Accessible SVG Charts with AChart Creator and AChart Interpreter

Christopher Alexander Kopel

Accessible SVG Charts with AChart Creator and AChart Interpreter

Christopher Alexander Kopel B.Sc.

Master's Thesis

to achieve the university degree of

Diplom-Ingenieur

Master's Degree Programme: Information and Computer Engineering

submitted to

Graz University of Technology

Supervisor

Ao.Univ.-Prof. Dr. Keith Andrews
Institute of Interactive Systems and Data Science (ISDS)

Graz, 16 May 2021

Zugängliche SVG-Charts
mit AChart Creator und AChart Interpreter

Christopher Alexander Kopel B.Sc.

Masterarbeit

für den akademischen Grad

Diplom-Ingenieur

Masterstudium: Information and Computer Engineering

an der

Technischen Universität Graz

Begutachter

Ao.Univ.-Prof. Dr. Keith Andrews
Institute of Interactive Systems and Data Science (ISDS)

Graz, 16 May 2021

Diese Arbeit ist in englischer Sprache verfasst.

Abstract

Presenting visual information to blind users may be regarded as one of the most complex challenges in the field of accessibility. The confluence of data visualisations moving to the web and initiatives towards web accessibility have led to a particularly promising approach: enriching SVG-based charts with underlying data and semantic information in machine-readable form using ARIA roles and properties.

This thesis first surveys current approaches to web accessibility, chart accessibility, and the semantic enrichment of SVG charts, concentrating on charts of tabular data, such as line, bar, and pie charts. A number of proposed taxonomies of ARIA roles and properties for accessible SVG charts are discussed and compared, leading to a new aggregate taxonomy, the AChart (Accessible Chart) taxonomy.

The remainder of the thesis presents AChart, a suite of open-source software tools written in TypeScript with Node.js, which currently supports bar charts, line charts, and pie charts. AChart Creator is a command-line tool which generates accessible SVG charts from CSV files using the D3 framework and injecting ARIA roles and properties from the AChart taxonomy. AChart Interpreter is a client-side web application which interprets an accessible SVG chart, displays side-by-side graphical and textual versions of the chart, and can read out the chart using synthetic speech. Its user interface is screen reader compatible, so it can be used by blind users to gain an understanding of a chart, as well as by developers and chart authors to verify and validate the accessibility markup of an SVG chart. AChart Summariser is a command-line tool which interprets an accessible SVG chart and outputs a textual summary of the chart.

Kurzfassung

Die Darstellung visueller Informationen für die Zielgruppe blinder NutzerInnen kann als eine der komplexesten Herausforderungen auf dem Fachgebiet Accessibility betrachtet werden. Das Zusammenspiel zweier Faktoren, nämlich der steigenden Zahl an im Web veröffentlichten Datenvisualisierungen einerseits und der Initiativen für Barrierefreiheit andererseits, haben in einen besonders vielversprechenden Ansatz gemündet: die Anreicherung SVG-basierter Charts mit ihren zugrundeliegenden Daten und semantischen Informationen in maschinell auslesbarer Form mittels ARIA-Attributen (sog. roles und properties).

Diese Arbeit untersucht zunächst aktuelle Ansätze der Web-Accessibility, der Erstellung zugänglicher Charts und der semantischen Anreicherung von SVG-Charts. Betrachtet werden hierbei auf tabellarischen Daten basierende Charts, wie zum Beispiel Linien-, Balken- und Kreisdiagramme. Mehrere vorgeschlagene Taxonomien von ARIA-Attributen für zugängliche Charts werden diskutiert, verglichen und in ein neues System überführt, die AChart- (Accessible Chart) Taxonomie.

In den verbleibenden Kapiteln dieser Arbeit wird AChart vorgestellt, eine Sammlung quelloffener Software-Werkzeuge, die in TypeScript und mit Node.js entwickelt wurde und in der aktuellen Version Linien-, Balken- und Kreisdiagramme unterstützt. AChart Creator ist ein Kommandozeilenprogramm, welches unter Verwendung des D3-Frameworks zugängliche SVG-Charts aus CSV-Dateien generiert und mit ARIA-Attributen aus der AChart-Taxonomie versieht. AChart Interpreter ist eine Client-seitige Webanwendung, die ein zugängliches SVG-Chart interpretiert, nebeneinander grafische und textuelle Versionen des Charts anzeigt und das Chart per Sprachsynthese vorlesen kann. Die Benutzeroberfläche wurde barrierefrei gestaltet, so dass blinde Nutzer das Programm dazu einsetzen können, ein Chart zu verstehen. Gleichzeitig kann es von Entwicklern und Chart-Autoren für die Verifizierung und Validierung des Accessibility-Markups von SVG-Charts verwendet werden. AChart Summariser ist ein Kommandozeilen-Werkzeug, welches ein zugängliches SVG-Chart interpretiert und eine textuelle Zusammenfassung des Charts ausgibt.

Contents

List of Figures

List of Tables

List of Listings

Acknowledgements

I would like to express my thanks to Prof. Keith Andrews at ISDS for giving me the opportunity to accomplish my desire to write my master's thesis about a subject in the field of web accessibility. In his Human-Computer Interaction course at TU Graz, he taught me the essentials of usability and gave me the opportunity to conduct an extensive accessibility evaluation of the web site by the Graz public transport services. During this course and on the long way of composing my thesis, I gained a profound understanding and learned numerous interesting details about data visualisation, usability, and accessibility, which will definitely be helpful for my future life as a computer engineer.

Many thanks also go to TU Graz as a whole for giving me as a blind person the chance to study in the domains of electrical, sound, and computer engineering under fair circumstances. Whether providing me lecture notes and exam sheets in LaTeX or other digital formats, granting additional time for certain tasks, or discussing all my questions – there was never any problem to agree on a good solution for overcoming visual barriers. Moreover, I would like to thank Jakob and the entire team of Zentrum Integriert Studieren at KFU Graz for adapting dozens of graphics and formulae as well as thousands of pages of other learning material according to my needs. Especially during the first years of my studies, this assistance was of great help.

I am indebted to Sigi, Barbara, and Jasmin at Help Tech GmbH for providing me photos and other useful sources, for assisting me in certain tasks of web research, and for giving me important feedback, especially in debugging the visual output of AChart Creator and Interpreter. In this context, I would also like to thank Philipp for his help.

Furthermore, I thank all my friends who would not stop praying or keeping their fingers crossed for my many exams and practicals and, finally, for the completion of this thesis as well! In particular, I would like to thank Marie, Nadine, and Thomas for testing the accessibility and usability of AChart Interpreter. Their comments and suggestions encouraged me in the development of the application and contributed to making AChart Interpreter a powerful tool for blind users.

Very warm thanks also go to my dear parents, without whom my studies would not have been possible. Despite all obstacles and uncertainties, they always gave me financial support, the freedom to choose my field of career, and the affirmation that I am on the right track.

My final thanks are dedicated to my wonderful girlfriend Annika, not only for testing AChart Interpreter regularly during its development or giving me advice as a professional in writing good English. She always stayed by my side as a true companion, on all the hills and through all the valleys on my way towards achieving my master's degree, understanding me, comforting me, cheering me up, encouraging me, and giving me the energy I needed to move on.

Christopher Kopel
Graz, Austria, 16 May 2021

Credits

I would like to thank the following individuals and organisations for permission to use their material:

- The thesis was written using Keith Andrews' skeleton thesis [Andrews 2018].
- Figures 2.1 to 2.4 are used with kind permission of Help Tech [2021].
- Figure 3.1 is used with kind permission of Orbit [2021].
- Figure 4.1 is used with kind permission of Keith Andrews, Graz University of Technology.

Chapter 1

Introduction

“ A picture is worth a thousand words. ”

[Internationally known proverb; quoted from [Gardner and Bulatov 2006, page 1243]]

Visualisations play an essential role across many types of information representations as a utility for conveying data and their relationships in a clear, intuitive manner. They are omnipresent not only in scientific and technical publications, but also in educational materials and popular media. With the advances in information technology, it has become possible to store and to automatically depict large amounts of data, as can be seen, for instance, in the visualisations of stock prices, election results, or pandemic incidences. A static, merely graphical representation, however, does not accommodate the needs of all potential target groups: blind persons are entirely excluded from information which is only available visually. People with some residual vision may be able to perceive graphics in general, but often require optical adaptations, such as enlargement, simplification, or colour modifications. Individuals with cognitive or learning disabilities can notably benefit from illustrations; nevertheless, for this target group, it is particularly important that the visualisations show a proportionate level of complexity.

In science and industry, as well as in recent standardisation efforts and consumer domains, modern technologies are being applied to make data visualisations accessible to recipients with special needs. Some of the strategies focus on converting an existing image to an alternative representation, or providing a textual description alongside an image. A different approach is to produce semantically-encoded data visualisations, which can adapt to the needs and preferences of an individual user. The semantics of the visualisation and the underlying data are provided in a machine-readable form alongside, within, or in place of a particular visual form. Strategies following this principle can be categorised under the *inclusive design* paradigm and are mostly related to modern web development.

This master’s thesis examines various options to create accessible charts by means of Scalable Vector Graphics (SVG) [W3C 2011], in conjunction with the Accessible Rich Internet Applications (ARIA) system [W3C 2017a], both standardised by the World Wide Web Consortium (W3C) [W3C 2021c].

The term *data visualisation* includes a wide variety of types of chart and graphics, including maps, flow diagrams, and network graphs. This thesis, however, will concentrate on charts of *tabular data*, such as line charts, bar charts, and pie charts. Tabular data can often be found in spreadsheets, where one or more dimensions are stored as columns, and one or more records (data points) are stored as rows. The data in each dimension can be numerical or categorical. One dimension often contains a text string with a name for each record.

In particular, the following types of charts are often found in the literature:

- *Cartesian Charts*: for instance, *bar charts* (also known as bar graphs), *line charts* (also known as line graphs or line diagrams), and *scatter plots*.

- *Non-Cartesian Charts*: in two-dimensions, such as *pie charts*.

- *Higher-Dimensional Charts*: such as *parallel coordinates charts*, *star plots* (also known as radar charts), and cartesian or pie charts with additional information encoded as variation in colour, size, or shape of the symbols.

Other types of visualisations may be mentioned in the context of related work, but are beyond the scope of this thesis.

To ensure unambiguous and consistent use of nomenclature throughout this thesis, the following often-used terms are defined:

- *data point* or *record*: a tuple of associated components representing one item in a tabular dataset (one row of a spreadsheet).
- *dimension*: a component or value associated with every record in a tabular dataset (one column of a spreadsheet).
- *title*: In many cases, one of the data dimensions contains text strings representing a name or title for each record (which is often used to label or identify the record).
- *dataset*: the set of all the data points in a table.
- *data series*: a sequence of data points. In the case of a *single-series* chart, it is identical to the entire dataset. In the case of a *multiple-series* chart, the data points in each series share a common property. In the case of a line chart, each data series is displayed as one line on the chart.
- *discrete axis*: an axis representing a set of distinct numerical or categorical values to which the data points are bound, such as the x-axis of a bar chart.
- *continuous axis*: an axis representing a range of real numbers out of which any value may occur in the data, as in line charts and scatter plots.
- *chart object*: a logical component of a chart, such as an axis, a data series, or a data point, not necessarily corresponding to a single SVG or any other single markup element.
- *chart root*: a chart object representing the entire chart and containing all other chart objects; for instance, the root `<svg>` or `<g>` element.

This thesis focuses on three specific types of chart: line chart, bar chart, and pie chart. For these three charts, exactly two of the original dimensions are provided as input to draw the chart. These dimensions can be thought of in terms of *name-value* pairs:

- *Line chart*: Two numerical dimensions are chosen to construct the chart: one mapped to the x-axis (which can be considered the name), and the other mapped to the y-axis (which can be considered to be its corresponding value).
- *Bar chart*: A categorical dimension is typically chosen for the x-axis (the name), and a numerical dimension is mapped to the y-axis (the corresponding value).
- *Pie chart*: The title of the record is used as the name and one numerical dimension provides the value.

The accessibility of visualisations may cater to several kinds of disability. This thesis focuses on *visually impaired* recipients, which includes both *blind* and *partially sighted* persons. When using the term *blind* in this context, this neither means the total absence of any optical perception, nor does it refer to any medical or legal definition. Instead, blind individuals shall be distinguished from partially sighted ones by the single criterion that their sight, considering all possible technical aids for optical adaptation, does not suffice to visually recognise the components of a chart.

This thesis is organised into two parts. The first part embeds this work into the wider context of related work. The second part describes the practical work and the software developed as part of the thesis. The next chapter, Chapter 2, introduces the topic of accessibility for visually impaired users and looks at web accessibility in particular. Chapter 3 continues with a summary of past and present solutions towards providing visually impaired individuals access to charts, including various charting libraries and their accessibility features. Chapter 4 examines current approaches for producing adaptable and accessible graphics documents by embedding data and semantic information within SVG charts. Most of these approaches use non-standard ARIA roles and properties, which are currently not recognised by browsers and screen readers. One goal of this thesis is to contribute to the standardisation of an SVG and ARIA system for accessible charts.

