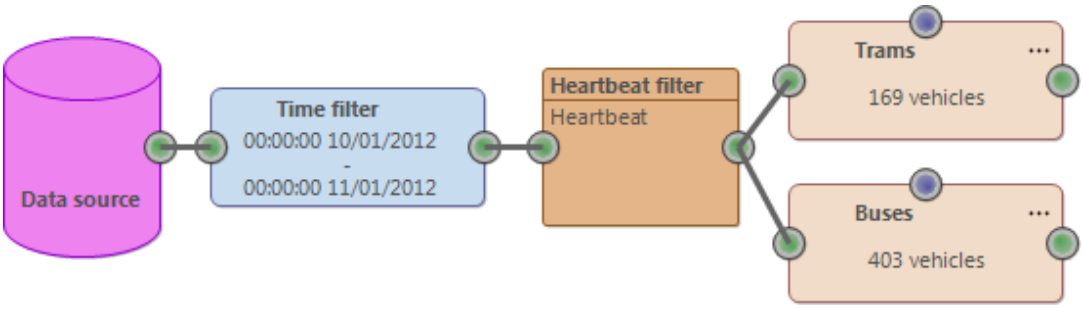


Figure 5.3: A filter graph as displayed by the system with a data source, a time filter, a data sequence filter, and two entity filters for trams and buses (Wörner and Ertl, 2014). ©2014 IEEE

value and for appending data.

For handling the vehicle data set, importing partial sets of 2, 10, 20, and 30 vehicles into two relational databases (PostgreSQL, SQLite) and one non-relational database (MongoDB) showed that for all these systems, the average time it takes to import a data item increases with the total number of data items, this is, the import time is not linear in the number of items. By extrapolation on the best case, the workstation machine used to implement *Frida* would take at least 32 hours to import the required data. For an application such as the public transport case, this creates a significant delay between collecting the data and being able to start the analysis. As a consequence, *Frida* uses a less sophisticated system that parses the text files and stores binary values in a number of files, one for each data sequence and entity. To allow for efficient access by time, the data items are sorted by their timestamp. Also, a hash-based index file is created, by which the most recent item for a given minute can quickly be found. From there, a binary search leads to the exact item. Importing the vehicle data set into this system took just over 3 hours.

5.4.2 Filtering

Once data preparation is complete, data items can be filtered based on whether they are relevant to the question at hand. An analyst will most commonly restrict a query to a certain time interval, a certain subset of entities, and certain data sequences. Additionally, there may be conditions on data field values. In *Frida*, an analyst creates a visual representation of this process in the data flow graph (Figure 5.3). Every data flow graph contains a node representing the data source as the entirety of all data items. Starting from there, subsets of data items can be defined by adding and connecting filter nodes. All filter nodes have data stream inputs and

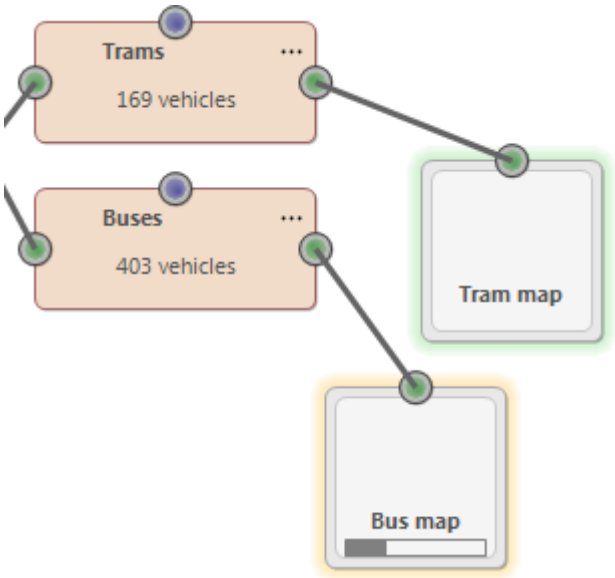


Figure 5.4: An evaluated and an evaluating consumer node (Wörner and Ertl, 2014). ©2014 IEEE

outputs (displayed as green circles).

A **time filter** node restricts the time span of data items to be processed.

A **entity filter** node restricts the set of entities to be processed. The user can manually select the entities to be passed through or connect the node's list input (blue circle) to a consumer node providing a list of entities. This can be used to define multi-stage analyses, in which the set of entities to be processed is the result of a previous analysis step.

A **data sequence filter** node restricts passing items to those of a specified data sequence, optionally also restricting certain data attribute values.

In addition to the data source node, the graph in Figure 5.3 contains a time filter node, a filter node that selects items from the 'Heartbeat' data sequence and two entity filter nodes, which each select a subset of entities.

5.4.3 Mapping

Mapping is the step that transforms the filtered data items into the quantities that are to be visualized. In *Frida*, an analyst defines mapping operations by adding consumer nodes to the data flow graph. These nodes consume data items streamed into them to generate an analysis result such as a diagram. In Figure 5.4, two map layer consumer nodes have been added, one for each of the entity filters in (Figure 5.3). Map layer consumer nodes take a number of parameters: the definition of a geographic region of

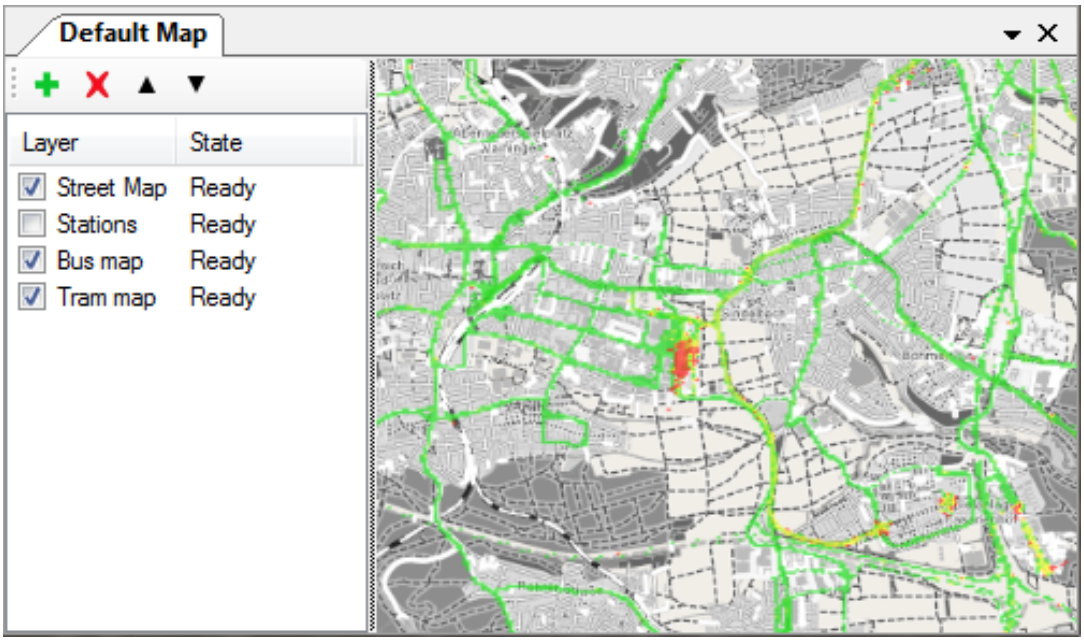


Figure 5.5: The map view plotting the radio coverage of buses and trams over a street map (Wörner and Ertl, 2014). ©2014 IEEE

interest, a resolution, the data attribute to be mapped, and an aggregation method. Limiting the covered area and the spatial resolution at which measurements are aggregated ensures that the memory requirements of this mapping step are independent of the number of data items to be processed. This is in accordance with the fourth requirement in Section 5.1.

Coloured halos indicate the state of a consumer node. The possible states are: disabled (this consumer will not be evaluated), pending (this consumer is scheduled for evaluation), evaluating (this consumer is currently being evaluated and might be able to present intermediate results), and evaluated (evaluation for this consumer has finished and the final analysis result is available). While a consumer node is being evaluated, it displays a progress bar with an estimation of the progress of its analysis (Figure 5.4). Consumer nodes can use a list output port (blue circle) to provide a result set to the list input port of filter nodes to construct multi-stage analyses.

5.4.4 Rendering

In the *Frida* framework, rendering a visualization of analysis results does not necessarily have to wait for previous analysis steps to complete. Instead, a user can request a visualization of a consumer node at any time during the analysis process and this visualization will update itself at regular intervals

to reflect the analysis progress. Some visualizations can be generated very quickly while in other cases, the generation of the visual representation itself incurs a significant delay. To ensure the continued responsiveness of the system, these long-running renderings have to be performed in a background thread. Rendering a complex multi-layer map is an example of one such visualization.

The results of a map layer consumer node can be visualized as a layer in a map view (Figure 5.5). This view shows a set of map layers that render an image of a given region at a given resolution. Different types of layers display different information in their rendered image. The current implementation includes two layer types that provide geographic reference information: One layer renders a street map based on data from the OpenStreetMap project (Haklay and Weber, 2008). This is a general geographic reference useful for any kind of geospatial data. The second reference layer is more specific to the example domain. It plots the location of bus and tram stops along with text labels identifying the stop. In addition to these static layers, a user can add layers that display geo-referenced analysis results from a consumer node. The map layer consumer node aggregates data values into a grid of cells. The corresponding map layer samples this grid using bilinear interpolation and maps the result to a user-defined colour palette. Using another bilinear interpolation for the layer's alpha channel, empty cells are mapped to transparent pixels.

Whenever the map view needs to be painted, the map will determine the geographic region visible in the view and start a background thread that renders this region for all map layers in turn. It will then composite the separate images and display the resulting image. Repaints can be triggered both by user actions, such as panning or zooming the map, and by data updates sent by consumers that are still being evaluated. Panning and zooming affect the map as a whole and trigger a repaint for all layers. While the map is rendering, the map view provides immediate feedback by drawing a translated and scaled version of the previous map composition as an early approximation of the final result. Some events, such as the user changing a layer's colour palette or the regular updates of a layer during its computation, only affect a single layer. In these cases, the information on other layer does not need to be updated and the map view creates a new composition by combining the updated layer images with cached images of the other layers. Since all layer images are stored separately, a user can make layers visible or invisible or change the layer opacity without triggering a rerendering of the layers. The user can also reorder map layers (higher layers paint over lower layers). The map view includes a list of all layers and their current states, by which the user can check the render progress.

5.4.5 Analysis feedback

A central element of visual analytics systems is the ability to use the interactive visual interface to provide feedback to the analysis engine. Ideally, an analyst can interact with the visualization directly to influence the corresponding analysis parameters. *Frida* supports this in that an interaction with a visualization can change the parameters of the corresponding consumer node. Panning the map, for example, will cause the node to start a new analysis to compute a view for the newly visible area. Visualizations can also be used to specify conditions for filter nodes: A geospatial restriction can be created by making a selection in a map view. An analyst can also click an element in a pie chart to add a data sequence filter node that selects items with the corresponding attribute value.

5.5 Graph evaluation

Before the results can be rendered as a map layer or any other kind of visualization, the data flow graph needs to be evaluated. In accordance with the first requirement in Section 5.1, *Frida* performs this evaluation in a background thread that runs as soon as the user makes a change to the data flow graph. The evaluation thread executes the following loop:

```
while not cancelled do
    update the pending consumers;
    if no pending consumers then
        break;
    end
    find active paths;
    remove consumers without an active path;
    aggregate all simple filters;
    query the data source;
    while there are evaluating consumers and not cancelled do
        read next item;
        if no more items then
            break;
        end
        distribute item along active paths;
        handle completed and cancelled consumers;
        update active paths and simple filters;
    end
end
```

The first step is to determine which consumers are in the ‘pending’ state. A consumer node is pending if it is to be evaluated, has not yet been evaluated, and does not depend on other consumer nodes that have not yet been evaluated. A dependency exists if one of the filter nodes in a consumer’s path is connected to another consumer via its list input port and thus requires that this consumer has completed its evaluation before it can filter data items.

Next, evaluator changes the state of all pending consumers to ‘evaluating’ and determines the active paths. An active path is a filter path that leads to an evaluating consumer. Evaluating consumers without a path complete their evaluation instantly (and reach the ‘evaluated’ state) due to lack of input data. The next step is to build an appropriate query to the data source. The way the data is arranged (as described in Section 5.4.1), it can be efficiently accessed by time, entity, and data sequence. Filter conditions that refer to these properties are considered simple filters. The evaluator computes the intersection of all simple filter constraints along a path and the union of the constraints of all paths to determine which items need to be read from the data source. Constraints that are not simple in this sense need to be checked by enumerating through the remaining items.

The evaluator now starts reading items from the data source. Each item is passed along all active filter paths. Path elements can alter an item, for instance by replacing a complex multivariate data item with a single numerical value that represents just one specific attribute. An element can also discard an item based on its properties, which will end the path evaluation for this item. To ensure that changes do not affect items travelling across parallel paths, each path works on a copy of the original data item. Segments shared by multiple paths are evaluated only once and if the path splits again, the separate paths continue on their own copy of the result. At the end of a path, the consumer node processes the item. A map layer, for example, might perform the necessary aggregation into a cell grid.

A consumer node can cancel its own evaluation and indicate that it does not need to see any more data items to complete its analysis. It can also be cancelled by the user disabling its evaluation. These consumers are removed from the set of evaluating consumers (changing their state to ‘evaluated’) and their paths from the set of active paths. An update of the simple filter conditions determines whether certain time spans, entities, or data sequences can now be skipped because they are not required by any of the other paths.

Reading and distributing data items stops when there are no evaluating consumers left, all data items have been processed, or the user cancels the evaluation as a whole. Repeating the cycle, the evaluator again checks

for pending consumers, which may now have all necessary dependencies met, may have recently had their evaluation enabled, or may have been invalidated by changes to the analysis graph, prompting their reevaluation. Once no pending consumers remain, the evaluation thread goes idle and waits for new pending consumers to appear.

5.5.1 Handling graph changes

In accordance with the first requirement, the user can continue to work with the system while an evaluation is being performed. This includes the possibility of making changes to a data flow graph that is currently being evaluated. When such changes are made, care must be taken to handle potential conflicts and ensure consistent results, restarting computations when necessary. On the other hand, the system should not overreact, discarding the partial results of a long-running computation just because the user made some cosmetic change to the graph. *Frida*'s general approach is to continue an evaluation as long as it is feasible but exclude all consumers whose results may have become invalid. Consumers with affected paths are removed from the set of evaluating consumers. Since they have not completed their evaluation, they will become pending consumers at the start of the next evaluation cycle. Should no evaluating consumers remain, the current cycle terminates early.

Events that may invalidate a partial computation are: Adding or removing nodes, adding or removing edges, and changing the parameters of a node. Adding a new node will not invalidate any current results but may introduce a new, pending consumer to be evaluated later. Removing a node will invalidate all consumers whose paths include this node. Adding or removing edges will invalidate all consumers whose paths include the added or removed edge. Changing the parameters of a consumer will invalidate this consumer and changing the parameters of a filter node will invalidate all consumers whose paths include this node.

When a consumer becomes invalid, *Frida*'s default response is to continue the evaluation for all consumers that are still valid, then start a new evaluation for any invalidated consumers. At the start of the next cycle, the evaluation thread may determine that these invalidated consumers operate on a different time span or entity set or a different set of data sequences than the evaluated consumers and simplify the query to the data source accordingly. If an evaluation has not yet progressed very far, a user may prefer to start over immediately, so *Frida* provides an option to discard all partial evaluation results of a graph and trigger an immediate reevaluation of all incomplete consumers. An analyst can also structure a complex analysis by splitting it into separate graphs. While the evaluation

thread will not differentiate between graphs and construct a single data source query to satisfy all active consumers, the option to enable or disable the evaluation of a graph enables the user to direct the computational resources of the system to specific analyses. For more fine-grained control, the evaluation of single consumers can be enabled or disabled as well.

5.5.2 Thread communication

In multi-threaded systems, thread communication and synchronization are frequent causes for non-deterministic behaviour and deadlocks (Lee, 2006). Multi-threading can ensure the responsiveness of the user interface only if the communication between the user interface thread and the worker threads causes no significant delays. In *Frida*, thread communication is needed between the interactive visualization of the analysis graph and the evaluation thread as well as between visualizations of analysis results and the corresponding consumer nodes. Synchronization is handled differently in these two cases. The consumer visualizations are meant to show intermediate results of the analysis as it progresses. If the consumer node is busy, the visualization can easily wait until new data is available and continue to show the previous update instead. The consumer node needs to be aware of the visualization only in that it must make sure not to delete data that is currently being read. For this type of communication, *Frida* uses a light-weight synchronization using state variables.

The visual representation of the analysis graph, on the other hand, is not simply a display showing the current state but also allows making changes to the analysis model that is the basis for the evaluation thread's processing. To ensure a correct analysis result, the evaluation thread must review all changes to this model and determine their consequences as described in Section 5.5.1. To avoid having to wait for a confirmation on every user interaction, *Frida* handles this communication using a message queue. The user interface thread keeps a representation of the data flow graph for display whereas the evaluation thread keeps a list of all consumers and their paths. When the user makes changes to the graph, the user interface posts the details of this change to the message queue, while immediately showing the visual result of the change and awaiting further adjustments. The evaluator thread regularly checks the queue and updates its model accordingly. Consequently, threads only block for the time it takes to read from or write to the queue and the user can make changes to the graph without having to wait for the evaluation to react to these changes. Feedback is provided in that these changes will eventually be reflected in consumer states, progress indications, or the overall progress report.