For the practical part of this thesis, a software suite called AChart (Accessible Chart) was developed. Chapter 5 gives an overview of AChart and describes the AChart taxonomy of ARIA roles and properties for accessible charts. Chapter 6 describes AChart Creator, a command-line tool which generates accessible SVG charts from CSV data files. It is written in TypeScript [Microsoft 2020b] and D3 [Bostock 2021] with Node.js [OpenJS 2021b] and jsdom [jsdom 2021], and uses nexe [Boyd et al. 2020] to produce binaries for various platforms. The software is open-source and is available at [Kopel, Andrews, Mendoza Estrada, Grass et al. 2021].

Chapter 7 describes its partner, AChart Interpreter, a client-side web application which interprets an accessible SVG chart, displays side-by-side graphical and textual versions of the chart, and provides synthetic speech and Braille output for blind users. It can be used by sighted users verify and validate accessible SVG charts and by blind users as a kind of screen reader for charts. AChart Interpreter is written in TypeScript [Microsoft 2020b] and uses Electron [OpenJS 2021a] to produce executable packages for various platforms. The software is open-source and is available at [Kopel, Andrews, Mendoza Estrada, Bodner et al. 2021a]. The design of AChart Interpreter was inspired by Doug Schepers' proof-of-concept tool Describler [Schepers 2015a]. A separate spin-off command-line tool, AChart Summariser, interprets an accessible SVG chart and outputs a textual summary of the chart.

Possible future work is discussed in Chapter 8, including the extension of AChart to further chart types and the possibility of packaging a version of AChart Interpreter as a web browser extension to make suitably marked-up accessible charts more widely available to non-sighted users. Chapter 9 concludes the thesis with some final remarks. Finally, three appendices contain user guides for AChart Creator, AChart Interpreter, and AChart Summariser.

Chapter 2

Web Accessibility

“ The power of the Web is in its universality. Access by everyone regardless of disability is an essential aspect. ”

[Tim Berners-Lee, W3C Director and inventor of the World Wide Web; quoted from [AnyChart 2020a]]

Computer systems represent one of the most important tools for visually impaired persons, used in education and work as well as in private every-day life [Lang et al. 2010, page 189]. Since they support various output modalities, from visual representations through synthetic speech to tactile characters, many visually impaired humans use computers as note-takers, calendars, calculators, audio and video players, as well as for reading text on paper in conjunction with a document scanner and optical character recognition (OCR) software. Computers can act as a helpful utility for persons with other types of deficiencies too. For instance, speech recognition systems can notably assist individuals with dyslexia, cognitive, learning, hearing, or motor disabilities.

The Internet and in particular the World Wide Web (WWW, or simply the “web”), has lead to an advance in information exchange which may be considered even more valuable for visually impaired users than sighted users. Whether finding an article in an encyclopedia or a magazine, looking up the timetable for public transport, shopping, or searching for public events — for a blind person, completing one of these tasks (with few exceptions) traditionally requires some form of sighted assistance, whereas all of them (in many cases) can be independently accomplished online. However, the potential benefits of every software application or web page for users with disabilities rely heavily on one crucial criterion, its *accessibility*. Appreciating the enormous potential of the web for users with disabilities, the World Wide Web Consortium (W3C) has directed much effort towards defining standards and producing supplementary advice with the goal of creating a web accessible for all [W3C 2021b].

2.1 Definitions

Iwarsson and Ståhl [2003, page 61] define the general term *accessibility* as follows:

> Accessibility is the encounter between the person’s or group’s functional capacity and the design and demands of the physical environment. Accessibility refers to compliance with official norms and standards, thus being mainly objective in nature.

The closely related concept of usability, by contrast, deals with the effectiveness, efficiency, and satisfaction of humans when interacting with the environment and is determined by user evaluations. Thus, usability can be regarded as a rather subjective measure. As to the relationship between both terms, it can be stated that accessibility “is a necessary precondition for usability” [Iwarsson and Ståhl 2003, page 62]. Both concepts cover a wide range of disciplines, such as architecture, public transport, media, and the organisation of public events [Iwarsson and Ståhl 2003, page 57–59].

In the context of information technology, Kleynhans and Fourie [2014, page 370] summarise that: “accessibility aims to enable users and to make access to content possible for everyone, regardless of disability or the type of device that may be used.” This affects not only visual impairment but also several other types of disabilities, such as auditory, learning, language, physical, and cognitive impairments [W3C 2018f, Section 0.1]. Specific criteria of information accessibility have been defined by various institutions: the legislation of several countries prescribes how hardware and software shall interact with disabled users. The international reference standard specifying detailed requirements for the behaviour of web pages is the Web Content Accessibility Guidelines (WCAG) by the W3C and will be introduced in Section 2.3.

2.2 Assistive Technology

In general, assistive technology (AT) can be described as techniques “designed to improve the functional capabilities of people with disabilities,” [EPRS 2018a, page 3]. The spectrum of AT ranges from special appliances for food preparation, personal care, and household chores, through tools for leisure activities, to mobility aids, such as crutches and wheelchairs for physically impaired people as well as white canes used by blind persons for orientation [EPRS 2018a, pages 7–9; EPRS 2018b, page 31].

In the context of information exchange, ATs have the purpose to adapt the information channels to the available capabilities of the target users [EPRS 2018a, pages 8–9; EPRS 2018b, page 31]: spoken information can be transported to a recipient with auditory impairments by hearing aids increasing the loudness of the acoustic signal or by communication systems translating it into text or sign language. Humans with cognitive or learning disabilities may understand a text better if it is accompanied by appropriate visualisations [Altmanninger and Wöß 2008, page 380]. In the case of partially sighted recipients, it is often sufficient to optimise optical information according to the degree and properties of the residual vision, usually achieved by magnification, modifications or the inversion of brightness, and the application of a different colour set. Blind persons, by contrast, need one or more alternative information channels to compensate for the missing visual channel, such as acoustic or haptic representations.

2.2.1 Braille

One of the oldest and best-known ATs for blind recipients is *Braille*, a system to represent text characters as patterns of tangible dots [Lang et al. 2010, page 15; EPRS 2018b, pages 27–28; Kipke 2006, page 1]. The system was developed by Louis Braille in 1825 and still represents the main technique of reading by the sense of touch. Braille dots are formed by convex elevations on a flat carrier medium, traditionally a special type of paper which is thick and robust enough so that they persist during exposure to mechanical impact. The elevations have a base diameter of 1.0 to 1.7 mm and a height of 0.3 to 0.7 mm, varying among Braille output techniques and national standards [ISO 2013, Section 3.4].

The classic Braille system consists of cells of six dots, arranged in a matrix of three lines and two columns, resembling the upright six-spots pattern on a dice [Lang et al. 2010, pages 15–17; Kipke 2006, page 1]. Each of the six dots can have one of two states, *set* or *not set*, so one Braille character can show $2^6 = 64$ distinct combinations of dots. In order to express a larger character set, the *Computer Braille* system appends a forth row of two extra dots, increasing the character set to $2^8 = 256$ possible combinations. Braille is commonly read using the fingertips, since they show a high density of haptic receptors. The characters can be embossed by dedicated printers (see Subsection 3.1.1), on a special kind of typewriter, or manually using a Braille slate in combination with a corresponding stylus [Lang et al. 2010, pages 21–23].

2.2.2 Computer Input and Output

As a replacement for the visual screen of a computer, two modalities are widely used. The first is output via *speech synthesis*, also known as text-to-speech (TTS), which is nowadays readily available as software for standard platforms and compatible with common sound interfaces [Lang et al. 2010, page 190].

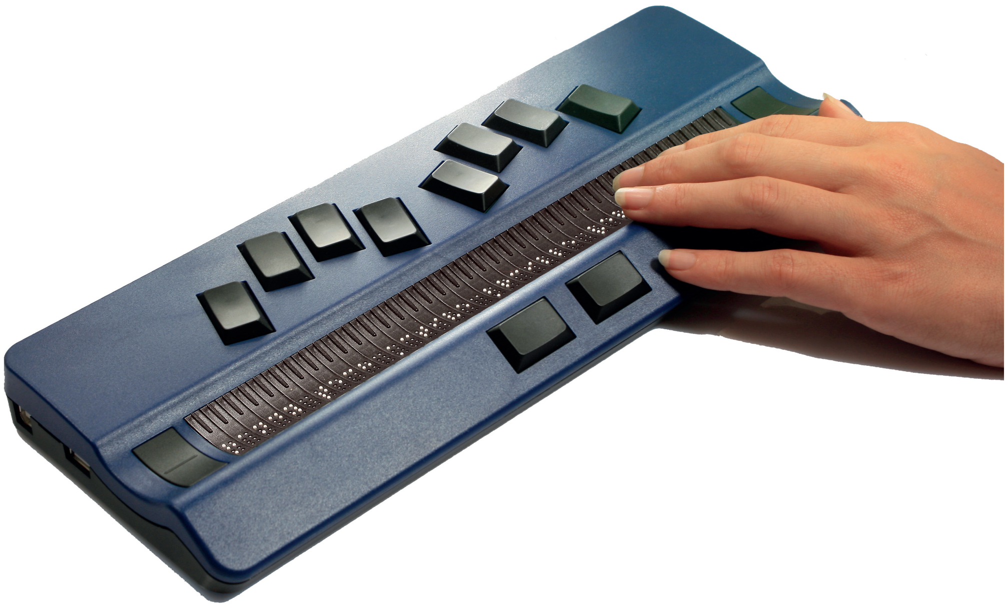


Figure 2.1: Active Braille by Help Tech [2021] is a portable Braille display with 40 cells. The surface of each module is concave to accommodate the shape of a reading finger. Each module incorporates an additional slim key (the `Cursor-Routing` key) above its Braille cell. Concave navigation keys are located to the left and right of the Braille line. Other keys are located above and below the cells. The device has been turned on, and some of its Braille dots are set. A left hand can be seen towards the right end of the line, reading the displayed Braille content. [Image provided by and used with kind permission of Help Tech [2021].]

The user can configure the speech output according to personal preferences with regard to speed, pitch, volume, and the pronunciation of certain characters and strings. More sophisticated TTS engines offer natural-sounding speech, where the user can choose from several voices and accents for most languages. Some current operating systems, such as Microsoft Windows 10 and Apple's macOS X and iOS are deployed with integrated speech synthesisers [Microsoft 2020a, Chapter 7; Apple 2020c; Apple 2020b].

The second solution is output on a *Braille display*, a device which is capable of presenting dynamically changing information as refreshable dot patterns [Lang et al. 2010, pages 189–190; Kipke 2006, pages 1–2]. In Braille displays, each character (cell) is produced dynamically by a module consisting of usually eight pins fixed in an upright position and arranged as a four-by-two matrix according to the Computer Braille system. The surface of each module is typically flat, but some products have modules with a concave surface to better accommodate the shape of a reading finger. Braille modules are aligned side-by-side to compose a line with a fixed number of characters. Additional keys are often provided to the left and right of the Braille line and above and below the Braille line. Typical portable Braille displays contain 40 cells, as shown in Figure 2.1; models designed for stationary use often have 80 cells. Some manufacturers also offer smaller or larger devices, as can be seen in Figures 2.2 and 2.3.

The reading surface of a module is formed by a cap the size of a Braille character with additional margins. Each cap contains eight openings located exactly at the positions of the underlying pins. In order to set or remove a particular dot, the corresponding pin is raised or lowered, respectively, so that it appears above or disappears beneath the level of the surface. The vertical movements are performed by actuators working on an electromagnetic or, more usually cases, a piezoelectric basis. An example of a Braille module is shown in Figure 2.4.

Many Braille displays can also act as an input device [Kipke 2006, pages 1–2]. Above each Braille cell, they typically provide a button known as the `Cursor-Routing` key. Pressing the button triggers an action on the character displayed on the corresponding Braille cell. Depending on the mode of operation

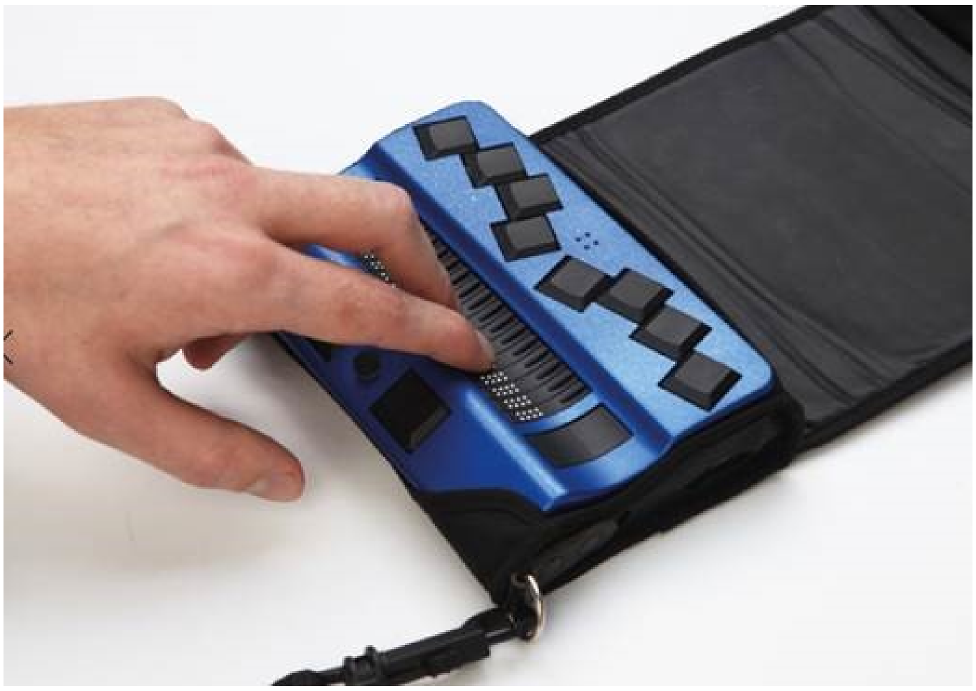


Figure 2.2: Actilino by Help Tech [2021] is a portable Braille display with 16 cells, deployed in a protective carrier bag. [Image provided by and used with kind permission of Help Tech [2021].]

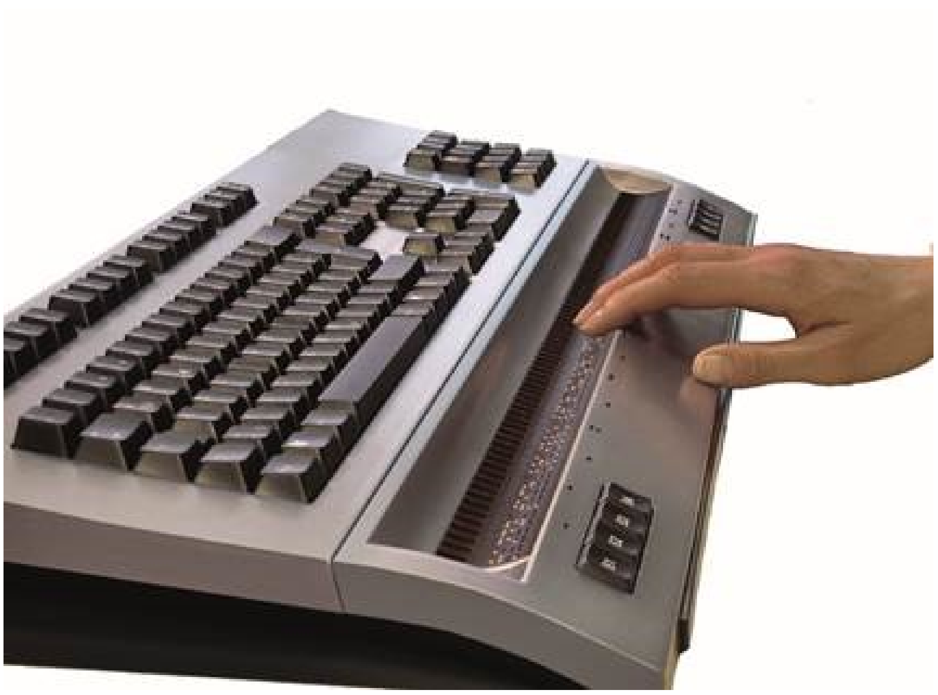


Figure 2.3: Modular Evolution 88 by Help Tech [2021] is a bulkier Braille display with 88 cells and an integrated keyboard. [Image provided by and used with kind permission of Help Tech [2021].]

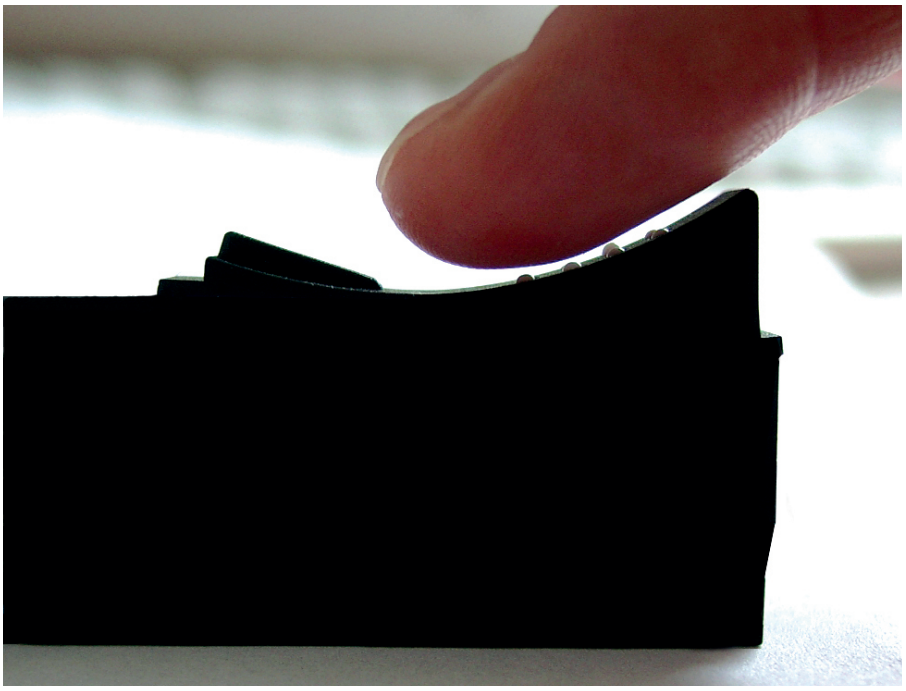


Figure 2.4: A single Braille module by Help Tech [2021], viewed from the left side. Its surface is concave in order to adapt to the shape of a reading finger. Six of its eight pins are raised. A finger is placed on the cell, reading the displayed Braille dots pattern. [Image provided by and used with kind permission of Help Tech [2021].]

and the user interface of the platform, this action might be a simulated mouse click on the location of the displayed character or the selection of the character within a text edit field. Moreover, most devices have additional keys to the left and right of and above and below the Braille line, whose presses can be passed to the computer. A comparably new feature is touch sensitivity of the Braille modules, enabling the detection of the reading position within the Braille line. The first example released on the market is the Active Tactile Control (ATC) technology developed by Kipke [2006] and applied by Help Tech [2021]. Modern Braille displays can be connected to personal computers and smartphones via standard interfaces like Universal Serial Bus (USB) and Bluetooth.

Apart from the additional buttons provided by Braille displays, the main input device for blind users is the standard computer keyboard, supporting touch-typing [Lang et al. 2010, pages 192, 194]. Some current AT supports selective speech and Braille output of the object the mouse cursor points to, but mouse navigation is of course particularly difficult for blind users. Recently, touch screen interaction has become increasingly popular in AT. These trends are discussed in Subsection 2.2.3.2.

Partially sighted individuals often use the standard computer screen in combination with features provided as part of the view settings of the operating system and/or special magnification software [Lang et al. 2010, page 191]. Pixellation artefacts occurring at high levels of magnification are eliminated automatically. For persons with a higher degree of visual impairment, magnification software often supports the combination of the described techniques with speech synthesis in the way that the text of the selected object is read aloud. In addition to magnification, some partially sighted users can benefit from aids such as increased contrast, the inversion of brightness, or alternative colour-coding.

2.2.3 Screen Readers

Both speech and Braille output exhibit two essential restrictions [Lang et al. 2010, page 192]. Firstly, they are only capable of transporting textual information, whereas the front end of many modern software systems is designed as a graphical user interface (GUI). Secondly, both modalities convey data at a significantly lower bandwidth than a visual screen.

Screen readers are a dedicated type of assistive software, which interacts with the user interface of the operating system and/or the current application to assist in navigation and to select which part of the screen to output. At the time of writing, the following screen readers are widely available and used:

- *JAWS*: a commercial product for Microsoft Windows, developed and marketed by Vispero [2020];
- *NVDA*: free open-source software for Microsoft Windows, developed by NV Access [2020];
- *Narrator*: proprietary software developed and integrated into Windows by Microsoft [2020a];
- *VoiceOver*: proprietary software developed and integrated into macOS X and iOS by Apple [2020a], Apple [2020b] and Apple [2020c];
- *TalkBack*: free software developed and integrated into Android by Google [2020a]; and
- *ChromeVox*: developed and integrated into Chrome OS by Google [2020b].

Throughout this thesis, most examples will be given for JAWS and NVDA, since these are the most widely used [WebAIM 2019].

2.2.3.1 Semantic GUI Transformation

The main purpose of screen readers is to analyse the GUI of the software currently running with regard to its component objects and to translate the results into a textual representation [Lang et al. 2010, pages 190–191, 193]. This analysis needs to consider several aspects of each GUI object:

- its *role* or *type*, that is, the manner of user interaction it performs; for example, *button*, *checkbox*, or *text input field*;
- its *name* or *title* as well as other possible text content;
- possible key combinations defined for access or activation; and
- its relationship to other objects within the hierarchical organisation of the GUI, that is, the application window it belongs to, its ancestor object, any descendant objects, and any text associated with it.

In short, the screen reader performs a semantic analysis of the internal GUI structure as a replacement for the intuitive analysis of the visual appearance performed by sighted users. In order to infer the semantics of GUI objects, several different approaches are pursued. The most recent strategy is the communication between screen readers and other software through dedicated application programming interfaces (APIs) provided by the operating system (see Subsection 2.4.3).

2.2.3.2 Input and Selective Output

Most screen readers provide mainstream GUI panels for configuration, which are fully accessible by both keyboard and mouse [Vispero 2020; NV Access 2020; Microsoft 2020a; Apple 2020c]. Nevertheless, the classic method of controlling a screen reader is using the keyboard, where the set of basic key commands integrated within the operating system is supplemented by a large number of additional key combinations [Lang et al. 2010, pages 190–191]. In order to make a clear distinction between the functions of the operating system and those of the AT, the strategy of reserving a particular modifier key has become established. For most screen reader actions, this key needs to be pressed along with one or more other keys. In NVDA, the user can choose either the `0` key on the numeric keypad for this purpose, where the `Num Lock` function needs to be disabled, or the `Caps Lock` key [NV Access 2020]. JAWS additionally offers the `Insert` key [Vispero 2020]. The original functionality of the respective key is achieved by pressing it twice consecutively within a time interval similar to that of a double-click with a mouse.

The key commands of the screen reader provide a powerful tool which allows users to choose from various navigation options, toggle between several modes of operation, and retrieve information on a multitude of different parameters, such as the time and date, the status of the power supply, the current window title, the content of the status bar within the current window, as well as the font, the colours, and other properties of the selected object. Moreover, screen readers are also capable of interacting with the current application, so that certain functions which are not designed to be keyboard accessible can nevertheless be reached via key commands of the AT.

Screen readers can be used in combination with any of the output modalities described in Subsection 2.2.2, where all of them can be combined flexibly and used in parallel according to the user's preferences [Lang et al. 2010, pages 190–191]. The decision which portion of the textual GUI representation is passed to the output systems depends on several parameters: by default, it is bound to the *keyboard focus* of the operating system, that is, the GUI object currently selected to receive keyboard events [Vispero 2020; NV Access 2020]. This method ensures that the user is aware of the current state of interaction.

However, many regions of a GUI window, such as static text and mouse-only toolbars, cannot receive keyboard focus and, thus, are outside the reach of this kind of navigation. Certain applications do not include any keyboard navigation at all. Therefore, screen readers provide additional output strategies detached from the keyboard focus. These features commonly include:

F1: Speaking text which has recently been added to the currently focused application window (called *screen echo* in JAWS) and temporarily showing it on the Braille display (called *flash messages* in JAWS);

F2: Speaking a certain piece of information on user request and showing it on the Braille display as a flash message;

F3: Binding the output to the object pointed to by the mouse cursor (called *mouse echo* in JAWS);

F4: Moving the mouse cursor by means of the `Cursor` keys and outputting information on the selected object (called the *JAWS cursor* in JAWS);

F5: Moving a mouse-like pointer independent of any cursor of the operating system by means of the `Cursor` keys and outputting information on the selected object (called the *invisible cursor* in JAWS);

F6: Navigating across the object hierarchy of the GUI independently of any cursor of the operating system by means of the `Cursor` keys (called the *touch cursor* in JAWS and *object navigation* in NVDA);

F7: Moving the displayed Braille frame by means of dedicated keys on the Braille display (called the *Braille cursor* in JAWS); and

F8: Displaying static text in a text field similar to a document in a text editor (called a *virtual cursor* in JAWS and *browse mode in NVDA*, see Subsection 2.2.3.3).

Features F4 to F8 above can be summarised as navigation methods which detach the reference point of the screen reader from the keyboard focus of the operating system. This reference point is sometimes referred to as *AT focus* or *accessibility focus* [Lang et al. 2010, page 191; Google 2020a]. After setting the accessibility focus to a particular object by one of the methods F4 to F8, the screen reader can perform certain actions on the selected object, for example, simulating left or right mouse clicks or performing drag-and-drop operations.

A relatively novel method of screen reader input and output is the integration of touch screens [EPRS 2018b, page 33]. In contrast to the usual behaviour without AT, touching the location of a particular GUI object does not trigger its default action. Instead, the accessibility focus of the screen reader is set to the respective object, causing its information to be read aloud and/or shown on the Braille display. This interaction paradigm enables the user to access any region of the displayed GUI and, in addition, to gain an idea of its visual arrangement. The default action of the selected object can then be launched by performing a double tap. Further gestures have been adapted to support choosing from additional options related to the object and opening an object's context menu. As an alternative to exploring the layout of the screen content as described above, a sequential shift of the accessibility focus across the GUI objects is possible by swiping with one finger to the left or right. This concept of non-visual touch screen interaction was pioneered by Apple with iOS and its integrated screen reader VoiceOver [Apple 2020a; Apple 2020b], giving blind users the opportunity to use mainstream smartphones and tablets for the first time. In the meantime, the basic concepts have been adopted by TalkBack for Android [Google 2020a] and NVDA and JAWS for Windows [NV Access 2020; Vispero 2020].