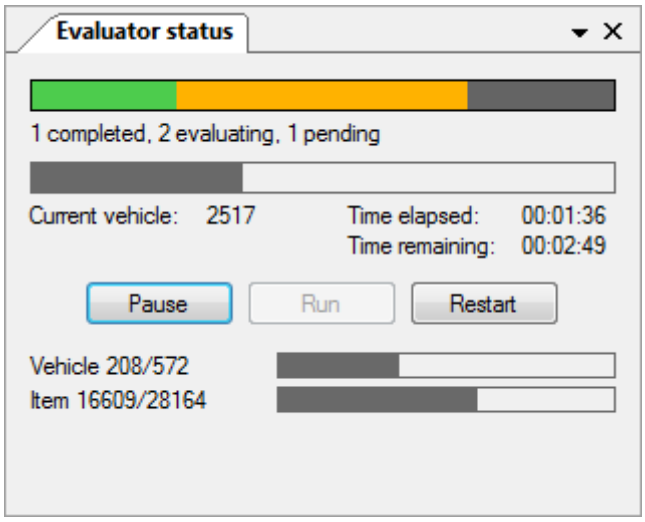


Figure 5.6: The evaluator status view. The coloured bar shows the distribution of consumer node states (Wörner and Ertl, 2014). ©2014 IEEE

5.5.3 Intermediate results and progress reports

The evaluation process is potentially very time-consuming. To meet the second requirement in Section 5.1, consumer nodes are expected to provide intermediate analysis results even before having seen all relevant data items. This can be achieved for the map layers (by supplying the current state of the aggregation grid), for consumers that are preparing a result list (by supplying the current state of the list), for consumers that count an attribute value distribution (by supplying the distribution among all data items seen so far), and for consumers preparing a diagram of data items (by adding more items to the diagram as they become available, progressively extending or refining the diagram). As described in Section 5.4.4, a user can create visualizations for all these consumers and view the current state of their results, with regular updates.

The third requirement calls for a progress estimation. *Frida* tracks the progress of the analysis by counting the consumer nodes in the various states and by giving an estimation for the current analysis cycle based on how many entities have been processed so far and for the current entity based on how many data items have been processed. In addition to the progress indicators shown on the consumer nodes (Figure 5.4), there is a separate view that shows this information along with an estimate of the remaining time (Figure 5.6). This window also includes manual controls to pause, continue, and restart the evaluation process.

Visualizations are rendered independently of each other, so each handles its own process indication: A pie chart diagram can be created instantly

and does not need to generate progress reports. Rendering a multi-layer map, on the other hand, can be complex, so the map view lists the current state and possibly progress of rendering each layer separately. The diagram view, which renders potentially complex diagrams of vehicle data, includes a progress bar at the bottom.

The data source requires the index files to be loaded into memory before any queries can be processed. During this early phase, the data source nodes in the data flow graphs show a progress bar that illustrates the index loading progress. The user can start to construct or modify data flow graphs while the indices are being loaded. Evaluation of any completed graphs will then start automatically once loading is complete.

5.6 Application to public transport

This section discusses three examples of analysis questions and how they can be answered by using *Frida* on the data set described in Section 5.3. The entities of this data set are the individual vehicles. Its data sequences include performance measurements (such as speed or location) and halt events. The analysis questions have been developed in collaboration with the domain experts from Stuttgarter Straßenbahnen AG. These questions are:

1. How good or bad is the vehicle radio coverage for trams and buses across the network (geospatial analysis)?

2. Is the percentage of failed radio transmissions higher in downtown than overall (statistical analysis)?

3. What is the average velocity curve of vehicles serving a given route? Where on that route do vehicles significantly deviate from this average (diagram generation)?

The first two questions can be answered using the data flow graph in Figure 5.7 that processes the ‘heartbeat’ telegram events in the data set. While in operation, a vehicle regularly sends a radio heartbeat to check whether it has radio contact. It logs these attempts along with a flag indicating whether a response was received. To analyse the radio coverage, an analyst uses a time filter to select the time span to be examined and adds two entity filters to split the items into separate evaluations for buses and trams. Each entity filter forwards its items to a map layer consumer (‘Tram map’ and ‘Bus map’) and a chart consumer (‘Tram chart’ and ‘Bus chart’).

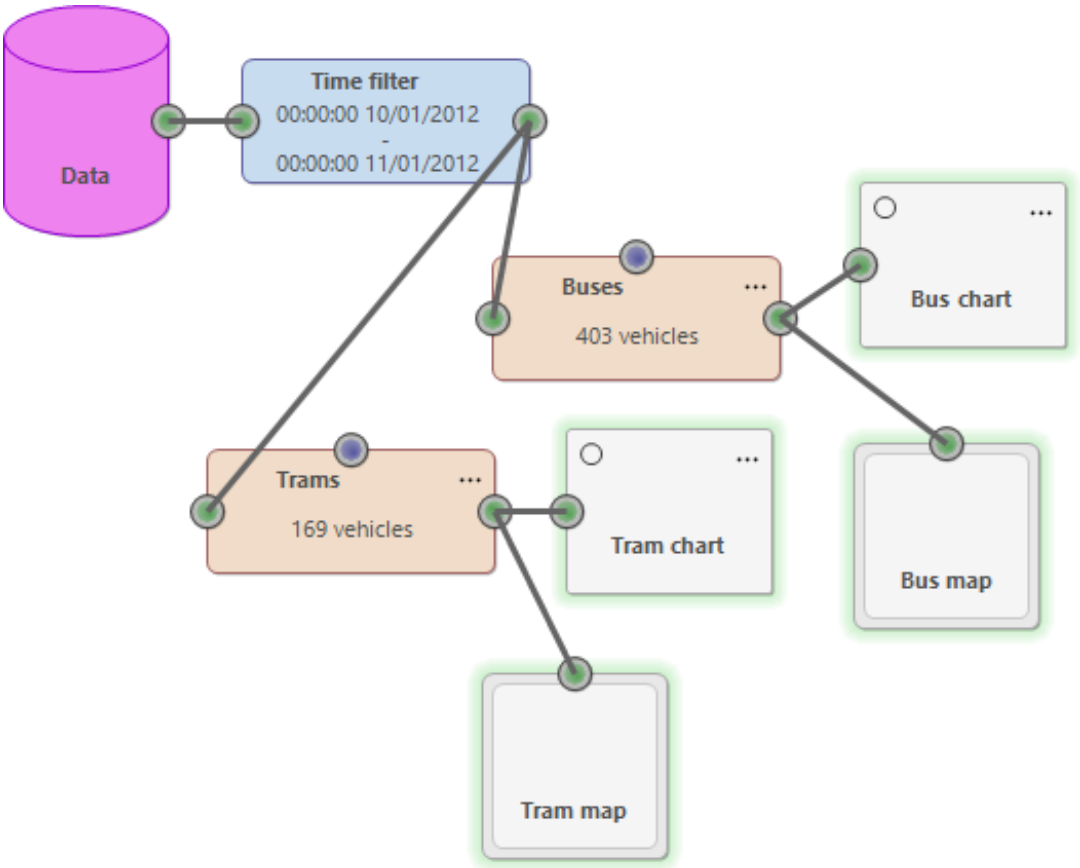


Figure 5.7: A data flow graph to answer the first two analysis questions.

The analyst can then use the map view to visually inspect and compare the radio coverage of trams and buses (Figure 5.8). In the figure, the opacity of the underlying street map layer has been reduced for better visibility. For both layers, the colour palette is set to interpolate from red at 50 % coverage through yellow at 75 % coverage to green at 100 % coverage. The final assessment of the situation will have to be left to the domain experts, but it is apparent that the bus coverage is generally very good except for the downtown area (top centre) and an area at the right of the image. Tram coverage is generally good, with noticeable red spots near depot locations (which may be due to vehicles starting up or shutting down and without practical relevance).

The second question can be answered by the chart consumers, which count successful and failed heartbeat communications across the entire network. The results can be visualized in a pie chart view (Figure 5.9). Overall, radio coverage is better for buses than for trams. To assess the severity of the apparent downtown dead zone, the analyst can draw a selection rectangle into the map view to create a filter node that selects

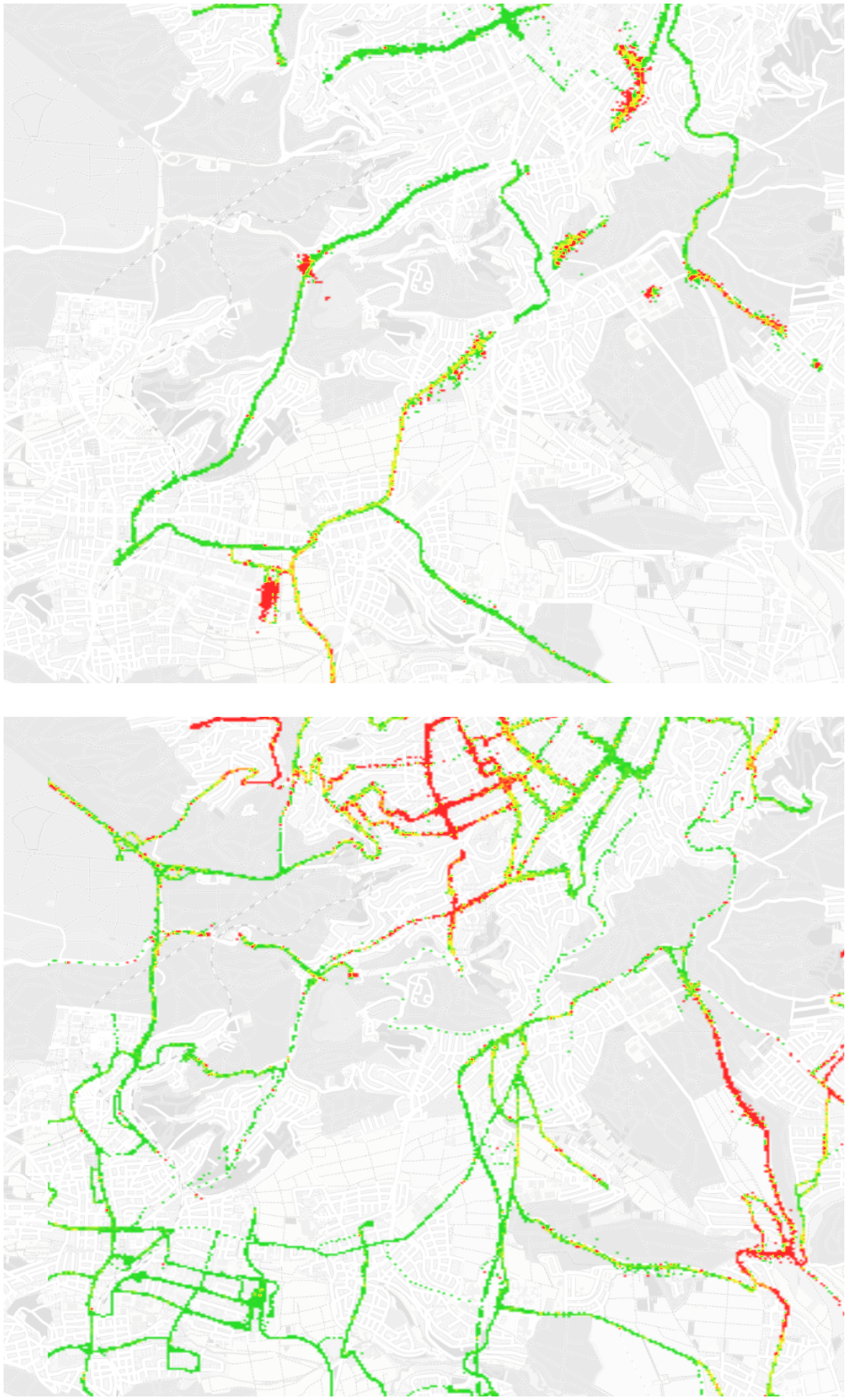


Figure 5.8: Comparing tram (top) and bus (bottom) radio coverage on the map view.

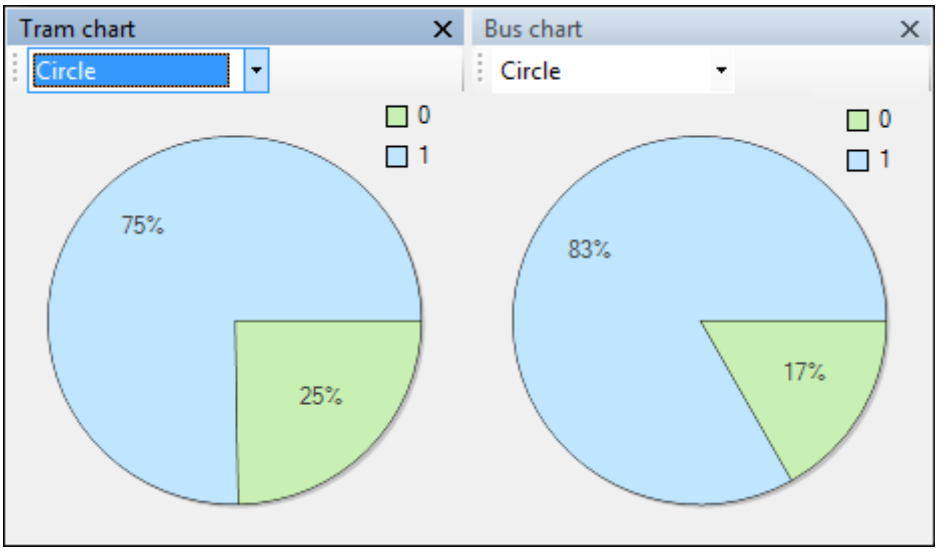


Figure 5.9: Comparing heartbeat response ratios for trams (left) and buses (right) (Wörner and Ertl, 2014). ©2014 IEEE

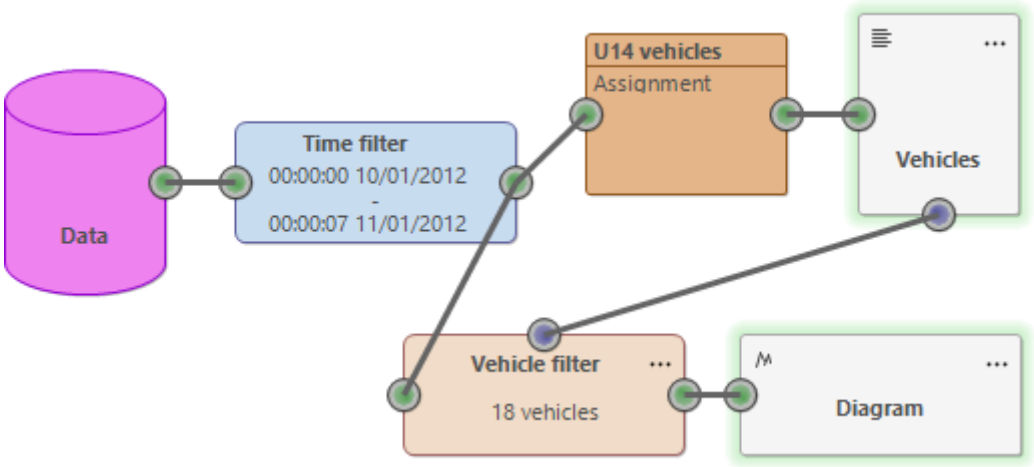


Figure 5.10: The third question can be answered using a two-stage analysis graph (Wörner and Ertl, 2014). ©2014 IEEE

only measurements from this area. Inserting this filter node into the data flow graph, a heartbeat failure ratio for just this region can be determined.

The third question involving the creation of a velocity diagram also asks for a restriction to a certain route. This can be achieved by using an entity filter node to forward only data items of vehicles serving a given route to the diagram consumer node. Which vehicles serve a route can be determined by a data sequence filter that checks route assignment events and compiles a list of vehicles on that route. This vehicle list can

be connected to the list input port of the entity filter to form a two-stage analysis process which first determines the set of relevant vehicles, then builds a diagram for just these vehicles (Figure 5.10).

5.6.1 Time and space

In *Frida*, data items are ordered by time and—disregarding data inconsistencies and resetting clocks—a vehicle can have only one state and be at only one location at any point in time. As a result, one can easily draw a diagram that shows the speed of a vehicle over time. Information on what route a vehicle was serving at a given time and whether it was within the stop zone of a station can be added to the diagram.

It is also possible to retrieve the GPS position of a vehicle, project that onto a two-dimensional map display and paint a track or animate a position indicator to visualize the vehicle movement in space and time. Public transport vehicles, however, do not usually move freely about a map but follow specific, predetermined routes. This is especially true for trains and trams, which are bound to follow their rail tracks, and it also holds for buses, which follow the roads from stop to stop. Therefore, one important alternative view of space is that of a strictly linear progression along a predetermined path. In these cases, diagrams that map speed or some other value to the travelled distance rather than a two- or three-dimensional location are much more meaningful than they would be for an arbitrary movement, as for different vehicles travelling along the same path, the same distance from some set point of origin refers to the same point in space. Overlaying the data curves of different vehicles by travelled distance rather than time can reveal peculiarities that are specific to a certain location.

In order to create this kind of diagram, data items must be accessed by travelled distance rather than time. The movement of a vehicle can be seen as a series of segments of travelling from one station to the next. The departure station is the point of origin and a vehicle should reach the destination station after travelling the distance between them. The diagram node receives and scans the vehicle performance data and, when a vehicle stops at a station, stores its odometer value. Once the odometer values at both ends of a segment are known, it is possible to map any position between the respective stations to a odometer value for the vehicle. Searching for a data item with this odometer value, one can find the corresponding time and use that to find any other related data items to determine, for example, the speed of the vehicle at that point.

While a vehicle will only have at most one velocity value for a given point in time, it will usually reach the same point in space multiple times

over the course of a day. Data sequences indexed by location therefore have multiple ‘runs’, multiple values per location belonging to multiple passes by the same vehicle. To account for the fact that there may be multiple paths between two stations, the diagram consumer node compares the distance actually travelled by the vehicle to the nominal station distance according to the infrastructure data and discards vehicles that seem to have deviated from the expected path.

5.6.2 The diagram view

The results of a diagram consumer node can be visualized in the diagram view (Figure 5.11). This view shows a graphical representation of one or more data sequences across a data range, which is either a time range or a route pattern range. For a time range, the view shows the development of one or more data sequences over time. Multiple vehicles and multiple data sequence types (such as current speed and halt events) can be displayed simultaneously, but there will never be more than one value for any combination of vehicle, data sequence type, and time. Multiple sequences can either be overlaid, which is useful, for example, for comparing the speed graphs of several vehicles, or stacked, which is useful, for example, for getting an overview over which vehicles serve which routes over the course of a day (Figure 5.11 bottom).

A route pattern range (as used in Figure 5.11 top and centre) is defined by a sequence of pattern segments, each specifying a departure and an arrival station. Data items can be mapped to this linearized spatial axis with the method described in Section 5.6.1. Unlike a time range, a route pattern range can contain multiple data values for a given vehicle and position. Plotting the speed of a single vehicle will usually result in a number of overlapping graphs, which can be used to compare multiple trips along the same route.

The user can enable the display of an average graph, which shows the average speed of the plotted vehicles at a certain location. It is easy to spot which vehicles travelled above and below the average speed. Additionally, the standard deviation can be added, overlaying the vehicle graphs with a range of common values. The average and standard deviation allow for some interesting observations: Local conditions that affect all vehicles, such as tight corners or general speed limits, can be seen in the graph of the average speeds (Figure 5.12 top). Conditions that affect a considerable number of vehicles, such as intersections or traffic lights, which may or may not force a vehicle to slow down or stop, cause visible bulges in the standard deviation tube (Figure 5.12 bottom left). If single vehicles deviate from the common path, these can be identified as outliers which create

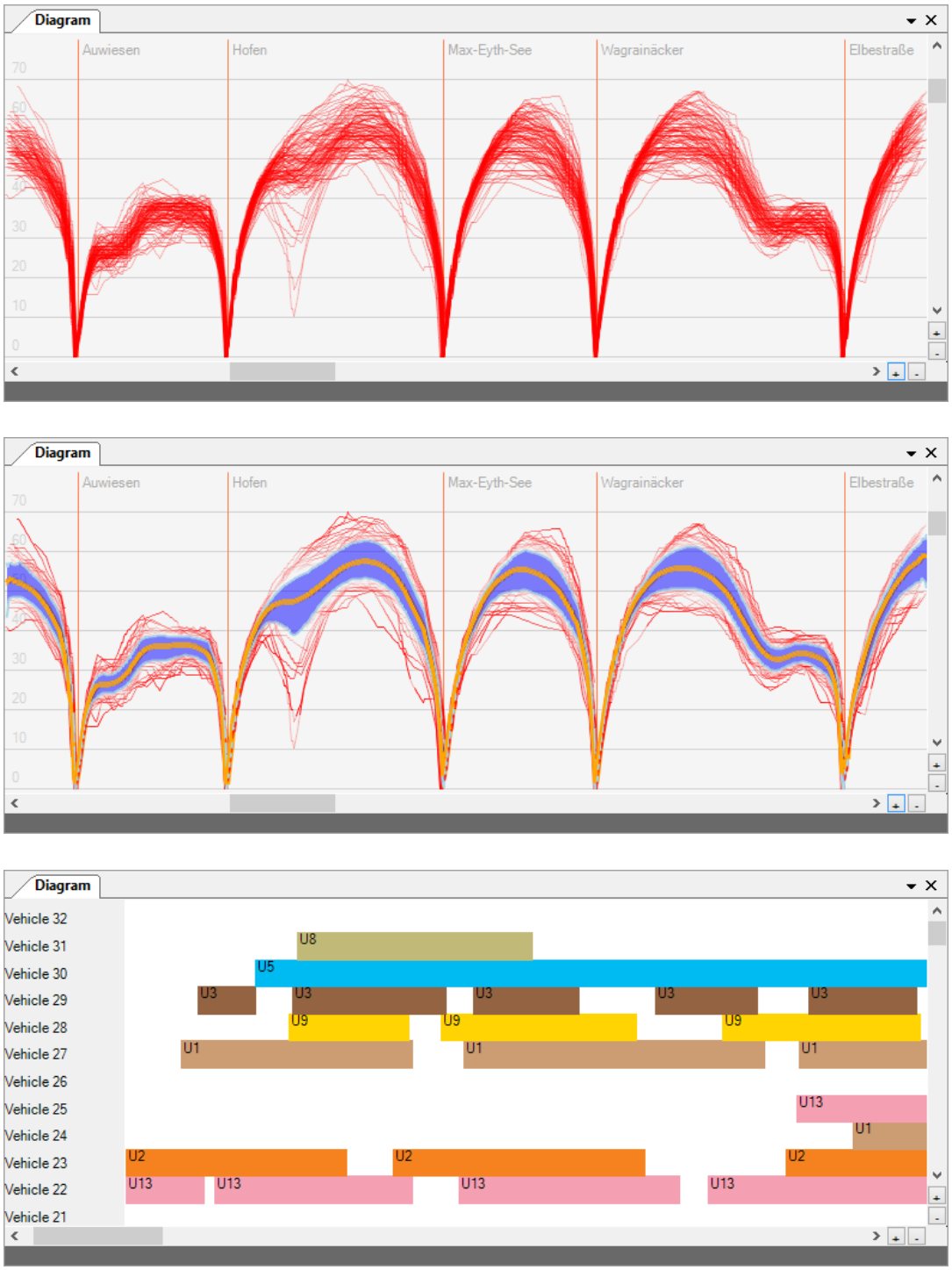


Figure 5.11: The diagram view. Top: Vehicle velocities only. Centre: Vehicle velocities, average velocity and standard deviation. Bottom: Stacked bars showing the assignment of vehicles to routes (time range).

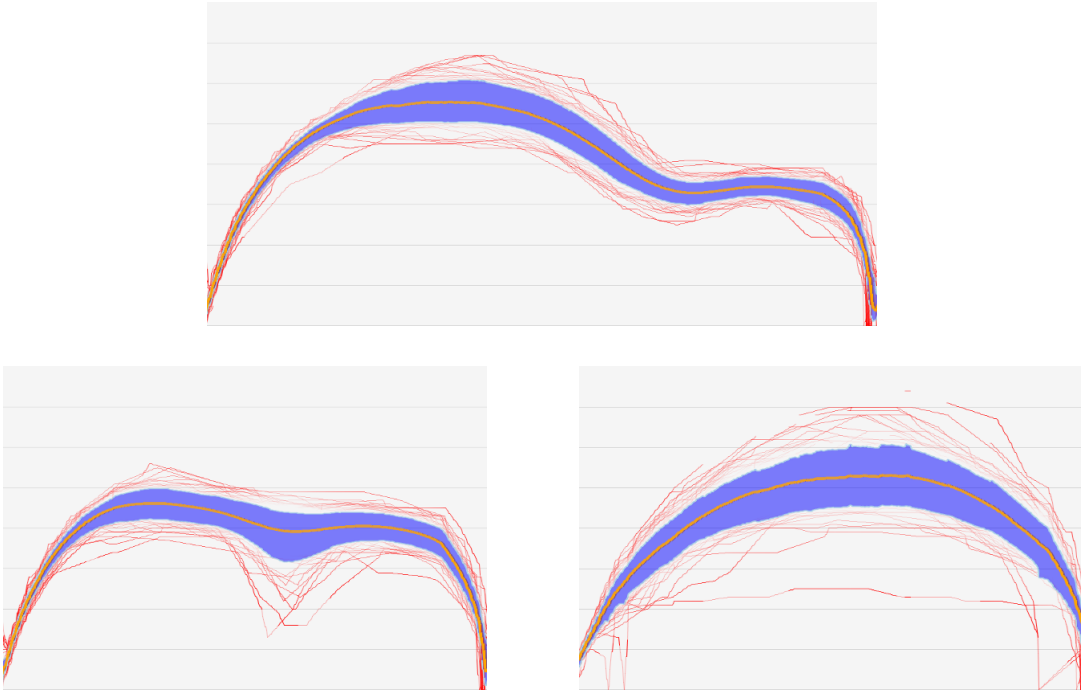


Figure 5.12: Top: An average speed (orange) characteristic for all vehicles. Bottom left: A noticeably increased standard deviation (blue) at a point where some vehicles decelerate and some do not. Bottom right: A single outlier of a vehicle going much slower than all other vehicles.

singular graphs well outside the standard deviation range (Figure 5.12 bottom right).

5.6.3 Derived data sequences

Filtering nodes can preprocess and transform the data items before they reach a consumer node. This can be used to calculate derived values that are not present in the original data. For example, the discrete differentiation of the speed sequence yields the acceleration of the vehicle. This may be significant for assessing passenger comfort. The second derivative of the speed is the jerk, that is, the rate of change of the acceleration. Sudden changes in acceleration reduce passenger comfort, so an analyst may be interested in whether there are certain locations within the transport network that are prone to causing jerks.

Jerks are to be expected when arriving at or departing from a station. Displaying the average jerk along a route can reveal locations where jerks occur outside a station. Figure 5.13 shows one such situation. The map view places these readings in a geographical context. Checking the location

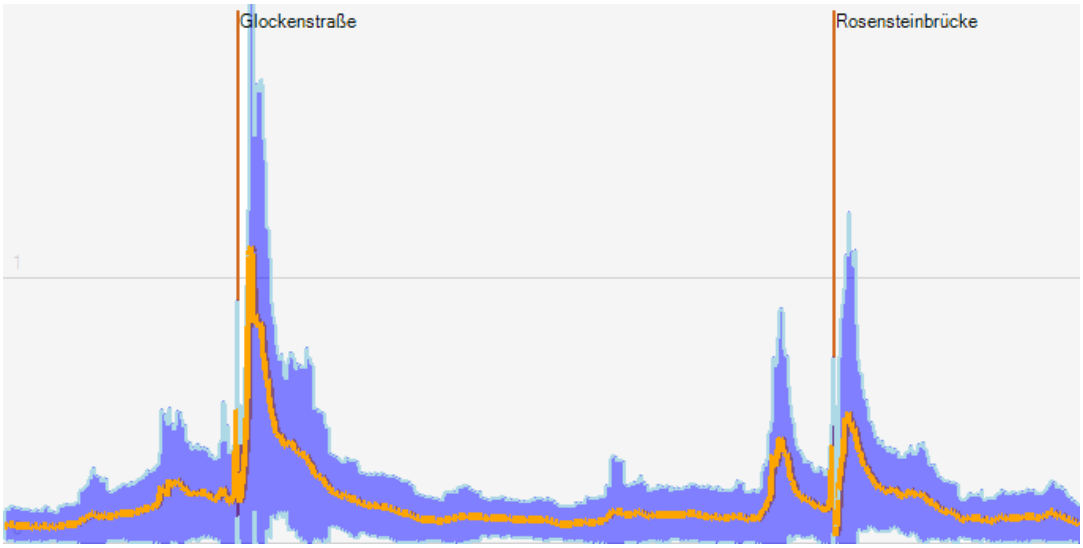


Figure 5.13: A display of the average jerk (orange curve) and its standard deviation (blue) helps in discovering locations where vehicles experience a distinct average jerk while not near a station (Wörner and Ertl, 2012). ©2012 The Author(s) Computer Graphics Forum ©2012 The Eurographics Association and Blackwell Publishing Ltd.

on a city map shows that there is both a street and a pedestrian crossing just before the station, so these jerks may be caused by conflicts with other traffic.

5.6.4 Expert feedback

An early version of the framework was presented to the domain experts at Stuttgarter Straßenbahnen AG. They were generally very interested and soon started to discuss assumptions on operational causes for various observations in the visualizations. When asked about the usefulness of this kind of visual analysis for their domain, they concluded that they were already able to do all this but ‘not with as little effort, not in a systematic way, and not in an automated way’. Instead, to obtain a similar analysis, they would usually turn to an external contractor and have the analysis produced specifically. This might be an indication that public transportation is an application domain that can benefit from visual analytics in that it can simplify the process of analysing network data and make on-site, ad hoc analyses feasible.

6

Linear Data Navigation

Chapters 3 and 5 presented systems for the exploration and analysis of large implicit or explicit data spaces. One way to make these large spaces accessible and an approach followed by both systems is filtering: Examining only a small, relevant subset of the data. If the user can give a formal definition of a relevancy criterion, automatic filtering can provide valuable support by generating the subset in question and presenting it to the analyst.

If users can precisely name what they are looking for—by giving a record id or a customer number, for example—the filtered subset will usually contain only the data item in question. Similarly, when a precise description of an item in question can be given—by providing various restrictions on its attributes—automatic filtering will likely reduce the data set to those few items that match the formal description. Visual analytics applications, however, are usually intended for those cases when a simple data query is not enough to answer an analysis question. Queries are of limited use when analysts search for something that is 'unusual' or

The approach discussed in this chapter has previously been presented in:

Wörner, M. and Ertl, T. (2011a), 'Multi-Dimensional Distorted 1D Navigation', in *Proceedings of the 2011 International Conference on Information Visualization Theory and Applications – IVAPP 2011* (SciTePress), 198–203;

Wörner, M. and Ertl, T. (2013b), 'SmoothScroll: A Multi-scale, Multi-layer Slider', in Csurka, G., Kraus, M., Mestetskiy, L., Richard, P., and Braz, J., eds., *Computer Vision, Imaging and Computer Graphics. Theory and Applications*, Communications in Computer and Information Science, 274 (Heidelberg: Springer), 142–154.

‘interesting’ and are interested in the context of a data item as much as in the item itself. Here, a visualization of the filtered data enables users to explore the data and possibly change or refine filter conditions or other automatic analysis settings. In these cases, a static visualization will generally not be sufficient and systems will instead present interactive visualizations that enable users to navigate through the data set. Typically, these visualizations will have to answer three questions:

1. ‘What am I looking at?’ The visualization needs to provide detailed information on the data item currently examined.

2. ‘Where in the data am I?’ The visualization needs to put the current data item in relation to the entire set.

3. ‘What else is there?’ The visualization needs to provide (possibly highly aggregated) information on other regions in the data set to facilitate a user’s decision on which regions need to be examined in more detail.

As stated in Section 2.3, an interactive visualization can answer these questions by following the overview + detail paradigm and combining an overview display, which shows a coarse, less detailed view of a large section of the data, and a detail display, which shows the current focus point in full detail but spans only a few data items.

This chapter focuses on data sets that are ordered along one dimension: They have a clear notion of ‘forward’ and ‘backward’. This includes the vehicle data set explored using the *Frida* framework in Chapter 5 (all data items included a timestamp) and the machine data set from Chapter 4 (which handled recordings of time-varying sensor values). In Chapter 3, *Sam* supported the optimization of process layouts by calculating simulation runs. These runs, too, represent layout state changes over time. Time may be the most common one-dimensional ordering criterion (and has extensively been studied, see Aigner et al. (2011) for a comprehensive overview) but it is not the only one. In the *Frida* framework, for example, vehicle attributes can be mapped to the distance along a route. Similarly, text can be browsed forward or backward, as can any sorted list.

A traditional way of navigating through data ordered along one dimension is using the arrow keys on a computer keyboard. On modern systems, these typically allow for two different speeds: navigating by line (using the ‘cursor up’ and ‘cursor down’ keys) or by page (using the ‘page up’ and ‘page down’ keys). The actual definition of a line or page may depend on the application. Additionally, the ‘home’ key moves to the first data item while the ‘end’ key moves to the last. The two sets of keys allow for both precise

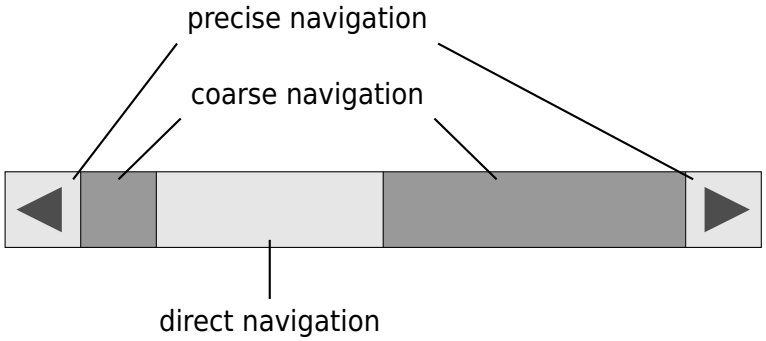


Figure 6.1: A standard scrollbar.

and coarse navigation through the data. With the exception of the ‘home’ and ‘end’ keys, this navigation is always relative. A user wanting to reach a specific point in the data, such as a certain chapter in a book, would have to repeatedly use the coarse navigation keys to get near the data item in question, then the precise navigation keys to focus it. Arrow keys provide no information on the context of the currently visible information or the current position in the data.

In graphical user interfaces, the typical control for navigating through one-dimensional data is the scrollbar (Figure 6.1). Like the arrow keys, scrollbars allow for precise navigation (clicking one of the arrows moves the focus point by a line) and coarse navigation (clicking between the arrow buttons and the scrollbar ‘thumb’ moves the focus point by a page). Unlike arrow keys, however, a scrollbar illustrates the relation of the currently visible portion of the data and the whole data set by varying the size of its central element, the so-called ‘thumb’. Also, a user can navigate quickly to any position in the data set by dragging the thumb to another position. However, the scrollbar visualizes only the position of the current view in the data set with no indication of what kind of data may be found at other positions. If the scrollbar represents a large data set, one pixel of thumb movement may correspond to a considerable jump in the data set, making precise direct navigation difficult.