2.2.3.3 Handling Read-Only Documents

When entire documents are displayed as read-only text, that is, without any cursor for locating a particular character, for example, web pages or files in Portable Document Format (PDF), some screen readers can enable a special mode of operation [Vispero 2020; NV Access 2020]. In this mode, the screen reader buffers the content of the document in a multi-line read-only text field and presents it to the user as if it was displayed in a text editor. This way, the user can navigate the document line-by-line or character-by-character with the `Cursor` keys, where the newly selected line or character is output by the screen reader. Elements with a certain meaning, such as links, headings, tables, form controls, and graphics, are represented with additional text indicating the type of the element.

Moreover, the user is provided with a set of key commands for a quick navigation of the document structure; for example, to move to the next or previous heading, graphic, table, or link. Tables can be navigated by line and column, where the content of the selected cell and the header of the newly selected line or column are output by the screen reader. When moving the cursor to a certain element and pressing `Enter` or `Space`, the screen reader can attempt to simulate a left mouse click on this element. In JAWS, the described feature for viewing read-only documents is called *virtual cursor*, in NVDA it is called *browse mode*. It is activated by default in conjunction with most browsers and electronic mail clients (when opening messages in the folder for incoming or sent mail) as well as with Adobe Reader.

However, in order to make this navigation possible, the screen reader takes over almost the entire range of key presses, including those of the `Cursor` and the alphanumeric keys. In other words, all these keys no longer trigger events within the current application. While this has no consequence in static documents, it prevents the user from entering text into form fields, for example. Moreover, if a web application defines its own key commands (in many cases especially targeted to visually impaired users), these key commands do not have the intended effect while the screen reader is running. An example is the YouTube video player [Google 2020c], whose sliders for adjusting the volume and moving the playback cursor can be controlled not only by mouse but also by the `Cursor` keys.

For this reason, the screen reader can temporarily switch to a mode of operation in which most key presses are passed through to the application. In JAWS, there are two modes which do this: *forms mode* and *application mode*. Forms mode is activated when the cursor is moved to certain form elements, such as text input fields, combo boxes, and sliders. Application mode is enabled when encountering a document region specifically marked as an application (see Subsection 2.4.2). NVDA has a single mode called *focus mode*. In both screen readers, the user can manually toggle between the two modes and configure under what circumstances the screen reader should automatically enter the respective mode.

2.3 The Web Content Accessibility Guidelines

In 2018, the W3C [W3C 2021c] published Version 2.1 of the Web Content Accessibility Guidelines (WCAG) W3C [2018f]. The standard defines 13 guidelines for creating accessible web content, grouped into 4 broad principles. The standard does not give recommendations on implementation, but rather on the expected behaviour in conjunction with ATs and alternative input devices. Each guideline is supplemented by success criteria, which can be used for testing the compliance of a web page with the standard. Moreover, several supporting documents are provided, among others, a multitude of suggestions for applying current web technologies [W3C 2020].

2.3.1 Principle 1: Perceivable

- *Guideline 1.1 Text Alternatives*: Any graphical content which does not only have decorative purposes shall be accompanied by an alternative text.
- *Guideline 1.2 Time-Based Media*: Information presented through audio-only or video-only media shall also be available in an alternative modality.
- *Guideline 1.3 Adaptable*: It shall be possible to display the web content in different representations without any loss in information or structure.
- *Guideline 1.4 Distinguishable*: Web content shall be clearly distinguishable, concerning both the separation of foreground from background and distinguishing individual pieces of information from each other. This includes various aspects, such as minimum contrast ratios, a minimum difference in volume for foreground and background audio signals, and the possibility to resize text without needing ATs. Text shall not be presented as image content (with certain exceptions). Visual information shall not be conveyed by colour alone.

2.3.2 Principle 2: Operable

- *Guideline 2.1 Keyboard Accessible*: All functionality shall be accessible using the keyboard.
- *Guideline 2.2 Enough Time*: All users shall be provided sufficient time to interact with content, taking into account that reading text or entering input may take longer for humans with certain disabilities.
- *Guideline 2.3 Seizures and Physical Reactions*: Visual representations known to cause seizures or physical reactions for some humans must be avoided. This includes flashing at certain frequencies and the animation of GUI objects.
- *Guideline 2.4 Navigable*: The user shall have several options for navigation, be able to rapidly find the desired content, and always be aware of the current location. For instance, links for skipping a region of the page shall be provided, keyboard navigation shall proceed in a logical focus order, a web page shall have a meaningful title, and documents shall be structured with appropriate headings.
- *Guideline 2.5 Input Modalities*: Web content shall provide alternative input modalities beyond keyboard interaction in a user-friendly way and shall not restrict the input methods of the local system.

2.3.3 Principle 3: Understandable

- *Guideline 3.1 Readable*: Users shall be provided utilities to read and understand text content, for instance, machine-readable indications of the language in which the text is written as well as optionally available explanations for abbreviations and certain terms. If the text requires more advanced reading ability, an alternative version with a lower reading level shall be provided.
- *Guideline 3.2 Predictable*: Interactive web content shall behave in a logical, consistent manner, and user input shall cause no other effects than those which can be expected by the user.
- *Guideline 3.3 Input Assistance*: Users shall be provided guidance in correctly handling an interface. Techniques for detecting input errors shall be applied, mistakes and errors shall be reported to the user as text, and the user shall have a means of correcting input errors. Unintended actions shall be reversible, and critical actions only processed after confirmation, especially for contracts, financial transactions, and the manipulation of persistent user data.

2.3.4 Principle 4: Robust

- *Guideline 4.1 Compatible*: Web content shall be compatible with all current and future technologies, including browsers and ATs. Markup shall be well-formed and used in accordance with the respective specifications. Roles and names of all objects shall be machine-readable, states and values intended to be set by the user shall in addition be writable by browsers as well as AT systems. Changes to any value or state shall be reported to both browsers and ATs.

2.3.5 Further Development

At the time of writing, version 3.0 of WCAG is being worked on, and a first draft has been published [W3C 2021a]. This successor can still be abbreviated as WCAG; however, the acronym here stands for W3C Accessibility Guidelines. The change of the title reflects that this document no longer solely concentrates on web pages, but also considers many other kinds of digital content, such as ePub and PDF documents, as well as standalone desktop and smartphone applications, including browsers, ATs, content management systems, authoring, and testing tools. Based on several years of research, WCAG 3.0 will address a broader range of individual functional needs among different users with and without disabilities.

While the requirements to conform with WCAG 3.0 are similar to those for the compliance with earlier versions, they have been restructured with the intention to be easier to understand and more flexible to adapt to rapid changes in technology. In the new document, guidelines now represent the top-level structure. Each guideline contains one or more outcomes, corresponding to the success criteria in previous versions. Each outcome contains a description, examples of critical errors, and a rating scale for conformance between 0 and 4. Incidents classified as a critical error immediately yield the lowest rating. For example, the new Guideline 7.4 “Structured Content” contains three outcomes: “Headings organize content”, “Uses visually distinct headings”, and “Conveys hierarchy with semantic structure”. Due to the fundamental changes in structure and scoring, WCAG 3.0 will not be backward compatible with previous versions. For this reason, WCAG 3.0 will not deprecate earlier versions of WCAG.

2.4 Techniques for Accessible Web Pages

Some web sites offer a separate version created especially with screen reader users in mind, for example, Hörzu [Hörzu 2020]. However, in most cases, a web page does not need to be designed in a non-graphical or less visually appealing way in order to achieve accessibility. Similar to the paradigm of responsive web design, which promotes the creation of one flexibly adaptable web site rather than several versions targeted to different browsers and displays, the recommended manner of producing accessible web content is applying the paradigm of *inclusive design*, as explained, for instance, by Pickering [2016]. First and foremost, this means that a web page is implemented as a well-formed document, employing the various web development technologies in accordance with the specifications of the W3C. A particularly important aspect is to define a web document primarily by its semantics, rather than its mere visual layout.

2.4.1 Semantic HTML

The Hypertext Markup Language (HTML) [W3C 2017c] defines various elements corresponding to particular types of text structure or user interaction. Examples include the `<p>` element for a paragraph of static text, the `<button>` element to create a clickable button object, and the `<input>` element for several types of user input controls, such as checkboxes and fields for entering text. Since these elements not only specify the visual appearance, but also the meaning and intended behaviour, they can be summarised under the term *semantic HTML* and represent the recommended means for designing accessible web pages [Pickering 2016, pages 34, 78–88]. Since screen readers try to determine the semantics of GUI objects (see Subsections 2.2.3.1 and 2.4.3), merely using this subset of HTML elements in the intended manner contributes significantly to the accessibility of a web page.

A well-known example is that of creating a button for a web application [Pickering 2016, pages 16–19]. it is technically possible to achieve the intended visual appearance and functionality using the semantically neutral `<div>` container element, attaching a JavaScript handler for click events, and embedding an image of the desired icon. However, if no additional measures are taken, this kind of implementation provides poor accessibility. Firstly, the element is not displayed as expected, if loading of graphics has been deactivated in the browser. Secondly, the button cannot be focused and triggered using the keyboard. Finally, screen readers are not informed about the meaning of the element. This has the consequence that the corresponding text label, if available, is interpreted and output by the screen reader as if it was a mere piece of text, so that a blind user is not told about the button functionality. In case that no textual label is provided, the embedded image might be interpreted as decorative, causing the button to be completely ignored by the screen reader. Even if the user infers the meaning from the prompt of a label and selects the button by means of the screen reader's navigation facilities, pressing `Enter` or `Space` might have no effect, since the AT is not supplied with definitive information about how to interact with the element or where exactly to place a simulated mouse click. Using the `<button>` element, by contrast, creates an object which is appropriately displayed by the browser even if graphics are not loaded, can receive keyboard focus, dispatches click events upon presses of the `Enter` or the `Space` key, and is well understood by ATs.

The scenario just described applies analogously for any other object intended to interact with the user. A highly problematic issue is the non-standard implementation of a checkbox or a radio button, since in both cases, the associated text is not included as a descendant element or as an attribute, but rather as a separate element placed next to it. Therefore, a checkbox or radio button created by means of the `<div>` element will contain only graphical information by default, meaning that the interactive object itself might not be considered by screen readers. As a consequence, the information about its state, that is, whether it has been checked or not, is not reported to a blind user. Moreover, if the user navigates to the associated text, pressing `Enter` or `Space` might cause the screen reader to perform a mouse click on the label but not on the checkbox or radio button itself, so that its state is not changed.

Another example concerns the formatting of text fragments as headings [Pickering 2016, page 79]. HTML specifies the elements `<h1>` to `<h6>` for the purpose of declaring headings at six distinct hierarchical levels. Applying this technique ensures that browsers display the respective text at appropriate font sizes, depending on the level, the size of the screen, and the visual preferences configured by the user. Screen readers present the text fragment as a heading of the specified level, for instance: "*Heading level 2, Introduction*". In addition, screen reader users can benefit from the feature of navigating directly to the next or previous heading, to quickly obtain an overview of a web document or to find a particular section.

If, by contrast, the respective text is only visually highlighted, for instance using Cascading Style Sheets (CSS), all the advantages just mentioned are not available. Conversely, the use of heading elements as a means of visual formatting for text fragments which are not intended to act as headings leads to screen readers erroneously announcing the piece of text as a heading, and the facility of navigating by headings might set the accessibility focus of the AT to unintended locations.

Although most screen readers offer a mode for browsing web pages by keyboard, independent of the operating system's navigation, it is recommended to ensure that every user input control can receive keyboard focus by means of the `Tab` key [Pickering 2016, pages 61–63]. All HTML elements explicitly designed for user interaction, such as `<a>`, `<input>`, `<button>`, `<select>`, and `<details>`, do so by default. If a different element is used, it can be declared focusable by attaching the `tabindex` attribute to it. The attribute's value is an integer ≥ -1 and determines if and in which order the elements are focused when pressing the `Tab` key (known as *tab order*). A positive number indicates elements which should be focused first, in that order. Elements with a `tabindex` value of `0` are focused afterwards in the order they appear in the source code. Elements with a `tabindex` value of `-1` are not included within the tab order, and can only be focused programmatically.

Furthermore, every element should provide an *accessible name*, that is, an associated title which can be read by ATs [Pickering 2016, page 39]. In the case of a cancel button, for example, this would typically be the string "Cancel". When text is embedded within an element, it is taken to be the element's name in most cases. If no associated text content is present, the accessible name needs to be explicitly defined by the author of the web content, for instance using ARIA properties, as described in Subsection 2.4.2.

In addition, every element can also provide an optional *accessible description* containing more detailed information. The accessible description, too, can either be explicitly defined or is derived from text content associated with the element. The computation of accessible names and descriptions is explained in more detail in Subsection 2.4.3.

The arrangement of elements in the HTML source code plays an important role too [Pickering 2016, pages 131–132]. While CSS provides the possibility to place every object at an arbitrary location in the visual interface, screen readers derive the order of objects for textual presentation from that of the corresponding node in the browser's document object model (DOM), which, in the case of markup, is determined by the order of the elements in the source code. In other words, the intended reading order of the document should be reflected in the HTML and the resulting DOM structure. If, for instance, a user action causes new content to be added to the page and the corresponding new elements are appended to the end of the DOM structure, a blind user often does not notice the change, since it appears at the very bottom of the textual representation and, thus, at an unexpected location. For this reason, a better option in terms of accessibility for dynamically adding content is to insert the new content next to the node of the currently focused element.