Ideally, a navigation control should enable users to reach distant positions of the data set quickly while also allowing precise movement near the current position. To improve upon the scrollbar concept, Ahlberg and Shneiderman (1994) presented the Alphaslider (Figure 6.2). It is similar to a scrollbar in appearance and function but divides the thumb into three sections. Depending on which section is clicked during a drag operation, the thumb moves through the data at fine, medium, or coarse granularity. In two alternative operating modes, the scrolling speed is determined

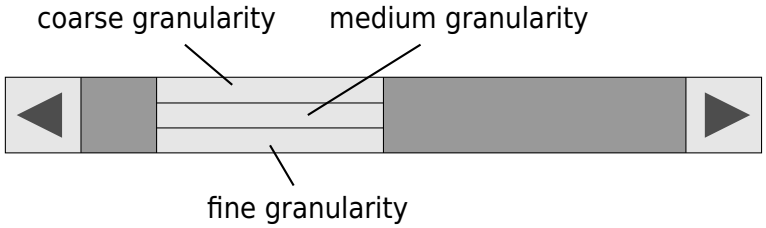


Figure 6.2: The Alphaslider as described by Ahlberg and Shneiderman (1994).

either by the speed of the mouse movement or by the orthogonal distance of the mouse cursor from the slider. The idea of using orthogonal mouse movement to control the granularity of the scrolling was explored further in the OrthoZoom Scroller by Appert and Fekete (2006). It, too, uses the orthogonal mouse cursor position to determine the scrolling speed but also integrates a representation of the data set into the control itself. The scale of this data display changes with the scrolling speed so the user sees a coarse data display while using a coarse scrolling granularity and a detailed display while using a fine scrolling granularity (Figure 6.3). Instead of displaying the data at a single, adaptive scale, it is also possible to show multiple data displays at multiple scales to provide both overview and detail information at the same time. The SIMILE project, for example, includes a timeline widget, which can show the timeline at multiple scales simultaneously (Huynh, 2006). Most examples use two scales (overview and detail). Another way to better utilize the two-dimensional screen when displaying one-dimensional data is to use a matrix display as demonstrated by Hao et al. (2007). A degree-of-interest function can be added to use different matrix resolutions for different parts of the data set.

6.1 The SmoothScroll control

This chapter builds upon these ideas and introduces the *SmoothScroll* control (Figure 6.4) for the efficient navigation through ordered data sets. As with the OrthoZoom Scroller, a *SmoothScroll* user can influence the scrolling speed through orthogonal mouse movements. *SmoothScroll*, however, does not vary the scale factor of a single data display but displays multiple data layers with fixed scale factors. When scrolling, the scrolling speed is determined by the scale factor of the layer beneath the mouse cursor and can be changed by moving the cursor from one layer to another while scrolling. In a horizontal alignment as in Figure 6.4, the scale factor increases from top to bottom: The top layer shows the entire data set

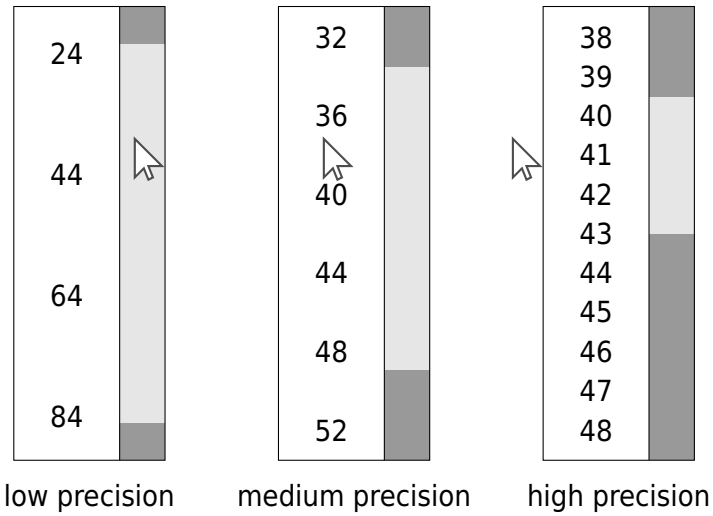


Figure 6.3: The OrthoZoom Scroller as described by Appert and Fekete (2006). Moving the mouse cursor orthogonally to the scrolling direction changes both the scrolling speed and the scale of the data display.

while the bottom layer shows single data items at a fixed item size. The intermediate layers interpolate between these two scale factors. Each layer highlights the data section visible on the detail layer to facilitate tracing the current position across the layers. *SmoothScroll* layers can simply repeat the information on the detail layer at a smaller scale or they can provide an aggregated or abstracted view of the visible data region and use colour or other visual cues to relate corresponding data ranges between the different layers. The scale interpolation across the layers creates a gradual distortion not unlike a fisheye view implementing the focus + context technique as described in Section 2.3. In this case, however, the transition is gradual but discrete: Data items are only distorted between layers, not within a layer, making comparisons of sizes and distances much more feasible. Since each layer has a certain thickness, items are represented as rectangular shapes. This two-dimensional area can be used to present various visual information on the items. This could be simply a label but might also include glyphs, function graphs, diagrams, or any other kind of visualization.

In Figure 6.4, the *SmoothScroll* control is used to browse a list of about 4,000 first names. The detail layer at the bottom displays one data item per name with the name shown as a label. The layers above it aggregate all names with the same first letter into a single data item displaying only this letter. Items are coloured in alternating colours according to the letter they represent. The two topmost layers are too thin to display text and use only colour. The colour coding creates a visible progression between

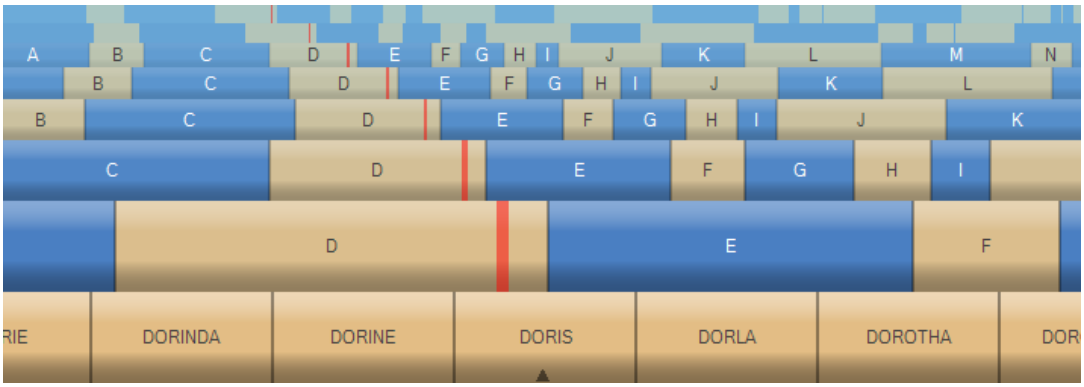


Figure 6.4: The *SmoothScroll* control displaying a list of about 4,000 names.

the layers and enable users to trace positions across layers, scales, and aggregation levels. In Figure 6.4, the display is currently centred on the name ‘Doris’. The names directly preceding or succeeding it in the list are also visible. The red highlight marks the section visible on the detail layer and indicates that ‘Doris’ is a name near the end of all names starting with ‘D’. These names are followed by about the same number of names starting with ‘E’ and a considerably smaller section of names starting with ‘F’. The top layers reveal that ‘D’ is a sequence of average length. There are longer sequences, such as the ‘C’ sequence immediately preceding it, and there are shorter sequences, especially near the end of the data set. To closer inspect one of these short sequences, a user can click on or near it on the top layer. This performs a long-distance, coarse navigation and will change the view to one similar to Figure 6.5. In this example, the clicked position corresponded to the name ‘Xuan’, the last of the names starting with ‘X’. The user can now examine the data items in the vicinity by clicking on or scrolling through the lower layers, performing short-distance, precise navigation.

The curvature seen when tracing data positions across the layers reveals that the scale interpolation is non-linear. A linear interpolation would form straight lines between corresponding data positions on different layers but would be impractical even for the relatively small name data set: If there are, as in this example, 8 layers with 4,000 items displayed on the top layer and 6 items displayed on the bottom layer and one were to interpolate the item size linearly across layers, the second layer from the top would show about 42 items, which is approximately 1 % of the total data. As a consequence, only the top layer would provide context information for more than the immediate vicinity of the focus position, as it would be the only layer to display a significant part of the data set. A more promising approach, therefore, is to interpolate not the size of the items but the

Figure 6.5: The SmoothScroll control after changing the focus point

number of visible items, which is proportional to the reciprocal of the item size. If one considers less detailed layers to be more 'distant', then this is similar to calculating a perspective projection of a 3D space.

Side note: Layer math

Perspective projection is a method of projecting a three-dimensional space onto a (screen) plane in a way that reproduces the familiar optical effect of distant objects appearing smaller. *SmoothScroll* interpolates layer sizes and positions in a similar way by associating a virtual distance $z(\lambda)$ with each normalized layer position $\lambda = l/l_{max}$, where l is the layer index between 0 (least detailed layer) and l_{max} (most detailed layer). The layer distance is converted into an item scale factor $s(\lambda) = e/(z(\lambda)+e)$, where e is an arbitrary 'eye distance' between the virtual eye and the screen plane. The topmost layer displays all items in the data set, so if S is the display size and n is the number of items in the data set, then the item scale factor for this layer is simply $s(0) = S/n$. On the other end of the control, $s(1)$, the item scale factor on the most detailed layer, is a user-defined parameter. Interpolating the distance values between $z(0) = e \cdot n/S - e$ and $z(1) = e/s(1) - e$ gives $z(\lambda) = z(0) \cdot (1-\lambda) + z(1) \cdot \lambda$ and thus $s(\lambda) = S \cdot s(1)/(\lambda S + (1-\lambda) n \cdot s(1))$.

The layers are aligned relative to the display position of the currently focused element. If f is the data index of the focus element ($0 \leq f < n$) and its position on layer λ is $x(\lambda)$, then $x(0) = S \cdot f/n$, that is, the topmost layer does not move and always displays the entire data set. On the most detailed layer, the focus point remains at the centre of the control: $x(1) = S/2$. The placement of the intermediate layers is determined by interpolating linearly between $x(0)$ and $x(1)$.

Clicking the control moves the focus position to the corresponding data item. This allows for coarse long-range navigation on the top layers and precise short-range navigation on the bottom layers. Additionally, holding the right mouse button scrolls the focus position towards the mouse cursor position. The scrolling speed is determined by the scale of the layer beneath the mouse cursor and can thus be changed by moving the cursor vertically across layers while scrolling. This is similar to the interaction principle of the OrthoZoom Scroller, which Appert and Fekete (2006) have evaluated to be very effective. With *SmoothScroll*, however, the scale transitions are discrete and information is displayed at multiple scales simultaneously, whereas the OrthoZoom Scroller uses a single data display with continuous scale changes.

As a consequence of the layer interpolation being based on a perspective projection, scrolling through the data set produces a parallax effect similar to panning along a three-dimensional scene. Indeed, when looking into the distance, one generally sees much of the scenery but at very little detail. This may be a suitable visual metaphor, so to emphasize this impression, *SmoothScroll* can optionally add a depth fog effect to the more ‘distant’ layers.

6.2 Glyphs and highlights

SmoothScroll items can be used to display a label or other information on the data item they represent. Glyphs, for example, can identify the state of an item in a very compact form. In Figure 6.6, the control displays a data set from the 2010 IEEE VAST challenge. This data set is from a text analysis task and contains natural language documents along with metadata such as timestamps. In the figure, each item represents a day in the data set. Items alternate between a blue and a grey background colour to indicate months. Items that actually contain documents are painted in a brighter shade. On the more detailed layers, document glyphs visualize the number of documents and the most detailed layer even shows the first letters of the document subjects.

For this example, a full-text search for two search terms has been performed and days with documents that contain one of the terms are highlighted in green or red (depending on which term was found). The view is centred on a case of a document containing the second term appearing just one day after a document containing the first term. The other *SmoothScroll* layers show that there are a few more occurrences of the first term before and after this incident, but there is no other occurrence of the second term.

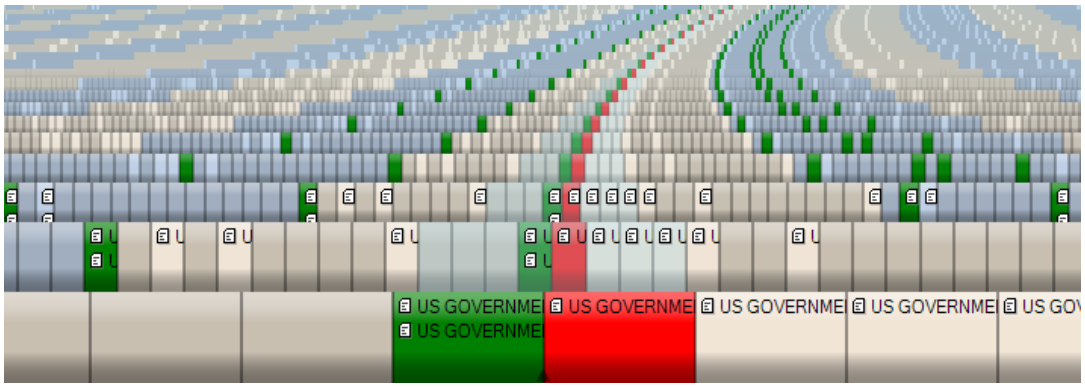


Figure 6.6: The SmoothScroll control displaying documents along a timeline. Documents matching search terms are highlighted.

6.3 Multiple aggregation levels

While time is a linear phenomenon that simply progresses from one moment to the next, it is natural to define common aggregations such as minutes, hours, or days. In the Gregorian calendar, these aggregations are strictly hierarchical: Years can be divided into months, which can be divided into days, which can be divided into hours. In Figure 6.7, the *SmoothScroll* control has been configured to display a set of pictures taken with a digital camera. On the top layer, different shades of grey set the months apart. The third layer also displays the abbreviated name of the month. The fourth layer begins to subdivide the months into days, the sixth layer adds labels showing the number of the day, and on the bottom layer, each item represents one hour, which is identified in the item's label.

The item label on the bottom layer includes the number of pictures taken during that hour in parenthesis. This information is also reflected in the colour of the item: If any pictures were taken during the corresponding hour, the item turns blue, with brighter shades of blue indicating more pictures. The second layer from the bottom uses different aggregation levels for its background colour and the picture count. On this layer, the item background colour alternates per day. On top of that, coloured stripes display picture counts per hour. These stripes span only half the item vertically, so the background colour will always be visible in the top half. This provides a quick overview of the distribution of pictures over the course of a day. In Figure 6.7, pictures were taken from morning to evening on 22 and 23 January, whereas on the 20th and 24th, pictures were taken only around midday. The remaining day layers aggregate pictures by day (displaying only one shade of blue per item) while the month layers again use the same scheme as the first day layer: There is only one item and one

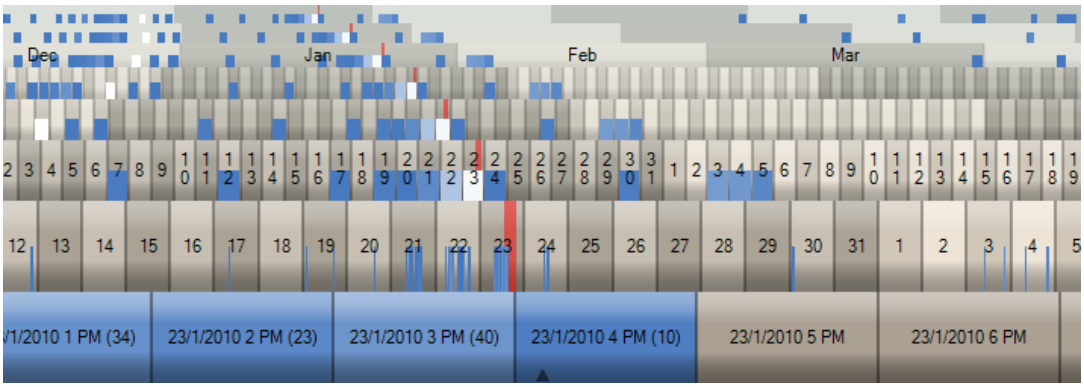


Figure 6.7: A timeline of pictures.

background colour per month, but blue overlays show picture counts per day.

In a real application, a user would probably be interested in more than just the picture count, so it might be desirable to incorporate single pictures into the hierarchy in order to display metadata such as the file names or file sizes per picture. However, while each day can be divided into 24 hours, dividing an hour into a number of pictures is not meaningful, as the number of pictures per hour is a quantity measured over time, not a hierarchical subdivision. If a layer listing pictures were added beneath the hour layer and the items on this layer were, as before, of fixed size, this layer would have to move at varying speeds to stay in sync with an hour layer moving at constant speed. This would break the strict hierarchical relationship between layers and the visual consistency of the scrolling.

If one were to abandon the requirement of a fixed size for detail items, one could adjust the item size so that the items taken during one hour always fill the screen space assigned to that hour, restoring a consistent relation between hours and pictures in terms of screen space. The obvious downside of this approach is that for hours with many pictures, these detail items might become very small, preventing the control from displaying any useful information on them even on the most detailed layer. Another approach is to keep the fixed detail item size, but reverse the mapping logic and define the picture to be the elementary element of the data sequence. Pictures can be ordered by the time they were taken and the resulting sequence of pictures is an ordered data set, which can be displayed by the *SmoothScroll* control. In this configuration, the layers above the picture layer could again aggregate pictures into groups of pictures taken in the same hour, on the same day, or in the same month. The size of these items would then depend on the number of pictures taken in the corresponding period of time, not on the length of the period. The result would not be a

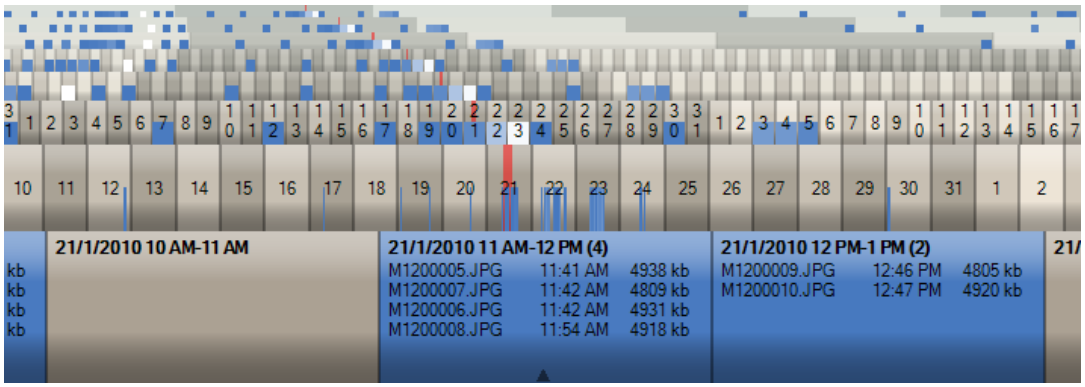


Figure 6.8: The pictures timeline with an integrated list of pictures on the detail layer.

continuous timeline, but—depending on the user's intent—may well be a useful visualization for exploring the data set.

Finally, one can keep both the hour as elemental quantity and the fixed item size and instead enlarge the items so that information on the individual pictures can be integrated into the item representing an hour. The width of a detail item is a user-defined parameter of the layer interpolation scheme. To increase the item height without increasing the total size of the control, one can extend the interpolation method to incorporate fixed heights for certain layers, determining the height of the other layers based on the interpolation scheme and the remaining space. The increased size of the detail items can then be used to integrate lists of file names, creation times, or file sizes, as in Figure 6.8. The number of pictures that fit into an item is certainly limited, but the extended item size would also allow for the inclusion of GUI elements such as scrollbars to access more than the immediately visible pictures. In practice, however, it would be advisable to discuss whether in such cases, integrating all this information into the control is in fact the best approach. It might be more efficient to use the control only for navigating in time and display the actual pictures in a separate view, possibly linked to the *SmoothScroll* control in a brushing + linking fashion.

6.4 Hierarchies and vertical display

A calendar-like time hierarchy is very regular. Month lengths vary between 28 and 31 days, but all days have 24 hours and all years have 12 months. There are other data sets that can be ordered in a hierarchical fashion but for which this hierarchy is much less regular. Text, for example, is essentially an ordered sequence of letters, words, or sentences but also

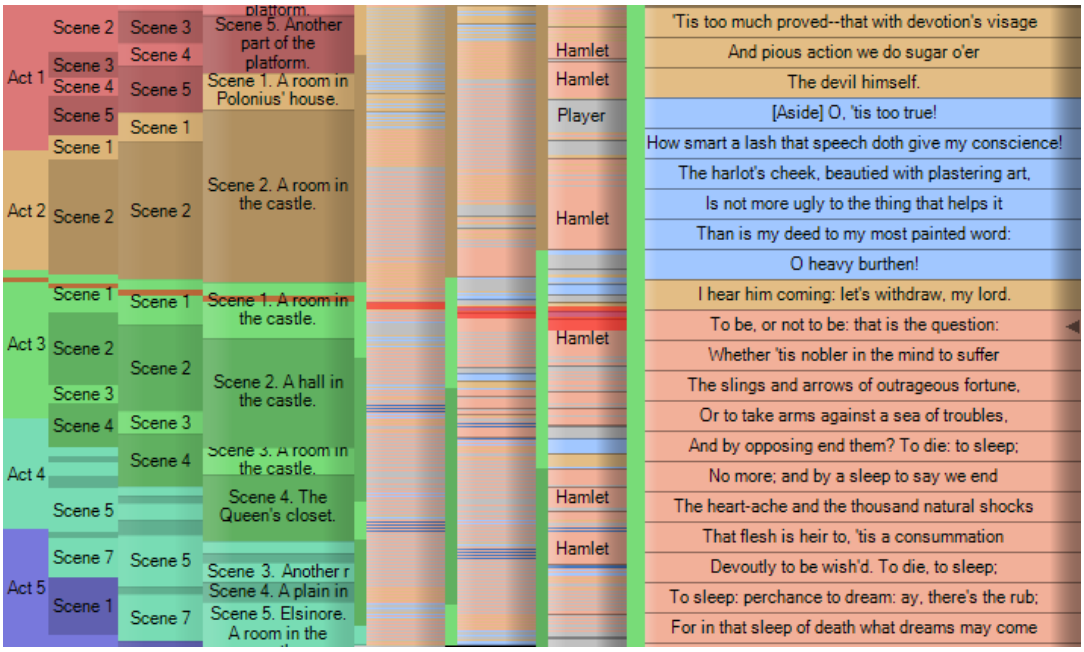


Figure 6.9: Shakespeare's *Hamlet* in acts, scenes, utterances, and lines of text. 'To be, or not not be' is said by Hamlet in the first half of the first scene of the third act and begins a long monologue.

often contains a hierarchical segmentation. Books, for example, may be divided into parts, chapters, and sections, whereas a classical play is presented in acts and scenes. In Figure 6.9, the *SmoothScroll* control is used to browse through Shakespeare's *Tragedy of Hamlet, Prince of Denmark*. In this configuration, the detail items show single lines of text. Since these are generally wider than they are tall, the layers have been arranged vertically, with the most detailed layer displayed on the right and the least detailed on the left.

The first layer displays the five acts of the play. The next three layers divide these into scenes. Scenes are then broken down into utterances—sections of text spoken by the same character—and finally individual lines of text. In the left half, colours represent acts (with scenes alternating between two shades of the colour of their act). In the right half, individual colours have been assigned to important characters in the play. The distribution of colours on the utterance layers gives an impression of which characters play an active role in the different parts of the play and whether a section contains intensive dialogue or long monologues. If a long monologue creates an utterance item large enough, this item is labelled with the character's name. To retain some visual continuity between those layers coloured by act and scene and those coloured by character, the more de-

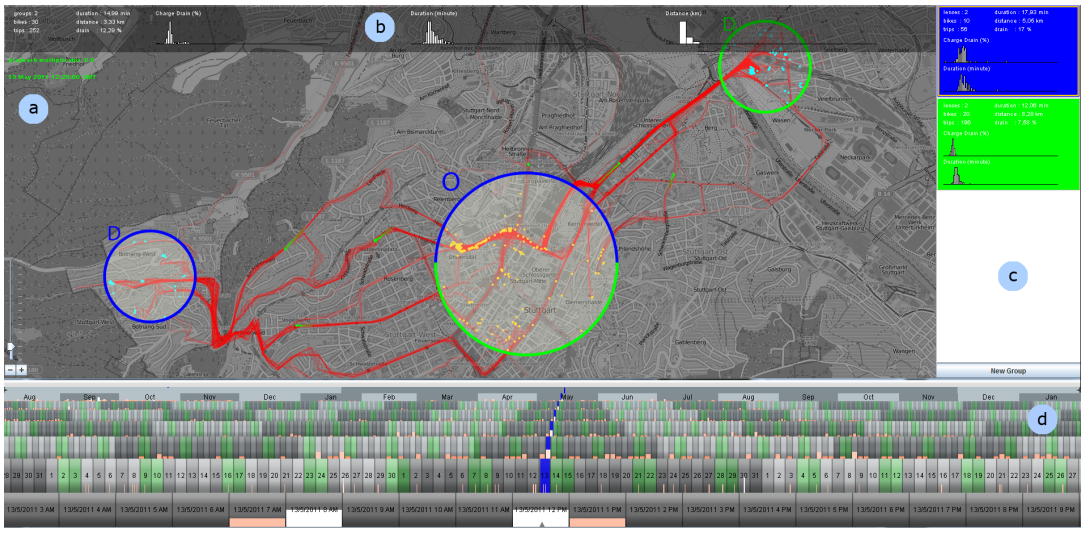


Figure 6.10: The *TrajectoryLenses* system using a *SmoothScroll* control to provide navigation through time (Krüger et al., 2013). ©2013 The Author(s) Computer Graphics Forum ©2013 The Eurographics Association and Blackwell Publishing Ltd.

tailed layers also display a stripe in the colour of the corresponding scene along the left edge of their items.

6.5 Application and discussion

The *SmoothScroll* control has been used in a scientific publication on a visual analytics system for the exploration of movement trajectories. *TrajectoryLenses* by Krüger et al. (2013) visualizes trajectories recorded by electric scooters over the course of about two years and provides various means to filter, explore, and analyse them, most prominently in the form of lenses a user can place on a map to define trip origin and end regions. *TrajectoryLenses* uses a *SmoothScroll* control to visualize the distribution of trips over time and as a means to set time spans for filtering. Because all layers can be used to define the time span, a user can first place a coarse approximation of the desired span on the less detailed layers and then refine it on the more detailed layers.

ScatterBlogs2 by Bosch et al. (2013) is is a visual analytics system for the real-time monitoring of microblog messages posted on Twitter or similar services. It supports the user-guided definition of filter criteria and displays matching messages in various views, one of which is a *SmoothScroll* timeline that visualizes the number of messages matching the filter criteria over time at different aggregation scales (Figure 6.11).

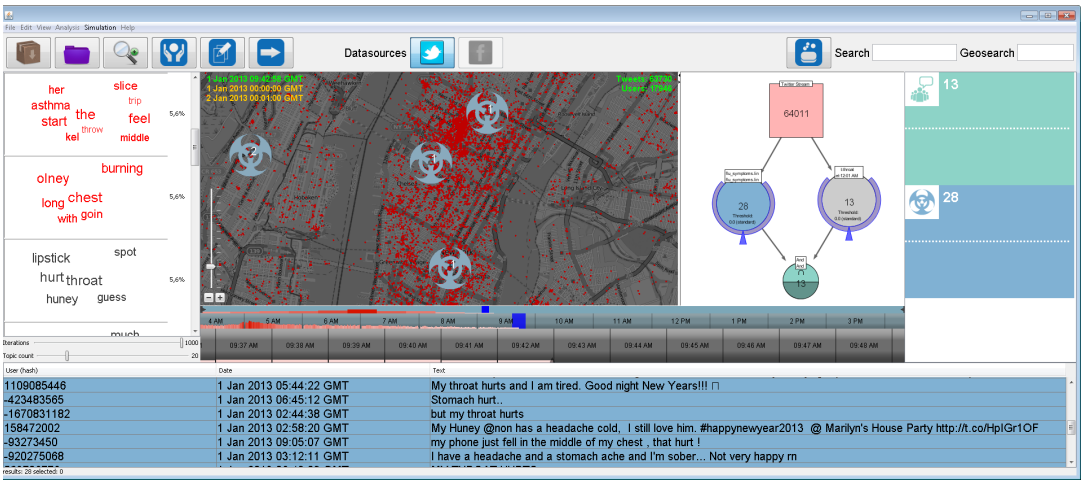


Figure 6.11: The *ScatterBlogs2* system using a *SmoothScroll* control (below the map view) to visualize the frequency of microblog messages matching complex filter criteria (Bosch et al., 2013). ©2013 IEEE

The most apparent disadvantage of the *SmoothScroll* control is that it requires considerably more screen space than a standard scroll bar if it is to display a reasonable number of layers and at least some on-layer information. Where screen space is a very limited resource, this space requirement may outweigh the benefit of intuitive multi-scale navigation and data overview. It may then be more efficient to replace *SmoothScroll* with a standard scrollbar to free up space for other information the application needs to display. *SmoothScroll* shows several layers at different scales and scrolling speeds, but in a basic configuration, these all display the same data set. This causes a certain visual redundancy, which, to some extent, is essential for the visual continuity between layers and thus the intuitive navigation through the data. Nevertheless, the control's screen space efficiency can be greatly improved by introducing aggregation and semantic abstraction on the less detailed layers. While still displaying the same data, these layers then present the data in a different way than the detail layer, which may provide additional insight. Also, since *SmoothScroll* assigns a certain area of screen space to each (possibly aggregated) item, this space can be used to display information such as labels, glyphs or even line graphs or other visualizations at different abstraction levels while always providing visual cues on how these different levels of detail relate to each other. This possibility is a significant advantage over a distorted display without discrete layers (which can be approximated using *SmoothScroll* by increasing the layer count until each layer is only one pixel thick). It should be noted, of course, that there is a limit to how much information can reasonably be included into the display before producing significant visual clutter and

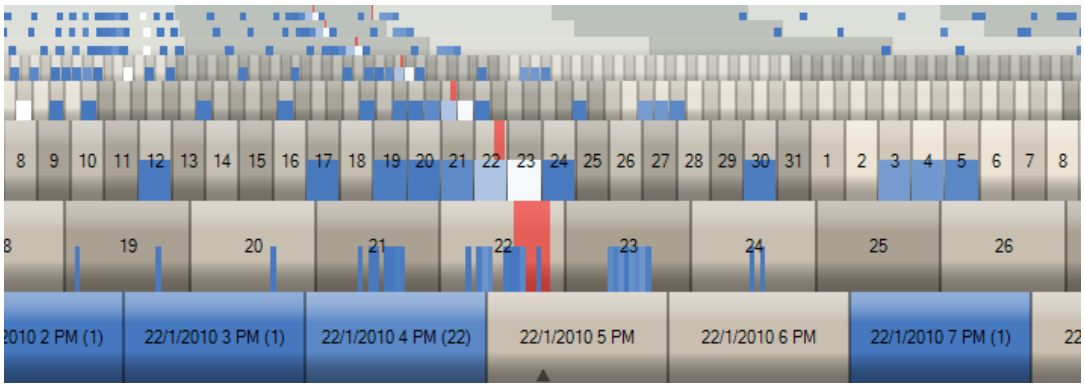


Figure 6.12: An alternative interpolation function, which causes smaller scale differences among the detailed (bottom) layers (as compared to the default interpolation in Figure 6.7).

overloading the user. It is important to always retain some sort of visual consistency between different layers (such as by choosing a suitable colour scheme) or the layers would appear as unrelated and independent displays that could easily be replaced by separate views. Aggregation levels should be chosen so that they result in reasonably-sized display items and do not, for example, attempt to plot events at hourly resolution on a coarse layer that spans several months.

Section 6.1 described the interpolation scheme used to determine the layer scales and why it is superior to a linear scale interpolation. It is, however, not the only sensible option. For example, one might take issue with the fact that the current scheme causes considerable scale differences between the two most detailed layers, which may reduce the perceived degree of consistency between these layers. This may be mitigated by using a non-linear interpolation for the λ values of the layers (which in turn determine their z values and thus their scale factors). Figure 6.12, for example, uses $\lambda = \frac{1}{2}(1 - cos(\pi \cdot l/l_{max}))$ (instead of simply $\lambda = l/l_{max}$), which reduces the z distances between the most detailed and between the least detailed layers in exchange for a steeper increase near the middle layers.

7 Conclusion

The previous chapters demonstrated the application of the visual analytics approach to different domains, most notably manufacturing and public transport. Chapters 2 to 5 each presented the design and implementation of a research prototype for a visual analytics system. Chapter 6 described a new kind of interactive control that can be used by a visual analytics system to support the user in navigating one-dimensional data sets. This chapter now summarizes and discusses the results of these previous chapters, then generalizes their analysis tasks to propose a classification of visual analytics systems.

7.1 Visual analytics systems

In the context of an introduction to visual analytics, Chapter 2 described a visual analytics system for solving the VAST contest 2009 mini-challenge 2—identifying a group of individuals in a social network based on a natural language description of the social structure of this group. The system provides automatic analysis support in the form of an automatic evaluation of formal rules that restrict the set of individuals that are candidates for the various roles in the group. Once these sets have been reduced sufficiently, a visual representation of the remaining individuals, their relations, and their potential roles communicate the situation to the user, who may discover conditions that further restrict the sets based on reasonings not covered by the automatic rules. The possibility to communicate these findings

back to the automatic algorithm completes the visual analytics cycle. The application of rules and the resulting effects on the candidate sets are visualized in a graph representation (contributed by Harald Bosch). This graph illustrates the individual reasoning steps that led to the current situation. The user can split the graph or go back to earlier steps. This also addresses the issue of analytic provenance, the possibility to reconstruct the reasoning that led to an analytic results and something often asked for in visual analytics systems.

The intention of the VAST contests is to promote and stimulate visual analytics research. By supplying synthetic data sets along with specific analysis questions, they provide welcome opportunities for developing new techniques and versatile tool sets. Using synthetic data sets has the advantage that a certain ground truth can be embedded into the data, so analysts can be certain that there is indeed something to be found and different approaches and solutions can be more easily compared. On the downside, synthetic data may not always faithfully reproduce all aspects of real-world data sets. With 6,000 members, the social network data set of the second VAST 2009 mini-challenge is certainly extensive enough to rule out all purely visual approaches for solving the problem. However, despite numerous uncertainties in the natural language descriptions of the social structures to be found—both in the actual number of contacts and in the validity of assumptions as a whole—the eventual solution matched the description to the letter. In fact, the solution can be found simply by formalizing the description (ignoring any uncertainty) and progressively eliminating accounts until only the solution remains. As a result and for this particular analysis, there is a clear need for automatic support, but there is no clear need for an interactive visual interface as a part of the analytic process. In Chapter 2, one important reasoning step—selecting the correct solution out of two remaining possible configurations—was left to the user. And indeed, the condition that social network users can only be part of a solution in a ‘handler’ role if they are not in contact with other handlers of the same solution may be easier to verify visually in a graph representation of a small number of possible solutions than to express in a formal constraint. Nevertheless, even this last reasoning step can be performed automatically without much effort.