2.4.2 The WAI-ARIA System

The Web Accessibility Initiative (WAI) of the W3C [W3C 2021b] produced a suite of recommendations and associated guides intended to improve the accessibility of web documents, known as Accessible Rich Internet Applications (WAI-ARIA or just ARIA). The core specification [W3C 2017a] defines a taxonomy of attributes which can be applied to elements of markup languages such as HTML, in order to add semantic information useful for ATs. The role attribute specifies the type and function of the element. Possible values for the role attribute include button, checkbox, radiobutton, and heading. The idea is that the type and function of an element with a valid *role* can be recognised by ATs, even if it is not implemented using a standard HTML element. For instance, a non-standard button such as described in Subsection 2.4.1 can be identified and treated as such by ATs if it is assigned the attribute role=button. Other examples of valid ARIA roles include:

- roles for structuring a web document into distinct sections, such as group, region, article, and main;
- textbox for text input fields;
- table, row, rowheader, columnheader, and cell for tables and their components;
- list and listitem for unordered lists and their contained list items;
- img for graphics; and
- presentation or none for content with no semantic significance, such as decorative graphical elements, causing the respective element to be ignored by screen readers.

It should be noted that some ARIA roles do not have a direct counterpart in standard HTML, for example menu and menuitem to denote a pop-up menu and its entries. The role application is intended to identify the user interface of a web application. This role has particular relevance, since it determines if screen readers switch to application mode (focus mode in NVDA) when encountering an associated interactive element (see Subsection 2.2.3.3).

Beyond the definition of roles, ARIA specifies *state* and *property* attributes, which are all denoted by the prefix aria- in the attribute name. Both states and properties specify further information on an element, where properties express persistent characteristics and states can be expected to change frequently. An example of states is the boolean attribute checked, indicating whether a checkbox or radio button is currently checked (checked=true) or not (checked=false). It should be emphasised that it is the author's responsibility to implement the functions to dynamically set the values of states, since neither browsers nor ATs can determine them for non-standard GUI objects.

Important examples of properties include aria-label and aria-labelledby, both used to specify the accessible name of an element. aria-label accepts an unstructured one-line string as value, directly specifying the verbatim accessible name of the element. By contrast, aria-labelledby can be used to reference one or more elements by their ids, where multiple ids are concatenated and delimited by a space character. This way, the accessible name of the referencing element is composed from the accessible names of the referenced elements. Both properties may be combined such that aria-labelledby points to an element whose accessible name is given by aria-label, causing the latter to be the accessible name of the former element as well. Moreover, it is permitted that an element points to itself with the aria-labelledby property, which can be used to concatenate the text of this element with that of others. In the same manner, the property aria-describedby is defined to specify an accessible description for an element. Other examples of ARIA properties include:

- aria-roledescription intended to provide additional information about the function of an element, in case it cannot be comprehensively described by a valid ARIA role;
- aria-flowto to specify a navigation order within a document different from that of the DOM nodes;
- aria-valuemin, aria-valuemax, aria-valuenow, and aria-valuetext to indicate the minimum, the maximum, and the current value of an adjustable GUI control, such as a slider or a spin button; and
- aria-keyshortcuts to specify key combinations for accessing and/or activating an element. This property only causes the user to be informed about the specified key commands, it does not implement any handling of key presses.

ARIA attributes should only be used in cases where the host language element lacks the appropriate information. An example is the non-standard implementation of a checkbox or radio button without any implicit semantics, as discussed in Subsection 2.4.1. If the `<div>` element used in this example is assigned the ARIA role checkbox or radiobutton, the element can be correctly recognised by ATs, even if it is not associated with any text label. Setting the state aria-checked for this element according to the user's actions allows to ATs detect if the object is currently checked or not. However, in order to achieve full accessibility, the element should also be given an accessible name and assigned the tabindex attribute so that it can receive keyboard focus.

2.4.3 The Accessibility Tree

Screen readers and similar ATs aim to transform GUIs into a textual representation for visually impaired users, To achieve this, they try to obtain information about the semantics of the objects within a GUI, as described in Subsection 2.2.3.1. One way of receiving such information is a so-called *Accessibility API*.

Watson [2015] discusses the history of Accessibility APIs. Microsoft Active Accessibility (MSAA) was released in 1997 for Windows 95 and was the first platform-wide accessibility API. Similar APIs followed for Linux (AT-SPI), macOS (NSAccessibilityProtocol), and later still for iOS (UI Accessibility) and Android (Accessibility Framework). Today, all major operating systems support at least one integrated Accessibility API.

Web browsers support one or more of the Accessibility APIs for the particular platform they run on, and expose information about both the browser itself and the currently rendered web page. To do this, the browser constructs a so-called *accessibility tree* [Pickering 2016, page 84; W3C 2017b], a hierarchical structure expressing accessibility information about the structure of the current web page, built from the document object model (DOM) tree and applied ARIA roles and properties. The accessibility tree is then passed to the screen reader.

The Core Accessibility API Mappings [W3C 2017b] is a W3C recommendation and is part of the WAI-ARIA suite. It specifies how browsers should map the semantic information of web content provided implicitly by native web content language elements or explicitly by the ARIA attributes defined in [W3C 2017a] to a corresponding representation for accessibility APIs. In general, the document states that:

- the accessibility tree must include all elements of the web document with semantic significance and exclude all elements which are hidden or important only for the graphical appearance (with certain exceptions);

- each node of the accessibility tree must contain the semantic information of the corresponding element, expressed by the appropriate keywords of the platform-specific accessibility API;
- in case of any conflicts between the native semantics of a web content language element and the semantics of an ARIA role assigned to this element, web browsers should prefer the latter;
- in case of any conflicts between a native state or property of a web content language element and an ARIA state or property assigned to this element, web browsers should prefer the former;
- if an AT requests a default action on an element, web browsers should simulate a mouse click on this element; and
- web browsers should notify an AT by means of dedicated events in case of any changes to the DOM, accessibility tree, or focus with semantic significance and in case of user input.

The Core Accessibility API Mappings document defines appropriate mappings for each of the various platform accessibility APIs. Furthermore, the recommendation describes how keyboard focus should be handled by web browsers in order to enable control of a web application by keyboard and to receive appropriate feedback for each keyboard event.

A further W3C recommendation, the Accessible Name and Description Computation recommendation [W3C 2018a] specifies a recursive algorithm for deriving the accessible name and description of a given DOM element. Both values are ultimately represented by plain character strings without markup, line breaks, tabs, or sequences of multiple space characters. The algorithm to determine an accessible name considers source(s) within the DOM tree in the following order:

1. an aria-labelledby property with at least one valid id reference, where for each referenced element its accessible name is computed and all accessible names are concatenated, delimited by a space character;
2. the value of an aria-label attribute;
3. the alternative text defined by a native attribute or element of the web content language, such as title, alt, or <label> in HTML;
4. the accessible names of all descendant nodes, that is, descendant elements and text fragments, and possible text content specified by means of CSS, all concatenated into one string;
5. the value of a tooltip attribute.

With regard to this order, the following exceptions apply:

- If an element is marked as hidden and is not directly referenced by aria-labelledby, aria-describedby, or a similar attribute or element of the web content language, the accessible name or description is an empty string.
- If an element is already involved in a computation by an aria-labelledby or aria-describedby reference, its name or description computation starts with step 2. In other words, indirect references by series of multiple aria-labelledby or aria-describedby pointers are not considered. The purpose of this restriction is that an element can reference itself by one of the named properties.
- If an element has an alternative text specified by a native attribute or element of the web content language, but is marked as presentational, that is, role=presentation or role=none, no accessible name or description is computed.
- If the element is a GUI control with an adjustable value, such as a text input field or a drop-down

list, and is embedded within the label of another GUI control, the adjusted value is considered.

The computation of an accessible description works analogously with the only difference that for step 1 the property aria-describedby is used instead of aria-labelledby.

Chapter 3

Chart Accessibility

The ambitious task of conveying charts and graphics to blind recipients has been addressed in numerous ways. The most common approach is to describe graphics by words, either orally or in textual form [Gardner and Bulatov 2006, page 1243]. For this reason, the Hypertext Markup Language (HTML) standard specifies the `alt` attribute which can be attached to the `<img>` element in order to assign it a so-called *alternative text*, that is, a concise description [W3C 2017c, Section 4.7.5.1]. In addition, the `longdesc` attribute was introduced which can be used to state the uniform resource locator (URL) of a more detailed description with possible markup [W3C 2015a]. The Web Content Accessibility Guidelines 2.1 (WCAG) recommend that alternative texts be provided to all non-text content within a web document [W3C 2018f, Guideline 1.1].

However, providing textual descriptions as an alternative for entire graphics has several difficulties. Firstly, they traditionally need to be produced by humans, meaning that whether they are available for a particular graphic relies on the willingness of at least one person to write and include an alternative text and/or a long description. Particularly for more sophisticated graphics like those in the domain of science, technology, engineering, and mathematics (STEM), this work can consume a considerable amount of time; for any dynamically created graphics, it is impossible.

Secondly, writing a comprehensive, precise, and concise description of a STEM graphic so that it is understandable for any potential target user is non-trivial. To address this problem, guidelines for describing visual content have been published by various institutions. A well-known example is the image description guidelines by NCAM and DIAGRAM Center [2015]. Based on these guidelines, which also consider various types of STEM graphics, the DIAGRAM Center [2017] published a web application to practice the creation of alternative text, and Morash et al. [2015] propose a system of conditional questions and templates as a utility for authors to create high-quality descriptions of charts.

A different approach towards this problem is the automated generation of graphics descriptions, which has meanwhile been implemented in several systems and is in some cases a result of research in the field of artificial intelligence (AI). A popular example is Seeing AI, an iOS application developed by Microsoft and available in the App Store for free [Microsoft 2021]. Among other functions, the application offers a mode for recognising and describing human faces including their expressions. Another mode describes *scenes*, that is, every-day pictures taken with the camera or stored on the device, detecting its main objects. Both modes produce the results within a few seconds using a cloud-based service. However, at the time of writing, the scene mode is still marked as experimental and can only handle simple pictures. SIGHT [Carberry et al. 2012; Moraes et al. 2014] is a system which generates textual descriptions of simple line and bar charts and is presented in Section 3.4. evoGraphs, [Sharif 2015a], an academic solution designed to automatically describe charts with regard to certain statistical properties, is presented in Subsection 3.5.6.

Nevertheless, the question as to whether verbal descriptions can provide an appropriate alternative to data visualisations is controversial. Altmanninger and Wöß [2008, page 378] point out that although textual descriptions are helpful for blind recipients, they might not accommodate the needs of users with other kinds of disabilities. Gardner and Bulatov [2006, page 1243] argue that charts and diagrams would not be used so commonly in the sciences if text could transport the same information with an acceptable number of words. Even if a description is assumed to include all the data contained in a visualisation, another question is whether the information can be consumed with a comparable degree of efficiency.

Shneiderman's Visual Information-Seeking Mantra [Shneiderman 1996, page 337] describes the basic strategy of viewing graphical data representations as a three-step process: "Overview first, zoom and filter, then details-on-demand". Based on this principle, Shneiderman [1996, page 337] defines a taxonomy of seven possible user tasks:

- *Overview*: Gain an overview of the entire collection.
- *Zoom*: Zoom in on items of interest
- *Filter*: filter out uninteresting items.
- *Details-on-demand*: Select an item or group and get details when needed.
- *Relate*: View relationships among items.
- *History*: Keep a history of actions to support undo, replay, and progressive refinement.
- *Extract*: Allow extraction of sub-collections and of the query parameters.

This taxonomy, however, implies the availability of non-sequential information access, which the visual perception system of fully-sighted users provides. Continuous text, by contrast, can be regarded as a linear structure intended for sequential consumption. This applies particularly in the case of blind users, who are not capable of scanning over several lines of text at a time and, thus, of searching for a particular piece of information within the text as quickly as a fully-sighted recipient. Moreover, extracting important characteristics of the data, such as minima and maxima, trends, patterns, irregularities, and outliers, requires the recipient of a text to memorise and interpret the data accordingly, whereas these characteristics are often readily apparent for a sighted person looking at a chart or visualisation [Zou and Treviranus 2015, page 108]. A solution could be to include such characteristics in the description, but this has the consequence that the decision on which features to pay attention to is shifted from the recipient to the author of the text or the designer of the text generation algorithm.

Another common strategy is to arrange the data in the form of a standard table, where each non-header row or column corresponds to a data series and each cell contains the value of a data point. This representation has several advantages in terms of data exploration compared to text descriptions. For example, it can be generated automatically with relatively low effort, and it ensures that the recipient has access to the values of all data points. Most current screen readers offer special key combinations for moving between the columns and rows of a table (usually involving the `Cursor` keys), where the header of the newly selected column or row, and the content of the selected cell are spoken. With tables, blind users can navigate to particular data points more easily than continuous text. However, this approach does not overcome the problems of lack of overview and difficult interpretation [Zou and Treviranus 2015, page 108].

For the reasons discussed above, various solutions have been proposed which strive to provide recipients with impairments the same level of access to data as it is possible for ones without any disability. In this chapter, several of these proposals will be introduced. The approaches can be categorised with regard to multiple aspects. Firstly, the solutions differ in the input format they handle: some systems are only laboratory prototypes working with dedicated test material, whereas others can be regarded as complete solutions which might require a certain category or format of graphics but accept arbitrary examples of this type. The latter can be subdivided into closed systems, which perform all the transformations from an input image to the output internally, and open systems, for which an input or intermediate file format is specified, so that they could interact with third-party graphics software. Moreover, the solutions vary in their implementation, ranging from Scalable Vector Graphics (SVG), other Extensible Markup Language

(XML) transformations, through image recognition, to AI algorithms. Some of the systems require special hardware, whereas others can be executed on common personal computers or smartphones.

In this chapter, the proposals are classified by the output modalities they use, that is, the interface through which they present the information to the user. Section 3.1 presents tactile systems. These solutions produce an output which can be perceived by the sense of touch, including static graphics embossed on paper as well as refreshable representations composed of moving pins. Section 3.2 covers solutions producing acoustic output and Section 3.3 looks at solutions which combine multiple output modalities. Software solutions which interact with a running screen reader is presented in Section 3.4. In essence, this means that the software generates (additional, potentially non-displayed) textual output in response to a user action, and exposes these strings to the ATs, such that the latter can read them aloud and/or output them to a connected Braille display. A special case of such screen-reader-friendly software used in web applications are charting libraries with accessible output, which will be described in Section 3.5.

Sometimes, however, the distinctions are blurred. For example, the iGraph-Lite system is described in Subsection 3.2.1, because its main exploration component provides speech output independently of any screen reader; nevertheless, its basic summary output is intended to be consumed by means of AT. This chapter gives a representative overview of some of the vast variety of existing approaches and is certainly not intended to be exhaustive or complete. For further reading, see the reviews in [Braier et al. 2014; Carberry et al. 2012; Choi and B. N. Walker 2010; Fredj and Duce 2006; Gardner and Bulatov 2006; Revnitski 2005; Yoshida et al. 2011; Zhao et al. 2008]. The chapter concludes with a separate section dedicated to Describler, a web application which inspired the development of the AChart software in the practical part of this thesis.

Some of the solutions presented in the following sections provide descriptive alternative texts as a means to provide the recipient a first idea of a graphic's content. Schepers [2020] calls alternative texts "a great starting point", not only for blind users but for all users, as it can help users decide whether to spend more time with a particular graphic or data visualisation. Throughout the chapter, the term *description* will be used for a text which conveys detailed information about the dataset, such as individual data points, statistics, and trends. By contrast, the term *summary* will be used for a text containing only information about the chart type, the axes, or the number of data series and data points.

3.1 Tactile Output

One of the oldest and most intuitive strategies to adapt a graphics document for blind recipients is to transform it into a haptic representation. Braier et al. [2014, page 2] found that the cognitive processing of tactile sensations is related to that of visual ones. A traditional technique to create tactile graphics is the handcrafted composition of a three-dimensional (3D) document or model [Lang et al. 2010, pages 103–116]. For example, haptic charts, maps, and diagrams can be composed by fixing strings, pieces of thick paper, adhesive tapes, and other shapes of well-tangible material on a sheet of cardboard. For the duplication of such models, the thermoform method can be applied: the master copy is placed into a compartment, and a special type of plastic sheet is laid above it. The plastic is then heated to its melting point, and a vacuum is produced within the compartment, so that the plastic sheet follows the form of the master. Devices which can be used for this technique are, for example, the Thermoform Graphics Machines by American Thermoform [2019].

The same company also produces the Swell Form Machine and a corresponding type of paper known as *swell paper* or *capsule paper*. When heated, the surface of this paper rises at all non-blank locations, such that the height of the respective portion of paper corresponds to the darkness of shade at that location in the original source. In this way, hand-drawn, photocopied, or printed graphics from arbitrary sources can be reproduced on swell paper, and thus made tactile. A similar product is the Pictures in a Flash (PIAF) Tactile Image Maker by Harpo [2021].

For several decades, the analogue techniques described above have been widely used to produce tactile graphics by schools and libraries for the blind [Lang et al. 2010, pages 103–116]. However, manual preparation and duplication is time-consuming and requires a certain amount of expertise and experience. For many potential recipients, tactile images are difficult to understand, especially for those who lost their sight early and, therefore, could not fully develop their visual imagination. Users often require training and practice to correctly and efficiently handle and interpret such representations. Haptic resolution is orders of magnitude lower than visual resolution and varies significantly among individual recipients [Klatzky and Lederman 2004, page 152; Gardner and Bulatov 2006, page 1243]. Simultaneously, the portion of a graphic perceivable by the sense of touch is relatively small, resulting in rather sequential exploration, whereas a fully sighted user can process several pieces of visual information in parallel [Prescher et al. 2017, page 392].

As a consequence, in most cases, graphics produced for visual perception cannot immediately be mapped onto an analogous tactile counterpart, but several steps of adaptation are necessary, including simplification, reduction of information to the essential parts, omitting solely decorative elements, and the magnification of objects too small for tactile recognition. Moreover, if information is also encoded as colours or shades of grey, this representation needs to be transformed to an equivalent tangible system. Another important issue is the placement of text labels, since Braille characters have a fixed size (see below) and often require more space than optical fonts [Gardner 2016, page 419]. Goncu and K. G. Marriott [2008] argue that converting visual charts to tactile copies does not accommodate the needs of blind recipients in many cases, because certain changes in the layout are necessary too. These include inserting sufficient inner space for the comparably large Braille labels, placing the horizontal axis at the top rather than the bottom of the chart, and adding contextual information to some elements, such as labels with numeric values to single bars or pie slices.

With the advent of information technology, various solutions have been developed to immediately produce haptic representations of graphics from digital ones. The most common strategy is to compose raster images out of tactile pixels in a similar manner it is done in the visual domain. Tactile pixels are usually formed by convex elevations, often referred to as *dots*, arranged on a plane carrier medium. Since Braille characters consist of such dots (see Subsection 2.2.1), most devices for producing tangible raster graphics benefit from techniques related to the Braille system. However, Braille dots do not fully meet the requirements of graphics, largely due to a maximum resolution of around 12 DPI and the spacing between neighbouring characters.

3.1.1 Static Output

One of the earliest solutions for the tactile embossing of data visualisations was developed at Vitro Laboratories and is presented by Wefold [1976]. It is based on a generalised system to author graphics documents for printing independent of the output medium, such as the printer type and resolution. For this purpose, graphics are defined as a series of macro calls. When printing a document on a particular hardware configuration, the appropriate set of macros is applied. If a graphic is to be embossed, a special macro set is chosen which constructs lines as a series of dots (that is, full stop characters) and translates letters to their corresponding braille dot combinations. The printout is performed on a line printer, where an elastic strap is placed behind the paper so that the physical impact by the printing mechanism causes the paper to be raised at the positions where the full stop characters are written. The system was used regularly by a blind engineer working at Vitro Laboratories for embossing various types of charts and diagrams.

Modern *tactile embossers* are devices which stamp depressions in the form of negative dots into a sheet of relatively thick paper, so that tangible dots arise at the corresponding positions on its opposite side. Tactile embossers were originally developed for printing Braille. However, many systems currently available on the market provide a graphics mode offering higher resolutions and an equidistant pattern of dots. Examples include the Phoenix, Romeo 60, and Juliet 120 by Enabling Technologies [2017] which support a resolution of 25 DPI [Enabling Technologies 2015, page 13–14], as well as products by Index Braille [2020] with resolutions from 20 DPI (Basic-D V5) to 50 DPI (Braille Box V5). The embossers by

ViewPlus [2021] and Irie-AT [2019] achieve similar resolutions and, in addition, are capable of setting the dots to eight different amplitudes: for instance, VP Elite and Premier [ViewPlus 2016a], VP EmBraille [ViewPlus 2016b], IRIE Braille Buddy, and IRIE BrailleSheet 120 w/Power-Dot Braille. The variation in dot height is used to encode visual information other than two-dimensional shape and position, such as brightness or colour. By default, the different amplitudes represent the shades within a greyscale version of the image, where level 0 (a blank dot cell) corresponds to white and the maximum level 7 to black.

Common graphics editing applications can be used to prepare an image for printing on an embosser, but there are also dedicated software solutions for this purpose. QuickTac is an application for drawing tactile graphics offered by Duxbury [2020] free of charge. The software provides tools similar to that of a simple paint programme and a preview of the resulting dots pattern. It supports direct printing on a tactile embosser as well as saving the created image in a special file format. Goncu and K. G. Marriott [2008] present a software application developed in collaboration with Vision Australia for the automated creation of bar and pie charts in SVG format optimised for tactile representation. The tool accepts tabular data stored in formatted text files as input. Braille labels can be printed that the labels and graphical content do not overlap. Afterwards, a sighted person can fine-tune several properties of the chart, such as margins and the spacing between elements. At the time of writing, no usability study had been conducted. However, the authors report that first feedback provided by sighted chart transcribers and (mostly blind) proofreaders at Vision Australia was highly positive. In particular, they note that the average time necessary for preparing tactile charts decreased from approximately half an hour in the case of manual creation to around 5 seconds when using the software.

Ferro and Pawluk [2013] developed algorithms which automatically perform segmentation and simplification of an existing image to make it suitable for tactile output. Implementations of such algorithms are offered by some of the manufacturers of tactile embossers, in combination with dedicated tools for drawing tactile graphics. For instance, the Tiger Software Suite by ViewPlus can be used in conjunction with the their embossers to prepare documents for tactile output. It includes a Microsoft Office add-in which provides a function to print charts from Excel [ViewPlus 2020, page 51] as well as Tiger Designer for creating and adapting graphics.

Tiger Designer supports the import of images in Portable Document Format (PDF) and their conversion to a format suitable for tactile embossing. The image is transformed into an eight-level greyscale representation with the resolution scheme supported by the target ViewPlus embosser model, applying anti-aliasing techniques [ViewPlus 2020, pages 58–78]. Text characters within the original document are encoded as Braille letters. The result is shown in a preview of the embossed dots pattern and can be modified with tools similar to those offered by a paint application. In order to represent particular colours or other characteristics, special dot patterns can be defined and then applied for filling certain shapes. Braille text labels can be added, the density and the contrast can be adjusted, and an automatic simplification function can be applied. The prepared image can then be saved in a dedicated file format or printed on a ViewPlus embosser.

A similar application is the Firebird Graphics Editor by Enabling Technologies [2017]. This software can be used to create graphics for tactile output, either by drawing or by editing existing images in several file formats. For this purpose, the software provides various automatic tools and filters to clean and simplify images and to preview the tactile output. Irie-AT [2019] offers the TactileView design software, which comprises various drawing tools and functions for the guided composition of maps and diagrams as well as for creating charts from mathematical functions. A number of image file types can be opened, and different filters are available to convert the imported image for tactile output. The software supports Braille embossers by Irie-AT, Index Braille, and ViewPlus.

A relatively novel alternative for creating static tactile graphics is the use of printers for 3D output. Braier et al. [2014], for instance, introduce the term *relief charts*, which means not only a flat haptic representation of two-dimensional chart elements but also the encoding of information in the amplitude of the relief shapes. The strategy is applied to five chart types in different manners. For scatter plots, pie charts, simple bar charts, and stacked bar charts, height is used for conveying the value, as a second, redundant means. In the case of star charts, two approaches are presented and were evaluated. Firstly the one just described and secondly using a distinct fixed height for each n-gon in order to ensure clear separation of the data points. In a qualitative study with 17 visually impaired participants, it was found that the redundant encoding of values in heights did not significantly improve the interpretation of the charts, whereas the test persons could highly benefit from the system of distinct amplitudes to different data points in a star chart. Based on the largely positive results of the study, a software tool for creating relief charts by 3D printers was implemented. The software takes tabular data as input and supports the five chart types mentioned above. The result can be edited by a sighted person and is stored in Surface Tessellation Language (STL), a common file format used for printing 3D models.

3.1.2 Refreshable Tactile Displays

Over the past decades, several devices were developed capable of dynamically producing tactile raster images instantaneously. The basic principle for setting and removing the tangible dots is the same as in the case of Braille displays (see Section 2.2): each possible dot is formed by a pin of an appropriate diameter. The pins are fixed in an upright position side by side as a horizontal dot matrix. To set or remove a particular dot, the corresponding pin is raised or lowered, respectively, by a piezoelectric or electromagnetic actuator. In contrast to standard Braille displays, however, the pins of most devices for tactile graphics are not arranged as a single line of Braille characters but rather as a rectangular, equidistant raster, similar to that of a screen in order to support the representation of two-dimensional shapes.

One of the first refreshable tactile displays was introduced by Bliss [1969]. It contains an array of 24×6 photosensors whose signals control corresponding piezoelectric tactile pins. The sensors are located in a handheld scanner, which the user can move across the printed material and is connected by cable to the main unit. The pins are arranged in 24 rows by 6 columns, intended to be perceived at a time by one fingertip. The pins vibrate at a frequency of 200 Hz, since experiments conducted by the authors showed that such a stimulation resulted in higher recognition accuracy. The initial design focused on reading printed text by means of the device. In an evaluation with four participants, all four achieved reading rates greater than 10 correct words per minute, two of them a rate greater than 20 words per minute. The device was commercially produced and marketed as Optical to Tactile Converter (Optacon) by Telesensory Systems from 1971 until 1996.