Of course, this only became clear after the solution had been found, so the system was designed with the vague descriptions and a possible requirement for human reasoning in mind. As a result, it is possibly much more powerful than required for this particular task, which, of course, is absolutely in line with the intentions of the contest. It is unfortunate, however, that the contest did not challenge the system beyond what can comfortably be achieved with automatic processing alone, where, after all,

visual analytics sees its true purpose. With the confidence values describing the degree of fulfilment of a vague rule and the possibility to assign a confidence to a rule (and thus limit its influence on the computation), the presented system could have identified solutions that almost but not entirely match one of the natural language descriptions. With the user steering the process, additional, implicit restrictions might have led the human analyst to deem some of a number of possible solutions less likely. For example, if the handlers are the employee's immediate contacts, one might expect them to live in the same city even though this is not explicitly stated in the description. This kind of reasoning is very different from evaluating a clear-cut definition of acceptable solutions and much harder to do automatically. This might indeed have resulted in an analytic challenge in which neither a purely automatic nor a purely visual approach can produce the desired results and visual analytics excels.

Chapter 3 presented *Sam*, a visual analytics system that supports the planning of manufacturing layouts for a reconfigurable manufacturing system. Given a set of goals and constraints, it relies on a human expert to design candidate layouts but offers automatic support in the form of a detailed simulation of a layout's operation. This not only provides insight in the form of an animation of work piece movements throughout the process but also produces a number of key figures that can be used for a quick and simple if somewhat superficial comparison of layouts. Further automatic support is provided by the layout discovery feature, which continuously creates and evaluates layout variations in the background and reports back once an interesting configuration has been found. The discovery usually starts with a set of randomly generated layouts, but whenever the user creates and simulates a layout that appears noteworthy with regard to the current search priorities, this layout becomes part of the automatic reasoning and may be the basis for further variations, forming the feedback loop and completing the visual analytics cycle. In fact, *Sam*'s approach may break up this cycle to a certain extent, as both the human and the automatic analysis can run in parallel for a while, with neither influencing the other. When either side makes a significant discovery, this is communicated to the other and may influence the further progress of their analysis. Nevertheless, the human analyst is always in control. The search priorities by which the automatic discovery judges layouts are user-defined (and can be adjusted during the analysis process). Also, while all layouts designed by the user are automatically evaluated and considered by the automatic discovery, the results of the automatic analysis are merely proposed in an unobtrusive way and the user is free to ignore them.

This collaborative approach may be suitable for a generic class of optimization problems, where the analysis task is finding a good (not

necessarily the best) element in a large data space. If the nature of this data space and the desired properties of the solution imply that elements that are close in the data space are often of similar merit, the automatic generation and evaluation of possible solutions can provide valuable support to the human analyst. Again, this problem is not necessarily one that depends on human intervention. An automatic search algorithm—given the right target function and left alone for long enough—might eventually find any solution a human analyst would have devised.

Nevertheless, human reasoning can tremendously improve this process. *Sam*'s automatic discovery currently uses an evolutionary algorithm. These are notorious for converging towards genetic monocultures, circling around a local optimum with no chance of ever exploring other regions of the data space again. This can be improved by a careful tweaking of algorithm parameters or the regular introduction of new genetic material into the population. However, unless the new individuals happen to be 'better' than those in the current population, they will quickly be eliminated in favour of the known solutions. Layouts created by a human analyst, on the other hand, will usually not be random but follow some specific idea. They may not be completely fleshed out initially, but the probability of coming across a layout that can stand up to the established population and thus have a lasting effect on the automatic search is considerably higher. These two very different approaches, the algorithm with the ability for a trial-and-error scan through hundreds of layouts but no notion whatsoever of *why* a given layout may be better than another and the human analyst going at a much slower pace but following specific plans and ideas, complement each other well. Even if the layout ultimately chosen was constructed without adopting any ideas from the automatic discovery, the fact that the algorithm watching the process was unable to produce a superior variant can raise the user's confidence in that the chosen layout contains no obvious flaws or wasted optimization potential. Finally, human involvement may be required for reasons entirely unrelated to the effectiveness of the analysis. When decisions may have considerable negative effects if they turn out to be flawed, decision-makers may be unwilling to trust a purely automatic solution because they do not want to take responsibility without sufficient insight into the reasoning that led to a proposed course of action. A collaborative and interactive planning process may provide this insight and thus ensure the necessary accountability.

Sam operates on an implicit data space, the space of all potential process layouts. To find a suitable layout in this space, various elements need to be evaluated, either by a human expert or by some formal metric. *Sam*'s layout simulation supports both forms of evaluation. An animated visualization of the layout's operation aids the expert in assessing its

advantages and disadvantages. The automatic algorithm, on the other hand, optimizes a single value, the layout score. This score is a weighted average of various key figures, so the evaluation considers different aspects when judging a layout. Nevertheless, it is a significant simplification of the human evaluation process and must be expected to not cover all aspects of an analyst's reasoning. This may be by necessity when the human judgement is at least partly based on implicit knowledge, exceptional requirements, or personal preferences. As a result, the score function can only approximate a user's decision but never replace it.

The analysis task addressed by the system in Chapter 4 is a classification of data sequences into those indicating working and those indicating failure conditions. In this case, the system does not contain a pre-built automatic analysis component but provides a set of analytic tools, from which the user can construct a suitable classifier in the form of a data flow graph. This method is useful when it is not clear in advance exactly which form of automatic support is required. In Chapter 4, there was a data set of recorded machine diagnostics data but no clear and formal definition of how the state of a machine can be assessed based on this data. As a result, the system supports an exploratory approach, in which the analyst combines the provided tools until a promising classifier has been found. Here, the design of the automatic analysis is part of the data analysis process. Once a useful classifier has been found, it can be used (by expert and non-expert users) to classify data and, if the need arises, be adjusted as required (possibly only by expert users).

This system provides automatic support through its machine learning capabilities that can help to correctly configure the individual analysis nodes based on a set of pre-classified data sequences. The system can find parameters that result in a correct classification of these sequences and may thus also be suitable for correctly classifying other, as of yet unknown sequences. Also, the immediate evaluation of the data flow graph and the direct comparison of the automatic classification with the user-provided pre-classification gives immediate feedback on the correctness of the current graph. Once the classifier is set up, the system supports the classification of data sequences, not necessarily by replacing the human expert entirely but by dividing the data into clear cases of working or failure conditions and borderline cases, which warrant a detailed inspection by the human expert.

The exploration aspect is the main focus of the *Frida* framework presented in Chapter 5. It, too, supports the construction of a data flow graph and provides various processing nodes that can plot data on a map or draw diagrams. In this case, however, the public transport data set used is so extensive that it raises the question of how to best manage the inev-

itable delays in the data processing. *Frida* uses a streaming model that passes individual data items rather than entire data tables along the data flow graph. Four requirements are considered throughout the framework: Do not let time-consuming analysis tasks reduce the responsiveness of the user interface, where possible, make intermediate results available, provide a progress estimation, and do not let hardware resources limit the total volume of data that can be processed. In addition, *Frida* gives the user control over what parts of an analysis are to be evaluated, and it allows changes to the data flow graph while the analysis is in progress, updating data queries and rerunning parts of the evaluation as necessary. *Frida* focuses on data exploration rather than providing specialized support for a specific analysis question. While not general enough to address all potential analysis questions, it is not limited to the public transport domain. Its assumption that the data consists of entities recording events or regular measurements over time is true for the buses and trams but also, for example, for the machine tools in Chapter 4. In fact, it would certainly be possible to rework the machine data classification to build on *Frida* and thus improve its responsiveness for large data volumes. Other possible applications may include administrative and performance data recorded in computer networks and weather data from observation stations.

The *SmoothScroll* control presented in Chapter 6 is not a complete visual analytics system but a component that a system may use to provide a way of navigating through data ordered along one dimension. Navigating data is not a common analytical task per se but it is often an important auxiliary element of the reasoning process. Data sets can be explored by applying filter conditions and viewing only a limited selection of data items or by browsing the data set. In the latter case, *SmoothScroll* can improve the awareness for location and context and offers both coarse long-distance and fine close-distance navigation. It can also be used for visualization. With its multiple layers, it displays both detail and overview information as well as a selectable number of intermediate representations. The layers can not only be used for navigation but also visualize the data space at various levels of aggregation and abstraction, which is useful when there is some inherent structure or hierarchy in the data space. The most obvious application may be time-oriented data sets such as the ones used in Chapters 4 and 5 or those analysed by the *TrajectoryLenses* and *ScatterBlogs2* systems, which use the control for navigation and selection and for visualizing trip or term frequencies over time. Text documents are not time-oriented but certainly ordered along one-dimension and often hierarchically structured. As was shown for the *Hamlet* example, the *SmoothScroll* approach may prove useful for navigating hierarchical and possibly annotated text resources in the emerging field of digital humanities and elsewhere.

7.2 Generalization of results

This thesis presented a number of different visual analytics systems all tailored to a specific application domain and a specific analysis task. These tasks can be generalized to show the wide range of fundamental problem statements to which visual analytics can be applied. The VAST contest contribution (Chapter 2) solved an *identification problem*: Given some information on a group of data items, identify these items in a data set. *Sam* (Chapter 3) addressed an *optimization problem*: Find a good layout in the large space of possible layouts. Chapter 4 was concerned with a *classification problem*: Find a way to differentiate between 'good' and 'bad' data records. The *Frida* framework (Chapter 5) supports a user faced with an *exploration problem*: Find significant, unusual, or otherwise interesting facts in a very large data set. The *SmoothScroll* control (Chapter 6) is a component that addresses a *navigation problem*: Provide the user with a sense of location in a data set and the means to purposefully change this location.

The systems can also be characterized by the type of data they operate on. Section 2.5 introduced *Sam* as a system that operates on an *implicit* data space, one defined by a model of its elements. *Sam*'s data space is the space of all possible process layouts and this space is defined by a model that defines which layout elements exist and how they can be combined into a layout. This is in contrast to the public transport application example for the *Frida* framework, which explores an *explicit* data space, one that is defined by a list of its elements. As a consequence, *Frida* is mainly concerned with how to best extract the required performance figures from its extensive data set, whereas *Sam* has no way of looking up the performance of a layout and instead estimates it using a simulation. The machine data classification in Chapter 4 incorporates elements from both approaches: The first step of a classification process is designing the classifier, which in this system involves constructing a data flow graph from a number of predefined elements. The data space of possible graphs is just as implicit as *Sam*'s process layouts. The evaluation of the classifier, however, is done using a test set of machine data records with known classifications. This data is explicit, just like the vehicle data set.

Finally, the systems can be characterized by the size of the data volume they operate on. *Sam*'s implicit space of all possible process layouts is infinite (for example, one can always connect another linear transport element to an existing group of linear transport elements). *Frida*'s explicit vehicle data set is 'very large' (the exact definition of which is certainly debatable—for the purpose of this discussion: the data set is considerably larger than the main memory of a typical workstation machine at the time

	Data	Analysis task	Application
Chapter 2	Explicit, medium	Identification	Social network analysis
Chapter 3	Implicit, large	Optimization	Manufacturing layout planning
Chapter 4	Both, medium	Classification	Machine diagnosis
Chapter 5	Explicit, large	Exploration	Vehicle data exploration
Chapter 6	Explicit, medium	Navigation	Various

Table 7.1: An overview of the data space types, data volumes, and analysis tasks addressed by the systems presented in this thesis.

of writing). The diagnostic machine data set and the example data sets used for the *SmoothScroll* control are much smaller. The size of the data volume has some implications on the design of a visual analytics system: While the machine data classification system can cope without a sophisticated data handling scheme, retaining system interactivity during inevitable delays is one of *Frida*'s main concerns. *Sam*'s infinite and implicit data space can only be accessed by evaluating the corresponding model.

All systems visualize elements of their data space as required by their respective analysis task. The social network analysis and the *Frida* framework additionally visualize the analytic process itself by drawing a graph that clearly illustrates how the current analysis result was obtained.

Table 7.1 summarizes this generalized classification of visual analytics systems. It is not limited to the systems presented in this thesis. Other examples of recent visual analytics research can be found for each of these classes:

The *Deshredder* by Butler et al. (2012) supports the reconstruction of shredded documents with visual analytics (Figure 7.1). Reconstructing a document involves finding the original position of each shred in the document. This is an **identification task**: Of all the possible arrangements of shreds, find the original configuration to reconstruct the document. *Deshredder* provides automatic support in the form of an algorithm that examines shreds and determines likely combination based on shred edges and the fragments of print on them. The user can annotate shreds with their orientation (in cases where it can be inferred from visible letters on the shred), which is fed back into the algorithm in the form of constraints. The user can also confirm or reject proposed matches, further restricting the possible configurations. This process is repeated until all shreds have been matched to their neighbours and the document is reconstructed.

Sedlmair et al. (2012) presented *RelEx*, a system that supports experts

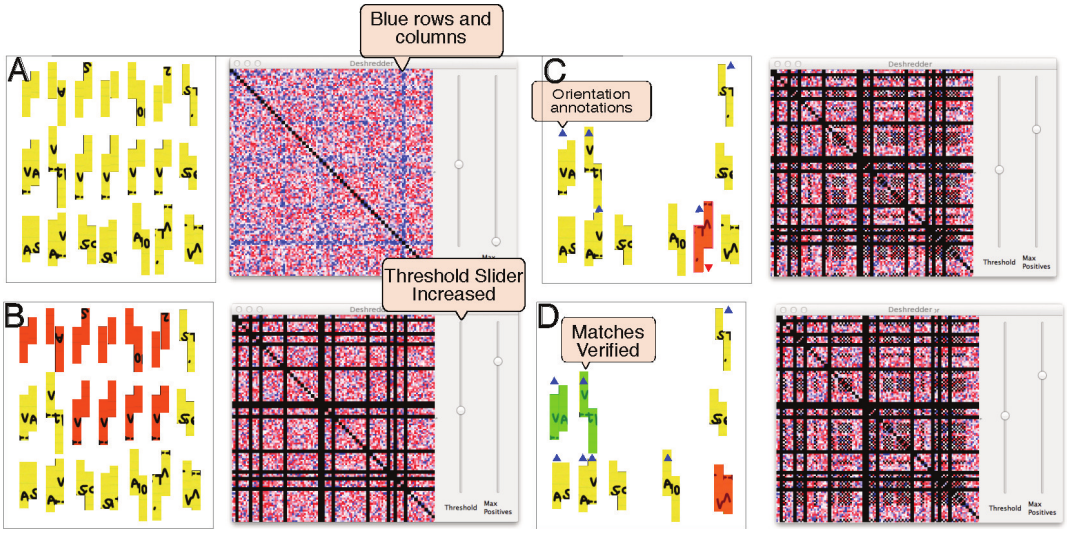


Figure 7.1: *Deshredder* by Butler et al. (2012) supports an identification task. ©2012 IEEE

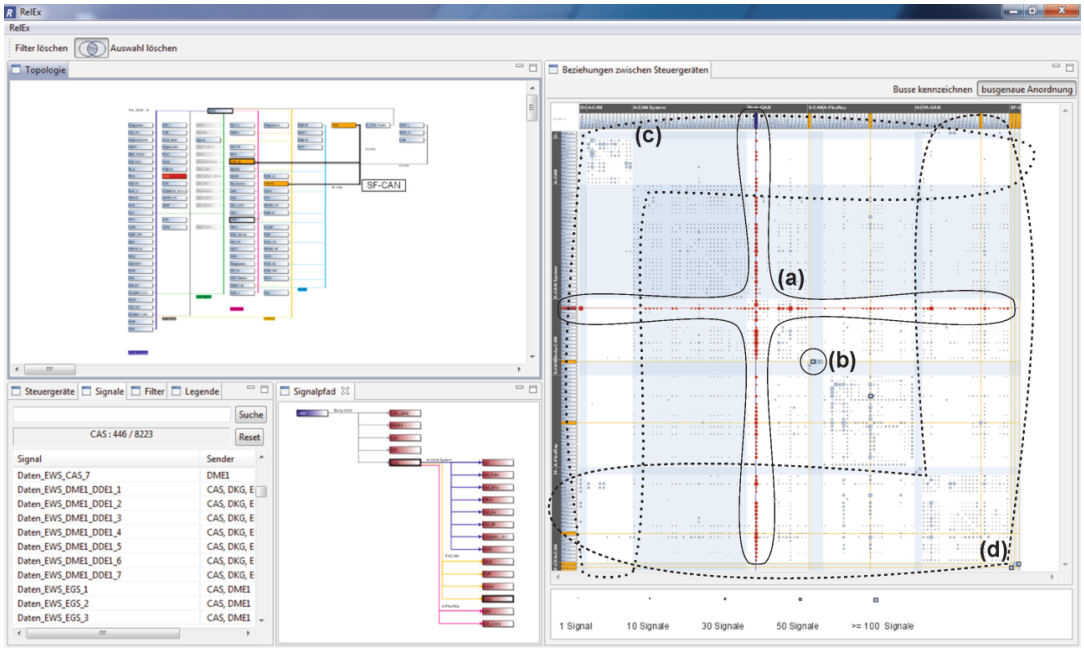


Figure 7.2: *RelEx* by Sedlmair et al. (2012) supports an optimization task. ©2012 IEEE

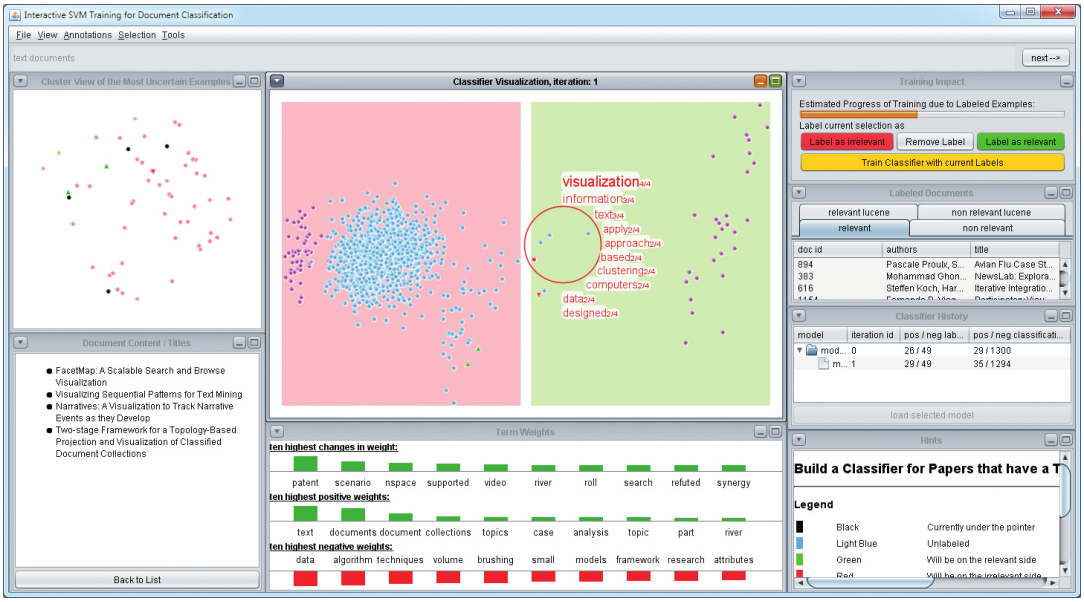


Figure 7.3: A visual analytics system by Heimerl et al. (2012) that supports a classification task. ©2012 IEEE

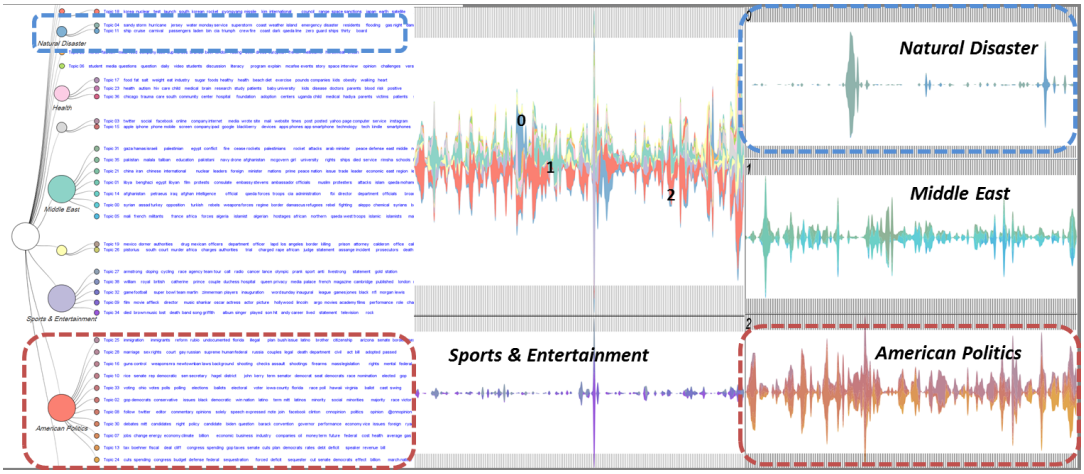


Figure 7.4: *HierarchicalTopics* by Dou et al. (2013) supports an exploration task. ©2013 IEEE

in designing and modifying in-car communication networks in a way that optimizes network traffic (Figure 7.2). This is an **optimization task**: Of all the possible ways to design the communication network, find one that results in a good distribution of traffic. Its approach is mainly visual. The optimization is carried out by the human expert and is supported by *RelEx* with four interactive and linked views that facilitate subtasks such as understanding the topology of the network and identifying hotspots.

Heimerl et al. (2012) described a system for interactively training a classifier on text corpora using an interactive visual interface. The user sees a spacial arrangement of text documents that separates them into two categories (Figure 7.3). Documents for which the classifier is still uncertain are placed closer to the border of these two areas, so the user can specifically pick one of these documents and classify it to train the automatic classifier most efficiently. Various other views provide further insight into the corpus, the classification, and individual text documents. This supports a **classification task**: Create a method that will automatically sort most of the text documents into one of two categories defined by a number of pre-classified examples.

HierarchicalTopics is a visual analytics system by Dou et al. (2013), which manages a hierarchy of topics found in a text corpus and visualizes both the hierarchical topic structure and the occurrences of topics over time (Figure 7.4). While analysing the data set, the user can modify the topic hierarchy by joining, absorbing, or collapsing topics to reflect insights gained from the visualizations. This supports an **exploration task**: Make sense of an unknown data collection, discover an overall theme or individual, significant facts.

PivotPaths by Dörk et al. (2012) invites users to ‘stroll’ through a data space with facets and relations (Figure 7.5). It shows a layer of resources in between a layer of people and a layer of concepts. Elements can be chosen as anchors, which rearranges the view to show the relations of this element. Animation is used to ensure smooth and gradual transitions between views. This supports a **navigation task**: Enable the user to move through a complex graph while providing a sense of context and pointing out possibly interesting routes to follow.

7.3 Closing remarks

Visual analytics research often regards understanding a data set as the aim of the analytic process. This corresponds to the exploration task outlined above. However, understanding alone is often not the ultimate goal of domain experts, whose daily tasks involve many other tasks with much

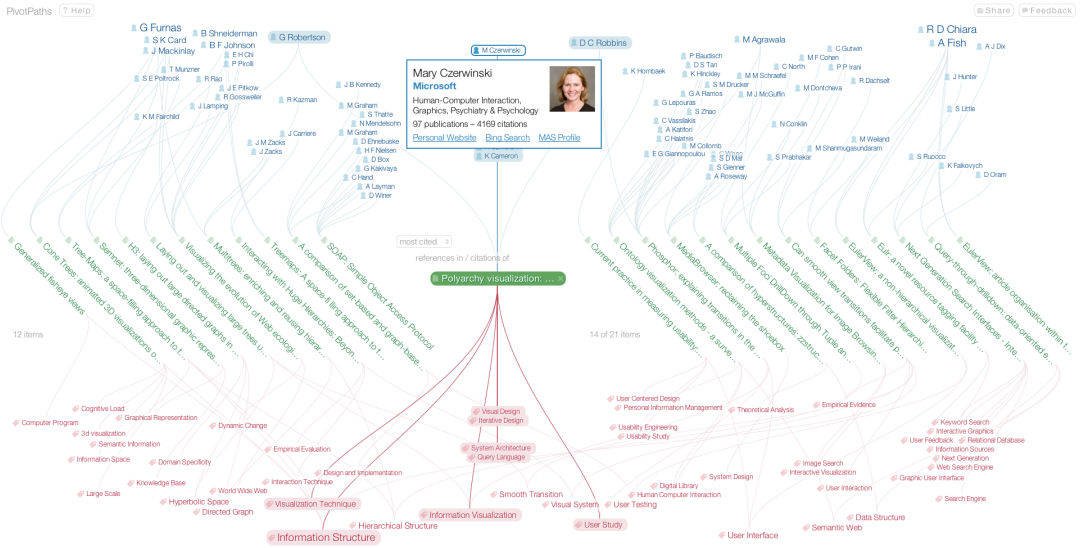


Figure 7.5: *PivotPaths* by Dörk et al. (2012) supports a navigation task. ©2012 IEEE

more tangible objectives. The classification in the previous section was derived from the systems developed in the course of this thesis but—as has been shown by the examples of other research—is not limited to them. It may serve as a taxonomy for further research and, in time, may be extended with additional analysis tasks.

8

Outlook

Computes are tools. One aspect of computer science is making these tools available to as wide a range of applications as possible. In this role, computer science is an enabling technology, which helps people perform their tasks better, with less effort, or more comfortably. It may also make things possible in the first place. To provide these benefits, applied computer science requires use cases and application domains. Computers have already been used for many different tasks and in many different domains with great success. They have had as profound an effect on society, technology, and science as few other inventions of the past centuries. And still, the journey feels far from over. Visual analytics in particular depends on relevant application cases. Theoretical advances are often either on the visualization or the automatic data analysis side, which both have their own research communities. The most impressive practical advances, however, are often achieved in collaboration with practitioners from outside the field.

At the very least, the visual analytics researcher requires three things: data, questions, and evaluation. Visual analytics is meant to support an analytical reasoning process and as such, it requires data or information to base this reasoning on. This data has to exist and it has to be available. Not all data that could theoretically be recorded is actually stored. Unless this data can be used to a company's advantage, there is little incentive to invest in the necessary infrastructure. Without an existing data base, however, it is often difficult to develop profitable methods of using it. If data exists, it may not be readily available to researchers due to concerns for confidentially and the lack of a clearly visible benefit to be gained from

investing time and effort into a cooperation with researchers. The question for the benefit, for the kind of improvement that can be expected from applying visual analytics to their particular use case is a very valid one. It is also often a difficult one to answer in advance, especially when, as is often the case, the practitioners do not see an immediate need for changes and improvements to their day-to-day processes and it is up to the computer scientists to discover and point out potential.

The first step in discovering this potential is asking the right questions. As the previous chapters demonstrated, visual analytics can support a number of different analysis tasks, including identification, optimization, classification, as well as data exploration and navigation. A detailed analysis can produce many different results, but not all of these are relevant to practitioners. Computer scientists often lack the required insight into an application domain to effortlessly identify relevant analysis questions. However, in collaboration with the experts, by interviewing and demonstrating the tools and possibilities available, one may find those question that can be answered better or at all using visual analytics. Provided a system can be built to answer these questions satisfactorily using the available data, the experts are needed for a third time to evaluate the results. One part of this is whether the answers found using the visual analytics system are correct in their terms, another whether they are useful. A successful first evaluation and thus demonstration of the visual analytics method may then spark new ideas for other analysis questions.

A vague prospect for benefits is not the only reason for possible initial scepticism among the practitioners. As any other method of automation, visual analytics is faced with the question of whether it intends to replace the human expert. It does not, as the inclusion of human reasoning in the analysis process is one of the very elements of the visual analytics approach. In fact, it is specifically addressed at those problems that cannot be solved purely automatically and intends to provide automatic support to those processes that depend on a human contribution. In spite of that, the attempt to develop a visual analytics system may result in discovering that a particular problem may well be solved automatically. However, increasing the problem solving power is not (or should not be) the only reason for a visual analytics system to include human reasoning. Visual analytics not only supports the search for a solution but also provides the necessary insight that enables domain experts to make informed decisions and ensures the necessary accountability and provenance. Also, automatic approaches require a formal definition of any conditions and constraints that are to be observed. A complete formalization of all factors and implicit knowledge that contribute to an analysis is not always feasible. In the VAST contest example, after having reduced the set of possible solution to

just two remaining candidates, simply visualizing them and discarding the one that was ‘obviously’ (from the human perspective) incorrect seemed easier than to devise and implement a formal definition of this reasoning. Similarly, Sedlmair et al. (2012) write that their target users ‘heavily relied on this implicit knowledge about costs and constraints during the process of proposing, implementing, and validating changes.’

Scepticism can also arise when practitioners view the visual analytics approach as a mere competitor of the established solutions and see no reason to support this competition when the current systems work well. However, visual analytics research does not usually intend to immediately replace existing systems but to explore the possibilities and develop methods which may then be incorporated into existing or new systems by the practitioners themselves or by software engineers. Consequently, this research rarely results in fully-fledged software products ready for productive use. The systems developed as part of this thesis all leave unfinished business when regarded as industrial products. This concerns their usability and reliability but also their functionality: The case study for the planning of a reconfigurable manufacturing layout used only a single product variant. The system supports multiple variants as it is, but how the interactions between these variants and the effects of different strategies of where and when to introduce which variants into the process can be communicated most effectively, has not yet been examined. When planning actual processes, users may want to keep considerable more candidate layouts and so the handling and categorization of layouts may have to be improved, possibly with a clear history of modifications and results for both manually and automatically designed layouts. Also, the iTRAME platform used is a prototype platform unlikely to be used for an actual production process, so the cost of adapting the system to other reconfigurable manufacturing systems is something worth investigating. Determining the usefulness of the machine diagnostics system and identifying the analytic components that are most useful for classifying machine data would require a data base larger than what this particular company currently collects. The *Frida* framework was used to perform several analyses on the vehicle data set. These are merely a starting point and domain experts will require additional analytical functionality before the system is viable for productive use. For example, domain experts may be interested in analysing how radio coverage changes over time and in a general assessment of how closely the actual vehicle movements follow the published schedule and whether deviations might hint at any systematic problems.

While none of these systems are ready for productive use, they can serve as a practical demonstration of the visual analytics idea to the experts in the respective domains. The expert from the machine tool manufacturer

concluded that he can imagine to use a system of this kind for diagnosing machines and that he might have to look more into the possibilities of visual analysis. The experts from the public transport provider said that the system shown to them does things they can already do but ‘not with as little effort, not in a systematic way, and not in an automated way’. This increased awareness of what is possible may prompt a more regular and more systematic collection of data and the eventual introduction of visual analytics as a valuable element of processes in a wide range of application domains.

Bibliography

Abouelaoualim, A., Das, K., Faria, L., Manoussakis, Y., Martinhon, C., and Saad, R. (2008), 'Paths and Trails in Edge-Colored Graphs', in Laber, E. S., Bornstein, C., Nogueira, L. T., and Faria, L., eds., *LATIN 2008: Theoretical Informatics*, Lecture Notes in Computer Science, 4957 (Berlin: Springer), 723–735. (Cited on page 70.)

Ahlberg, C. and Shneiderman, B. (1994), 'The Alphaslider: A Compact and Rapid Selector', in *Proceedings of the SIGCHI '94 Conference on Human Factors in Computing Systems – CHI '94* (New York: ACM), 365–371. (Cited on pages 131 and 132.)

Aigner, W., Miksch, S., Schumann, H., and Tominski, C. (2011), *Visualization of Time-Oriented Data*, Human-Computer Interaction Series (London: Springer). (Cited on pages 33 and 130.)

Andrienko, N. and Andrienko, G. (2013), 'Visual analytics of movement: An overview of methods, tools and procedures', *Information Visualization*, 12/1, 3–24. (Cited on page 107.)

Appert, C. and Fekete, J.-D. (2006), 'OrthoZoom Scroller: 1D Multi-Scale Navigation', in *Proceedings of the SIGCHI '06 Conference on Human Factors in Computing Systems* (New York: ACM), 21–30. (Cited on pages 132, 133, and 136.)

Bäck, T. and Schwefel, H.-P. (1993), 'An Overview of Evolutionary Algorithms for Parameter Optimization', *Evolutionary Computation*, 1/1, 1–23. (Cited on page 22.)

Bederson, B. B. (2000), 'Fisheye menus', in *Proceedings of the 13th annual ACM symposium on User interface software and technology* (New York: ACM), 217–225. (Cited on page 10.)

Benzmüller, C. and Woltzenlogel Paleo, B. (2013), 'Formalization, Mechanization and Automation of Gödel's Proof of God's Existence', *ArXiv e-prints*, arXiv:1308.4526 [cs.LO]. (Cited on page 8.)

Berger, W., Piringer, H., Filzmoser, P., and Gröller, E. (2011), 'Uncertainty-Aware Exploration of Continuous Parameter Spaces Using Multivariate Prediction', *Computer Graphics Forum*, 30/3, 911–920. (Cited on page 19.)

Berthold, M. R., Cebron, N., Dill, F., Gabriel, T. R., Kötter, T., Meinl, T., Ohl, P., Thiel, K., and Wiswedel, B. (2009), 'KNIME – The Konstanz Information

Miner: Version 2.0 and Beyond', *ACM SIGKDD Explorations Newsletter*, 11/1, 26–31. (Cited on pages 90 and 107.)

Blinn, J. F. (1977), 'Models of light reflection for computer synthesized pictures', *Computer Graphics*, 11/2, 192–198. (Cited on page 56.)

Booshehrian, M., Möller, T., Peterman, R. M., and Munzner, T. (2012), 'Vismon: Facilitating Analysis of Trade-Offs, Uncertainty, and Sensitivity In Fisheries Management Decision Making', *Computer Graphics Forum*, 31/3pt3, 1235–1244. (Cited on pages 19 and 33.)

Bosch, H., Heinrich, J., Müller, C., Höferlin, B., Reina, G., Höferlin, M., Wörner, M., and Koch, S. (2009), 'Innovative Filtering Techniques and Customized Analytics Tools', in *Proceedings of the 2009 IEEE Symposium on Visual Analytics Science and Technology – VAST 2009* (IEEE), 269–270. (Cited on pages 4 and 12.)

Bosch, H., Thom, D., Wörner, M., Koch, S., Püttmann, E., Jäckle, D., and Ertl, T. (2011), 'ScatterBlogs: Geo-Spatial Document Analysis', in *Proceedings of the 2011 IEEE Conference on Visual Analytics Science and Technology – VAST 2011* (IEEE), 309–310. (Cited on page 4.)

Bosch, H., Thom, D., Heimerl, F., Püttmann, E., Koch, S., Krüger, R., Wörner, M., and Ertl, T. (2013), 'ScatterBlogs2: Real-Time Monitoring of Microblog Messages through User-Guided Filtering.', *IEEE Transactions on Visualization and Computer Graphics*, 19/12, 2022–2031. (Cited on pages 4, 141, and 142.)

Boyd, S. and Vandenberghe, L. (2004), *Convex Optimization* (Cambridge: Cambridge University Press). (Cited on page 21.)

Butler, P., Chakraborty, P., and Ramakrishan, N. (2012), 'The Deshredder: A Visual Analytic Approach to Reconstructing Shredded Documents', in *Proceedings of the 2012 IEEE Conference on Visual Analytics Science and Technology – VAST 2012* (IEEE), 113–122. (Cited on pages 152 and 153.)

Cattell, R. (2011), 'Scalable SQL and NoSQL Data Stores', *SIGMOD Record*, 39/4, 12–27. (Cited on page 109.)

Cedilnik, A., Geveci, B., Moreland, K., Ahrens, J., and Favre, J. (2006), 'Remote large data visualization in the ParaView framework', in *Proceedings of the 6th Eurographics conference on Parallel Graphics and Visualization – EG PGV'06* (Aire-la-Ville: Eurographics Association), 163–170. (Cited on page 106.)