Falk [1999] describes a prototype of Graphics Window Professional (GWP) developed by Handy Tech Elektronik GmbH[1], a tactile device with 24×16 piezoelectric pins at a distance of 3 mm. GWP was originally intended as a means for blind persons to access graphical user interfaces. Since the dot matrix can only represent a small part of the image, the device supports scrolling and zooming using dedicated buttons next to the tactile elements, so that regions of interest can rapidly be navigated and inspected. The author developed algorithms to adapt rich graphical inputs so as to yield more meaningful representations on the tactile device. Firstly, the visual resolution is reduced by a factor of approximately 3000 to the significantly lower tactile resolution. Secondly, as the tactile pins can only be set to one height, a colour or greyscale image is converted to a monochromatic form, by applying methods of optimal thresholds.

A commercially produced version of GWP is described by Revnitski [2005]. This version is implemented in a smaller case and offers some additional navigation keys. Furthermore, it provides an application programming interface (API) for setting the displayed dots pattern and reading certain parameters from the device. GWP was produced and sold by Handy Tech until 2009.

A project with a focus similar to that of GWP is HyperBraille [Völkel et al. 2008; Prescher et al. 2017; HyperBraille 2020]. The tactile display developed within this project is based on the Stuttgart pin-matrix device and considerably larger than GWP. It features 60×120 pins and, in addition, supports touch input from multiple fingers using capacitive measurement techniques. The device has been commercially

[1]Known as Help Tech [2021] since 2017.

produced by metec [2020] since 2012. The manufacturer currently offers more compact variants of the HyperBraille system for mobile use, as well as wireless communication to personal computers and smartphones.

A system with a different approach to navigation was proposed by Owen et al. [2009], namely a special kind of mouse for tactile exploration of graphical content. The prototype works in combination with a tablet, where the mouse is moved across the displayed graphics and its current position is detected by a radio frequency system. Between the mouse buttons, instead of a scroll wheel, a common Braille module is located which is used to display shapes at the current location as raised pins. The Braille pins can be vibrated with frequencies varying between 0 and 300 Hertz because, based on [Bliss 1969], the authors hypothesise that vibrating shapes are easier to recognise than static ones. The device is accompanied by a software library for integration with third-party systems. As a test case, the authors provide a function for Matlab which displays three-series line charts, where each of the three lines is haptically represented by pins vibrating at a distinct frequency. One of the mouse buttons can be used to retrieve the coordinates of the current mouse position. Headley and Pawluk [2010] present an enhanced version of the tactile mouse, which is capable of setting the braille pins to different amplitudes, for example to represent a height field or different graphical textures.

Hribar et al. [2012] address the problem of integrating text labels within tactile graphics with a two-layer approach. A tactile mouse is used to explore graphical content as described in [Headley and Pawluk 2010]. Whenever the user moves the mouse across a text character, the display switches to a mode in which the character is shown in Braille. This Braille representation is fixed as long as the mouse remains within the area of the character so that small movements do not impair readability. In order to facilitate a clear distinction between textual and graphical content, two options are proposed. The first is to set the pins to different amplitudes, where text characters are represented at the maximum level and graphical elements at lower ones. As an alternative, both text and graphics are represented at the same heights, but text characters are enclosed in borders. In a comparative study, nine visually impaired participants were given the task of exploring different maps to identify contained three-letter labels and textures for particular countries. The two strategies were evaluated along with the third option to represent text and graphics at the same amplitudes without any means of distinction. All three methods performed similarly in terms of finding the text labels. However, with regard to identifying graphical features, the tests showed significantly worse results for the strategy of different pin amplitudes than for the other two options.

A recent development in the field of tactile devices is Graphiti, a commercially available device produced by Orbit [2021] with a matrix of 60×40 tactile pins which can be set to variable amplitudes. It can handle graphics files in various formats from memory stick, SD card, or a host computer connected via USB or Bluetooth. Moreover, Graphiti features a High Definition Multimedia Interface (HDMI) input port, which can display the screen content of an HDMI source, refreshing the tactile display every few seconds. The pin array is touch-sensitive, enabling the user to either perform gesture commands for scrolling, zooming, and navigation or to draw shapes by moving a fingertip along the desired curves. The manufacturer provides an open API for communication via USB or Bluetooth, so that software for external devices can be developed to both set the tactile output and to read the touch input. A photograph of the Graphiti displaying an image is shown in Figure 3.1.

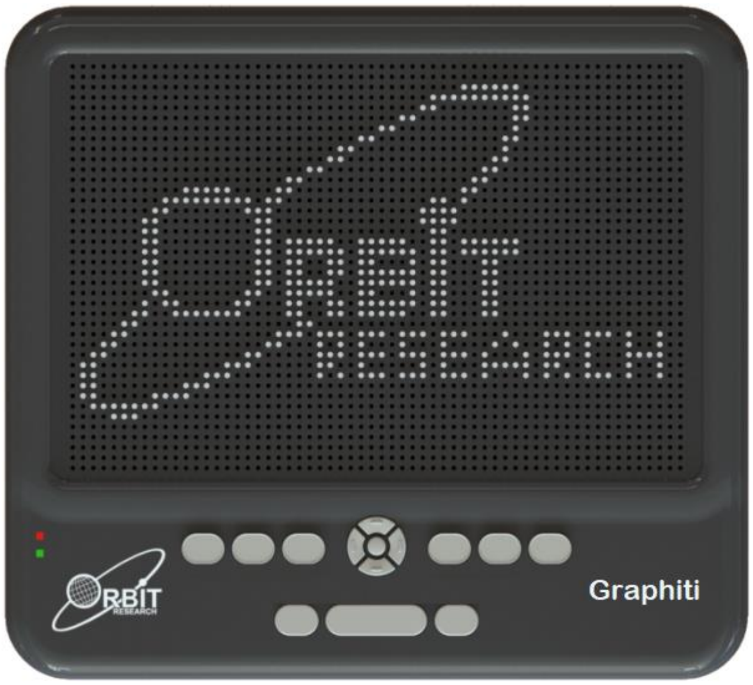


Figure 3.1: Graphiti by Orbit [2021], a portable tactile graphics display with 40×60 pins, viewed from above. Navigation keys are located in front of the pin array. The device is displaying the logo of the manufacturer. [Image provided by and used with kind permission of Orbit [2021].]

3.2 Auditory Output

In scientific literature, many solutions can be found which transform the content of a graphics document in general, or a chart in particular, into an acoustic representation. There are two basic approaches: speech output and sonification. Systems using speech output read out the graphics document using integrated speech synthesis. Systems using sonification produce an acoustic non-speech representation of the graphics document. Some systems combine both approaches.

3.2.1 Speech Output

Ferres et al. [2007] describe the iGraph-Lite software, an interactive screen reader for charts. The software takes a so-called *visual description* as input, which means a machine-readable tree structure representing the objects and characteristics of the chart. This interface was chosen in order to make the main software as independent of the chart authoring software as possible. The prototype is accompanied by a plug-in for Microsoft Excel, which automatically generates such visual descriptions on chart creation.

The core component of iGraph-Lite applies several analysis algorithms in order to derive interpretations of the data, which include the detection of trends and the identification of titles not explicitly specified as such. For example, if an axis does not have a title but its axis items are labelled according to a pattern like “Q1, 1996”, the software infers an axis title of “Quarter by Year”. This component, too, is organised in a modular design, so that the set of algorithms can be extended by plug-ins with other capabilities.

From the user’s point of view, the software offers two levels of access. Firstly, it composes a textual summary of the chart, which is available as a plain text file and includes the chart type and title as well as information about the axes and the data points with the two lowest and two highest values. Secondly, details of the data can be inspected as full-sentence speech messages by keyboard commands within a special navigation tool. For instance, it is possible to navigate among data points along the `x`-axis using the `Cursor-Left` and `Cursor-Right` keys, where the software speaks the `x` and `y` values of the selected data point and indicates whether it represents an increase or a decrease compared to the previously selected data point. Moreover, the user can skip a specified number of data points and have the textual summary spoken. The two-level user interface is intended to be integrated into web documents, where the textual summary is referenced by the longdesc attribute of the HTML element embedding the chart, and a mouse click on the graphic enables the navigation tool.

In [Ferres et al. 2013], an extended version of the system is presented. This implementation includes the use of more plug-ins to other plotting and chart authoring applications, which generate the visual descriptions from the internal object model of the respective software. The system was evaluated in two formative usability tests with ten blind and ten (blindfolded) sighted participants at two different development iterations, showing that it was considered as likeable and helpful. Based on the first usability test, some new key commands were introduced and others were adapted according to the feedback of the participants. The new commands include navigation to the first and last data point as well as to the data points with the maximum and minimum value. Existing commands were reassigned in order to achieve more consistency with the common interface of screen readers and to take into account the spatial location on the keyboard. The source code of the iGraph-Lite software is publicly available at [Ferres 2015].

3.2.2 Sonification

Choi and B. N. Walker [2010] propose a hardware system called Digitizer Auditory Graph, which transforms single-series line charts on paper to sequences of musical tones. The form factor is intended to resemble an overhead projector: a camera is fixed above the surface the chart is placed on. After this camera has captured the graphics, image recognition and edge detection are used to transform the detected data points into numerical values. Once the analysis has been completed, the data series is sonified as a sequence of piano tones with a total duration of 10 seconds, where the frequency of each tone corresponds to the value of the respective data point. The audio is generated using the Auditory Graph Model library of the Sonification Sandbox framework [B. Walker and Cothran 2003]. In an evaluation with four blind and four sighted participants, the system received positive average ratings.

Yoshida et al. [2011] present an iPhone application which maps contours of arbitrary images to synthetic sounds. Edges are extracted using computer vision. Depending on the operating mode selected, either sine tones indicate whether there is an edge at the current finger position on the touch screen, or a pulse train of variable frequency represents the distance to the next edge from the finger position. In evaluations, most of the blindfolded sighted participants could recognise most of the shapes in the test material after some initial training.

3.2.3 Combined Speech Output and Sonification

Zhao et al. [2008] introduced a hybrid solution, combining both speech output and sonification. This Windows software was originally intended for the acoustic exploration of geospatial statistical data commonly presented in maps, such as population or crime rates, but can be extended to other graphical representations like charts. The data can be displayed not only in a map but alternatively as a tabular view. Data points and particular properties can be navigated with both keyboard and touchpad. A selected value is first represented by a musical tone, whereas names and numerical details can be obtained as synthetic speech. Two blind persons were involved in the iterative design process. In an evaluation with seven blind test users, all of them rated the system as “easy to use” and “helpful”, and an average of 90 % of tasks were completed.

For the Graphics Accessible To Everyone (GATE) framework proposed by Kopecek and Oslejsek [2008], the techniques of speech and sonification are combined the other way round. An overview of an image can be obtained by a question-answer interface, where the questions are entered via speech recognition and the answers given by speech synthesis. More detailed exploration is possible using the standard numeric keypad, with each of the number keys from 1 to 9 representing one of nine rectangular sections of the image. Once a section is selected, it can recursively be explored by nine subsections the same way. For every selected rectangle, three different tones are played, where each of them represents the intensity of one of the additive primary colours red, green, and blue in this section.

Banf and Blanz [2013] present a Windows software system with a similar exploration strategy: the software analyses arbitrary images in Joint Photographic Experts Group (JPEG) format by means of computer vision and machine learning. A brief overview is given via speech output, whereas details are explored pixel-wise by the user and represented as synthesised or prerecorded sounds. These details include low-level features, such as colours and edges, as well as high-level information, like recognised objects. While exploring, the user can navigate by keyboard or touch screen. The software was evaluated qualitatively with three blind test persons and everyday pictures as material. All the participants could solve most of the tasks and appreciated the system.

3.3 Multimodal Output

Various solutions for presenting graphics to recipients with disabilities address multiple senses, in order to convey several parameters of complex visual information in parallel. One of these is the commercial IVEO system produced by the company ViewPlus Technologies [ViewPlus 2021] and described by Bulatov et al. [2005], Gardner and Bulatov [2006] and Gardner and Bulatov [2010]. The system consists of three components: IVEO Creator, IVEO Viewer, and IVEO Touchpad.

IVEO Creator can be used to annotate objects within SVG documents with accessible names and descriptions. Instead of textual information, audio recordings can be added as well. A Pro version of Creator transforms images of any format into an SVG representation using image recognition techniques. All the text labels contained in an image are converted to machine-readable text by optical character recognition and structurally associated with their corresponding object within the graphics.

IVEO Viewer is a Windows application available for free and represents the counterpart used by the recipient. An IVEO SVG file can be displayed, where each accessible name is read aloud by synthetic speech when clicking on the corresponding object, and descriptions are spoken after a double click. According to the authors, this kind of exploration may be helpful especially for users with low vision or cognitive disabilities. Moreover, the user can print the SVG file on a ViewPlus tactile graphics embosser directly from IVEO Viewer.

The third part of the system is the IVEO Touchpad, which can be connected to the computer running IVEO Viewer. Subsequently, the user places an embossed graphic on the touchpad and explores the tactile representation, where any name is read aloud on the connected computer when a finger presses on the associated object, and descriptions can be retrieved pressing a certain key combination. A more recent version of the IVEO system is presented by Gardner [2016]. The improvements include better automatic and manual image simplification using edge detection, automatic generation of Braille labels for the tactile graphics, a digital infra-red pen as an alternative to touch gestures for input, and various usability enhancements.