Champion, J., Champoin, C., and Sullivan, R. O. (2012), ZedGraph, <http://sourceforge.net/projects/zedgraph/>, accessed 17 Oct. 2013. (Cited on page 93.)

Chan, S. M., Xiao, L., Gerth, J., and Hanrahan, P. (2008), 'Maintaining Interactivity While Exploring Massive Time Series', in *Proceedings of the 2008 IEEE Symposium on Visual Analytics Science and Technology – VAST 2008* (IEEE), 59–66. (Cited on page 106.)

Chi, E. H. (2000), 'A Taxonomy of Visualization Techniques using the Data State Reference Model', in *Proceedings of the IEEE Symposium on Information Visualization 2000 – InfoVis 2000* (IEEE), 69–75. (Cited on page 105.)

Christofides, N. (1976), 'Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem', Management Sciences Research Report 388, Carnegie-Mellon University, Pittsburgh. (Cited on page 22.)

Cooley, J. W. and Tukey, J. W. (1965), 'An Algorithm for the Machine Calculation of Complex Fourier Series', *Mathematics of Computation*, 19/90, 297–301. (Cited on page 93.)

Dangelmaier, W., Fischer, M., Gausemeier, J., Grafe, M., Matysczok, C., and Mueck, B. (2005), 'Virtual and augmented reality support for discrete manufacturing system simulation', *Computers in Industry*, 56/4, 371–383. (Cited on page 33.)

Darwin, C. (1859), *On the Origin of Species by Means of Natural Selection* (London: Murray). (Cited on page 78.)

Dijkstra, E. W. (1959), 'A Note on Two Problems in Connexion with Graphs', *Numerische Mathematik*, 1/1, 269–271. (Cited on page 75.)

Doleisch, H. (2007), 'SIMVIS: Interactive Visual Analysis of Large and Time-dependent 3D Simulation Data', in *Proceedings of the 2007 Winter Simulation Conference* (Piscataway: IEEE Press), 712–720. (Cited on page 106.)

Doleisch, H. and Hauser, H. (2002), 'Smooth Brushing for Focus+Context Visualization of Simulation Data in 3D', *Journal of WSCG*, 10/1-3, 147–154. (Cited on page 33.)

Dou, W., Yu, L., Wang, X., Ma, Z., and Ribarsky, W. (2013), 'HierarchicalTopics: Visually Exploring Large Text Collections Using Topic Hierarchies', in *Proceedings of the 2013 IEEE Conference on Visual Analytics Science and Technology – VAST 2013* (IEEE), 2002–2011. (Cited on pages 154 and 155.)

Dörk, M., Henry Riche, N., Ramos, G., and Dumais, S. (2012), 'PivotPaths: Strolling through Faceted Information Spaces', *IEEE Transactions on Visualization and Computer Graphics*, 18/12, 2709–2718. (Cited on pages 155 and 156.)

ElMaraghy, H. A. (2005), 'Flexible and reconfigurable manufacturing systems paradigms', *International Journal of Flexible Manufacturing Systems*, 17/4, 261–276. (Cited on page 32.)

Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996), 'A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise', in *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining* (Menlo Park: AAAI Press), 226–231. (Cited on page 93.)

Ford, H. and Crowther, S. (1922), *My Life and Work* (Garden City: Doubleday, Page & company). (Cited on page 30.)

Furnas, G. W. (1986), 'Generalized Fisheye Views', in *Proceedings of the SIGCHI '86 Conference on Human Factors in Computing Systems – CHI '86* (New York: ACM), 16–23. (Cited on page 10.)

Giereth, M., Wörner, M., Bosch, H., Baier, P., and Ertl, T. (2008), 'Utilization of Semantic Annotations in Interactive User Interfaces for Large Documents', in Hegering, H.-G., Lehmann, A., Ohlbach, H. J., and Scheideler, C., eds., *INFORMATIK 2008: Beherrschbare Systeme – dank Informatik*, ii, Lecture Notes in Informatics, 134 (Bonn: Gesellschaft für Informatik), 706–711. (Cited on page 4.)

Gouraud, H. (1971), 'Continuous Shading of Curved Surfaces', *IEEE Transactions on Computers*, C-20/6, 623–629. (Cited on page 57.)

Haber, R. B. and McNabb, D. A. (1990), 'Visualization Idioms: A Conceptual Model for Scientific Visualization Systems', in *Visualization in Scientific Computing* (IEEE Computer Society Press), 74–93. (Cited on page 105.)

Haklay, M. and Weber, P. (2008), 'OpenStreetMap: User-Generated Street Maps', *Pervasive Computing*, 7/4, 12–18. (Cited on page 113.)

Hameed, B., Minguez, J., Wörner, M., Hollstein, P., Zor, S., Silcher, S., Dürr, F., and Rothermel, K. (2011), 'The Smart Real-Time Factory as a Product Service System', in Hesselbach, J. and Herrmann, C., eds., *Functional Thinking for Value Creation* (Heidelberg: Springer), 326–331. (Cited on page 4.)

Han, J., Kamber, M., and Pei, J. (2006), *Data Mining: Concepts and Techniques* (2nd edn, San Francisco: Morgan Kaufmann). (Cited on page 9.)

Hand, D., Mannila, H., and Smyth, P. (2001), *Principles of Data Mining* (Cambridge, Massachusetts: MIT Press). (Cited on page 9.)

Hao, M., Dayal, U., Keim, D., and Schreck, T. (2007), 'Multi-Resolution Techniques for Visual Exploration of Large Time-Series Data', in *EuroVis07: Joint Eurographics / IEEE VGTC Symposium on Visualization* (Aire-la-Ville: Eurographics Association). (Cited on page 132.)

Heimerl, F., Koch, S., Bosch, H., and Ertl, T. (2012), 'Visual classifier training for text document retrieval', *Visualization and Computer Graphics, IEEE Transactions on*, 18/12, 2839–2848. (Cited on pages 154 and 155.)

Huynh, D. F. (2006), SIMILE: Timeline, <http://www.simile-widgets.org/timeline/>, accessed 2 Oct. 2013. (Cited on page 132.)

Imai, M. (1986), *Kaizen: The Key to Japan's Competitive Success* (New York: McGraw-Hill). (Cited on page 31.)

Jardine, A. K. S., Lin, D., and Banjevic, D. (2006), 'A review on machinery diagnostics and prognostics implementing condition-based maintenance', *Mechanical Systems and Signal Processing*, 20/7, 1483–1510. (Cited on page 88.)

Keim, D., Kohlhammer, J., Ellis, G., and Mansmann, F., eds. (2010), *Mastering The Information Age: Solving Problems with Visual Analytics* (Goslar: Eurographics Association). (Cited on pages 7, 10, 11, and 109.)

Klopper, B., Honiden, S., Pater, J.-P., and Dangelmaier, W. (2011), 'Decision Making in adaptive Manufacturing Systems: Multi-Objective Scheduling and User Interface', in *Proceedings of the 2011 IEEE Symposium on Computational Intelligence in Control and Automation – CICA 2011* (IEEE), 123–130. (Cited on page 33.)

Krüger, R., Thom, D., Wörner, M., Bosch, H., and Ertl, T. (2013), 'TrajectoryLenses – A Set-based Filtering and Exploration Technique for Long-term Trajectory Data', *Computer Graphics Forum*, 32/3pt4, 451–460. (Cited on pages 4 and 141.)

Law, A. M. and Kelton, W. D. (1999), *Simulation Modeling and Analysis* (3rd edn: McGraw-Hill). (Cited on page 76.)

Lee, E. A. (2006), 'The Problem with Threads', *Computer*, 39/5, 33–42. (Cited on page 117.)

Leung, Y. K. and Apperley, M. D. (1994), 'A review and Taxonomy of Distortion-Oriented Presentation Techniques', *ACM Transactions on Computer-Human Interaction – TOCHI*, 1/2, 126–160. (Cited on page 10.)

Liu, R., Paxton, W. A., Choe, S., Ceradini, D., Martin, S. R., Horuk, R., MacDonald, M. E., Stuhlmann, H., Koup, R. A., and Landau, N. R. (1996), 'Homozygous Defect in HIV-1 Coreceptor Accounts for Resistance of Some Multiply-Exposed Individuals to HIV-1 Infection', *Cell*, 86/3, 367–377. (Cited on page 80.)

Lundblad, P., Jern, M., and Forsell, C. (2008), 'Voyage Analysis Applied to Geovisual Analytics', in *Proceedings of the 12th International Conference on Information Visualisation – IV '08* (IEEE), 381–388. (Cited on page 107.)

MacQueen, J. (1967), 'Some Methods for Classification and analysis of Multivariate Observations', in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, i (Berkeley: University of California Press), 281–297. (Cited on page 93.)

Mehrabi, M. G., Ulsoy, A. G., and Koren, Y. (2000), 'Reconfigurable manufacturing systems: Key to future manufacturing', *Journal of Intelligent Manufacturing*, 11, 403–419. (Cited on page 32.)

Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., and Euler, T. (2006), 'Yale: Rapid Prototyping for Complex Data Mining Tasks', in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining – KDD'06* (New York: ACM), 935–940. (Cited on page 90.)

Mitchell, T. M. (1997), *Machine Learning* (New York: McGraw-Hill). (Cited on page 9.)

Montigel, M. (1992), 'Representation of track topologies with double vertex graphs', in *Proceedings of the 3rd International Conference on Computer Aided Design, Manufacture, and Operation in the Railway and Other Advanced Mass Transit Systems*, ii (Southampton: Computational Mechanics Publications), 359–370. (Cited on page 70.)

Ong, S. H., Goh, K. M., Chan, H. L., Lim, T. Y., and Ling, K. V. (2010), 'Towards Machine Diagnostics on Chip', in *Proceedings of the 11th International Conference on Control Automation Robotics & Vision – ICARCV* (IEEE), 1353–1358. (Cited on page 88.)

Piringer, H., Tominski, C., Muigg, P., and Berger, W. (2009), 'A Multi-Threading Architecture to Support Interactive Visual Exploration', *IEEE Transactions on Visualization and Computer Graphics*, 15/6, 1113–1120. (Cited on page 106.)

Piringer, H., Berger, W., and Krasser, J. (2010), 'HyperMoVal: Interactive Visual Validation of Regression Models for Real-Time Simulation', *Computer Graphics Forum*, 29/3, 983–992. (Cited on pages 19 and 33.)

Riffelmacher, P., Kluge, S., Kreuzhage, R., Hummel, V., and Westkämper, E. (2007), 'Learning Factory for the Manufacturing Industry: Digital Learning Shell and a Physical Model Factory – iTRAME for production engineering and improvement', in *Proceedings of the 20th International Conference on Computer-Aided Production Engineering* (Glasgow), 120–131. (Cited on page 34.)

Robertson, G. G. and Mackinlay, J. D. (1993), 'The document lens', in *Proceedings of the 6th annual ACM symposium on User interface software and technology* (New York: ACM), 101–108. (Cited on page 10.)

Rohrer, M. W. (2000), 'Seeing Is Believing: The Importance of Visualization in Manufacturing Simulation', in *Proceedings of the 2000 Winter Simulation Conference* (San Diego: Society for Computer Simulation International), 1211–1216. (Cited on page 33.)

Rother, M. and Shook, J. (2003), *Learning to See: Value-Stream Mapping to Create Value and Eliminate Muda* (Cambridge, MA: The Lean Enterprise Institute). (Cited on page 31.)

Sarkar, M. and Brown, M. H. (1992), 'Graphical Fisheye Views of Graphs', in *Proceedings of the SIGCHI '92 Conference on Human Factors in Computing Systems – CHI '92* (New York: ACM), 83–91. (Cited on page 10.)

Schrijver, A. (2003), *Combinatorial Optimization: Polyhedra and Efficiency* (Berlin: Springer). (Cited on page 21.)

Schulze, T., Schumann, M., and Rehn, G. D. (2000), 'Language Based Simulation Models as Management Tools for Assembly Lines', in *Proceedings of the 2000 Winter Simulation Conference* (San Diego: Society for Computer Simulation International), 1393–1401. (Cited on page 33.)

Sedlmair, M., Frank, A., Munzner, T., and Butz, A. (2012), 'RelEx: Visualization for Actively Changing Overlay Network Specifications', *IEEE Transactions on Visualization and Computer Graphics*, 18/12, 2729–2738. (Cited on pages 152, 153, and 159.)

Spence, R. and Apperley, M. (1982), 'Data base navigation: an office environment for the professional', *Behaviour & Information Technology*, 1/1, 43–54. (Cited on page 9.)

Spencer, H. (1864), *The Principles of Biology* (London: Williams and Norgate). (Cited on page 79.)

Tang, J., Zhao, L., Yu, W., Yue, H., and Chai, T. (2010), 'Soft Sensor Modeling of Ball Mill Load via Principal Component Analysis and Support Vector Machines', in Zeng, Z. and Wang, J., eds., *Advances in Neural Network*

Research and Applications, Lecture Notes in Electrical Engineering, 67 (Berlin: Springer), 803–810. (Cited on page 88.)

Thom, D., Bosch, H., Koch, S., Wörner, M., and Ertl, T. (2012), 'Spatiotemporal Anomaly Detection through Visual Analysis of Geolocated Twitter Messages', in *Proceedings of the 2012 IEEE Pacific Visualization Symposium – PacificVis 2012* (IEEE), 41–48. (Cited on page 4.)

Thomas, J. and Kielman, J. (2009), 'Challenges for visual analytics', *Information Visualization*, 8/4, 309–314. (Cited on pages 2 and 11.)

Thomas, J. J. and Cook, K. A., eds. (2005), *Illuminating the Path: The Research and Development Agenda for Visual Analytics* (IEEE Computer Society Press). (Cited on pages 2, 7, 11, and 103.)

Verl, A., Heisel, U., Walther, M., and Maier, D. (2009), 'Sensorless automated condition monitoring for the control of the predictive maintenance of machine tools', *CIRP Annals – Manufacturing Technology*, 58/1, 375–378. (Cited on page 88.)

von Ferber, C., Holovatch, T., Holovatch, Y., and Palchykov, V. (2009), 'Public transport networks: empirical analysis and modeling', *The European Physical Journal B*, 68/2, 261–275. (Cited on page 107.)

Wang, J., Gao, R. X., and Yan, R. (2012), 'A hybrid approach to bearing defect diagnosis in rotary machines', *CIRP Journal of Manufacturing Science and Technology*, 5/4, 357–365. (Cited on page 88.)

Wenzel, S., Bernhard, J., and Jessen, U. (2003), 'A Taxonomy of Visualization Techniques for Simulation in Production and Logistics', in *Proceedings of the 2003 Winter Simulation Conference* (New Orleans), 729–736. (Cited on page 34.)

Westkämper, E. (2007), 'Digital Manufacturing in the Global Era', in Cunha, P. F. and Maropoulos, P. G., eds., *Digital Enterprise Technology: Perspectives and Future Challenges* (Springer US), 3–14. (Cited on page 30.)

Wierse, A. (1995), 'Performance of the collaborative visualization environment (COVISE) visualization system under different conditions', in *Visual Data Exploration and Analysis II*, Proceedings of SPIE, 2410 (SPIE), 218–229. (Cited on page 106.)

Wörner, M. and Ertl, T. (2011a), 'Multi-Dimensional Distorted 1D Navigation', in *Proceedings of the 2011 International Conference on Information Visualization Theory and Applications – IVAPP 2011* (SciTePress), 198–203. (Cited on page 4.)

Wörner, M. and Ertl, T. (2011b), 'Visual Analysis of Advanced Manufacturing Simulations', in *EuroVA 2011: International Workshop on Visual Analytics* (Goslar: Eurographics Association), 29–32. (Cited on page 4.)

Wörner, M. and Ertl, T. (2012), 'Visual Analysis of Public Transport Vehicle Movement', in *EuroVA 2012: International Workshop on Visual Analytics* (Goslar: Eurographics Association), 79–83. (Cited on pages 4 and 127.)

Wörner, M. and Ertl, T. (2013a), 'Simulation-based Visual Layout Planning in Advanced Manufacturing', in *Proceedings of the 46th Hawaii International Conference on System Sciences – HICSS 2013* (IEEE), 1532–1541. (Cited on page 4.)

Wörner, M. and Ertl, T. (2013b), 'SmoothScroll: A Multi-scale, Multi-layer Slider', in Csurka, G., Kraus, M., Mestetskiy, L., Richard, P., and Braz, J., eds., *Computer Vision, Imaging and Computer Graphics. Theory and Applications*, Communications in Computer and Information Science, 274 (Heidelberg: Springer), 142–154. (Cited on page 4.)

Wörner, M. and Ertl, T. (2014), 'Retaining Interactivity in a Visual Analytics System for Massive Public Transportation Data Sets', in *Proceedings of the 47th Hawaii International Conference on System Sciences – HICSS 2014* (IEEE), to appear. (Cited on pages 4, 110, 111, 112, 118, and 122.)

Wörner, M., Reina, G., Grottel, S., and Ertl, T. (2010), 'Vide: an editor for the visual exploration of raw data', in *Visualization and Data Analysis 2010*, Proceedings of SPIE-IS&T Electronic Imaging, 7530 (Bellingham: SPIE), 75300C–75300C–12. (Cited on page 4.)

Wörner, M., Metzger, M., and Ertl, T. (2013), 'Dataflow-based Visual Analysis for Fault Diagnosis and Predictive Maintenance in Manufacturing', in *EuroVA 2013: International Workshop on Visual Analytics* (Leipzig: Eurographics Association), 55–59. (Cited on page 4.)

Young, D. and Shneiderman, B. (1993), 'A Graphical Filter/Flow Representation of Boolean Queries: A Prototype Implementation and Evaluation', *Journal of the American Society of Information Science*, 44/6, 327–339. (Cited on page 107.)