Wall and Brewster [2006] introduced a system called Tac-tiles for exploring pie charts by speech, sonification, and a tactile display using a graphics tablet with a pen as an input device. The non-speech audio consists of musical tones which are played when reaching another sector of the pie chart, indicating its proportion. The tactile display shows borders of the sectors when reached. Exact numerical values can be obtained as speech output. For orientation and guidance within the chart, a circular overlay tile is placed onto the tablet. In a qualitative evaluation, six blind test users (some with residual vision) answered 89 % of all the questions correctly. All the participants appreciated the system on the whole, preferring audio output to the (redundant) tactile output.

The GraCALC software presented in [Goncu and K. Marriott 2015] takes a mathematical function or tabular data as input. A web service based on the statistics software R generates a textual summary of important features, such as extrema and turning points, and plots the function or data in a dedicated format. The result can be downloaded to a corresponding iOS application, in which blind users can explore the plot by sound and vibration feedback depending on the finger positions on the touch screen. In an evaluation with ten visually impaired students, eight of them stated that they preferred this system to classic tactile graphics on paper due to its high flexibility and interactivity.

A multimodal software solution for accessing and exploring mathematical graphics on the tactile display GWP (see Subsection 3.1.2) was developed by Revnitski [2005]. The software called Plot Explorer provides a refreshable tactile representation of visualisations combined with speech and Braille output via the screen reader JAWS. It takes a tree structure as input, which contains the underlying data and other semantics of the visualisation. The actual plot is then rendered by the Plot Explorer core application in two formats, namely visually and as a representation optimised for the intended multimodal output. The tactile image is displayed on the GWP, where the user can zoom and navigate by means of the mouse, keyboard, or buttons of the tactile device. Furthermore, the user can choose between several output modes, hide and unveil the axes, show only the data points, or show interpolation lines. The data structure can be passed to Plot Explorer using a corresponding dynamically linked library. This library has been integrated into the mathematics software Maple by Maplesoft.

Rotard et al. [2004] also presented a system for the multimodal exploration of graphics in conjunction with a tactile display. While the prototype of the system was tested and evaluated using a particular 60×120 pin device, the interface is described to be flexible so that the software can support arbitrary tactile displays. The system accepts SVG documents as input and performs a transformation to a simplified representation. Once a document has been analysed completely, SVG elements can be navigated step by step, and groups of elements can be entered recursively. The shape of the currently selected element is displayed on the tactile device, and its accessible name and description are read aloud by speech synthesis.

Engel et al. [2019] present a Java application called SVG-Plott for generating SVG charts optimised either for visual output, tactile embossing, or exploration on a tactile display. If the latter is chosen as target output, titles, descriptions, and interactive regions are added to the file so that the user also receives speech feedback when moving the finger to objects of the chart on an audio-tactile device. The system accepts one or multiple Comma-Separated Values (CSV) files as data input and can create stacked and grouped bar charts, line charts, and scatter plots. The programme can be controlled either via command line or a graphical user interface (GUI). Within the GUI, data can also be entered, deleted, and edited by the user. Depending on the data, the chart type, and the target output (screen, tactile graphics embosser, or tactile display), many parameters of the SVG file are chosen automatically, but can also be customised by the user. The GUI was designed to be accessible so that, according to the authors, creating charts is possible not only for sighted but also for blind users. In an evaluation of embossed charts created by the application with two blind participants, both of them could recognise the charts well and solve most of the given tasks, with partly worse results for scatter plots than for bar and line charts. In another study, the usability of the GUI was evaluated with two blind and ten sighted persons. All of them performed all the given tasks successfully and appreciated the structure of the GUI. The source code of the software is publicly available [Harlan et al. 2019].

3.4 Screen-Reader-Friendly Output

The Access2Graphics project introduced by Altmanninger and Wöß [2008] aims to achieve the inclusion of all humans with or without any kind of impairment. The authors argue that one form of graphical representation cannot serve the diverse needs of recipients with different disabilities. For this reason, a web-based framework is proposed, which validates an SVG document against the guidelines published in [Dürnegger et al. 2010] (see Subsection 4.1.2) and can transform it into various output formats.

The core component is a server-side programme designed to be used by any web application. The software runs a database in which every user can store a profile with characteristics relevant for the target output, such as any disabilities, used assistive technologies, and individual preferences. When a user has created such a profile and encounters a valid SVG document on a web page which includes the Access2Graphics service, an additional link is inserted next to the graphic. After activating this link, the graphic is sent to the service, which then returns a representation according to the individual characteristics specified in the profile of the particular user. For example, a screen reader user receives a hierarchical structure of textual elements according to the logical structure of the objects in the graphic. In the case of a chart, a tabular view of the data can also be obtained. A user who has indicated having colour deficiencies, by contrast, is provided the SVG with modified colours according to the details specified in the profile or, as an alternative, a greyscale version of the graphic.

In the statistical software environment R, a package called BrailleR is available which generates a textual representation for certain chart types [Godfrey 2013; Godfrey 2019]. The text output comprises a listing of all the data points and is derived from an internal object of the software on chart creation. Fitzpatrick et al. [2017] and Godfrey et al. [2018] present an extension to the BrailleR package creating accessible charts which can be displayed in any browser and explored by keyboard commands in combination with screen reader output.

The frontend is based on the DIAGcess library [Sorge 2016] (see Subsection 4.1.2) and provides a hierarchical navigation model consisting of at least three layers of abstraction. The user can move within one layer using the `Cursor-Left` and `Cursor-Right` keys, and between the layers with the `Cursor-Up` and `Cursor-Down` keys, where upward navigation corresponds to an increased level of abstraction. Whenever an item is selected, its title is conveyed to a running screen reader and the corresponding object within the graphical representation is highlighted. An optional synchronised magnification of the selected object is possible as well. If the user has enabled *verbose mode*, an additional description is passed to the screen reader. If configured appropriately, the screen reader reads out the title and optionally the description and/or outputs it to a connected Braille display.

The root of the hierarchical structure consists of a so-called top-level summary, which includes the chart type and title, in verbose mode also the number of data points as well as the titles and ranges of the axes. The next level is the major component layer, whose items represent a title component including all the subtitles and other overall text information, one component for each axes, and a component for the whole dataset. On the lower layers, single items such as data points can be selected. In the case of charts resulting from continuous functions, the BrailleR extension divides the output into small sections which can be navigated instead. Furthermore, statistical values are also provided, such as maxima, minima, quartiles, and outliers. Two blind persons participated in the development process, and feedback was requested from other blind users, most of whom had a mathematical background but no experience working with R. The test persons used their own devices with various combinations of operating systems, browsers, and screen readers and appreciated the means of exploration in general. However, many of them had problems understanding the charts without asking for additional information. Moreover, some of them requested other output modalities, in particular, Braille and sonification.

Carberry et al. [2012] introduce the Summarizing Information Graphics Textually (SIGHT) software, an intelligent system which generates textual descriptions of single-series line charts and bar charts automatically, trying to convey the essential message and important visual features of the chart. After retrieving this description, the user can request follow-up information interactively. The system is designed for readers of popular media rather than scientific literature, which means that the focus is not put on detailed exploration of the data, but on retrieving high-level information expressed by the chart, even when the user is not familiar with consuming numerical data.

The application is organised in a modular architecture. A browser helper object for Microsoft Internet Explorer searches for graphics within a web document after loading and performs some initial image recognition on each graphic, trying to detect whether it is likely to be a chart. If so, the software adds a message to the alternative text of the graphic, saying that the user can request a description by pressing the key combination `Control+Z`. After the user has entered this command, the image is sent to the Visual Extraction Module which tries to identify all the objects of the chart and their properties using image recognition. Other components of the software then derive higher-level semantics by probability-based algorithms. The result is returned to the frontend and presented to the user in a separate window. In an evaluation with seven visually impaired participants, all of them could solve all the given tasks correctly and rated the system as very helpful and easy to use.

A newer version of the SIGHT system is presented in [Moraes et al. 2014]. This version is additionally capable of processing stacked bar charts and of adapting the level of the language for the descriptions to that of the language used in the article containing the analysed chart. For instance, the description of a chart is expressed in every-day language when embedded in newspaper articles, whereas it is more detailed and mathematical in the case of scientific documents. Moreover, the new version was designed as a client-server architecture. While the previous version needs to be completely installed on the user's local machine, most processing is now performed on a cloud-based backend. The user interface is provided as a browser plug-in for Google Chrome. In an evaluation with several line charts, four blind test persons performed almost as well in answering questions about the charts as sighted participants viewing the graphical representations.

Another commercial solution is the SAS Graphics Accelerator [Summers et al. 2018; SAS 2019]. The software is available free of charge as a plug-in for Google Chrome and offers alternative output modalities for STEM graphics, including textual descriptions provided by the graphic's author, tabular views, interactive sonification controlled by the user, and modified visual representations for partially sighted recipients. Graphics Accelerator can be used as a self-sufficient system or in combination with a screen reader. It accepts charts and maps of more than ten different types created with other SAS applications and conforming to the recommendations by Summers et al. [2018]. Moreover, it offers the Laboratory function, which lets the user create visualisations as well. For this purpose, data can be entered as tabular structures, imported from CSV and Microsoft Excel files, or extracted from web documents opened in the browser.

3.5 Charting Libraries with Accessibility Features

Several of the existing web programming libraries for generating charts from datasets, so-called charting libraries, have support for the production of screen-reader-friendly output. While some of them do so by default, in the case of others, accessibility is an option which only takes effect when explicitly activated by the author of the chart. The accessible output ranges from textual summaries, through table representations of the data, to web applications enabling the user to navigate the objects of a chart interactively by keyboard and speech or Braille output. Examples of such charting libraries will be described in the following subsections.

3.5.1 Highcharts

The commercial software Highcharts offers an *Accessibility module*, which is being developed with the involvement of users with disabilities [Highcharts 2021b; Highcharts 2021a]. In order to produce accessible charts, this module needs to be added to the web document along with the other library modules. Once included, it automatically attaches `tabindex` attributes as well as ARIA roles and properties [W3C 2017a] to certain SVG elements of the generated chart. This enables navigation between these elements by keyboard and exposes the labels of focused elements to a running screen reader, so that it can output this information by speech and/or Braille. All the labels are generated automatically by the software from the underlying data according to a pattern which can be adapted by the author of the chart.

When a user reaches the SVG by means of the `Tab` key, the focus is set to the first data point of the first data series, and a running screen reader is supplied with the title and subtitle of the chart, as well as the labels of the first data series and first data point. Afterwards, the data points can be navigated within one data series by means of the `Cursor-Left` and `Cursor-Right` keys and across multiple series using the `Cursor-Up` and `Cursor-Down` keys. Whenever a new data point is focused, the screen reader is informed about its label (for example, x and y coordinates). In addition, if the user has moved to a different data series, the label of the newly selected data series is conveyed to the screen reader as well. If a focused data point contains a link, it can be activated by the `Enter` or the `Space` key. In particular, Highcharts provides a facility to drill down into aggregate data points by following such a link.

Another press of the `Tab` key takes the user to the first item of the legend, if available for the particular chart, where the remaining legend items can again be navigated using the `Cursor` keys. The legend items are represented as buttons, which can be activated by the `Enter` or the `Space` key to hide the corresponding chart object (for example, a data series) and to unveil it again. Regarding navigation by the `Cursor` keys, however, it should be noted that this functionality is available in most screen readers only if the latter are not running in their default mode for web browsing (i.e. not running Virtual Cursor mode in JAWS or Browse Mode for NVDA), because in this case, the key presses are captured by the screen reader.

If the default mode for web browsing is active, from a blind user's perspective, the information on the data points is listed sequentially by name, grouped by data series, and followed by the legend items, so that all the objects can be viewed using the navigation facilities of the respective screen reader. In this interaction mode, the user can also read several additional automatically-generated text summaries which are presented in an invisible region placed into the accessibility tree before the SVG. They include the chart type, the number of data series, and information about all the axes. Moreover, the user can choose to view the data as a standard HTML table. Additional descriptions of the chart, data series, and data points can be provided by the author and are then included in the accessible representation.

The default colours for Highcharts are chosen such that neighbouring objects can also be distinguished by users with colour deficiencies. Every chart can also be exported as an SVG file which contains the same accessibility attributes and is additionally optimised for tactile embossing. Sample demonstration charts created using the accessibility module of Highcharts can be found at [Highcharts 2021c]. An example of the ARIA roles and properties as well as the other attributes used to indicate chart objects is shown in Listing 3.1. Further technical details can be found in Subsection 4.5.6.

3.5.2 FusionCharts

The commercial library FusionCharts provides a so-called *accessibility extension* [FusionCharts 2020], similar to the Accessibility module of Highcharts. This FusionCharts extension, too, needs to be included within the document of the chart in order to take effect. It then attaches `tabindex` attributes and ARIA roles and properties to certain SVG elements of the generated chart, so that keyboard navigation between these elements is possible and labels of the focused elements are conveyed to a running screen reader. All the labels are automatically generated according to a pattern which is customisable by the author of the chart.

The first focusable element in the visualisations produced by FusionCharts is the root `<svg>` element. When navigating to it, which can be achieved by means of the `Tab` key, a running screen reader is supplied with the label of the chart. By pressing the `Tab` key another time, the focus is set to the first data point of the first data series. Subsequently, the data points can be navigated within a data series by the `Cursor-Up` and `Cursor-Down` keys and across multiple series using the `Cursor-Left` and `Cursor-Right` keys. Whenever a data point is focused, the screen reader is informed about its label, for instance, its x and y coordinates, index number, the total number of data points in its data series, the title and index number of its data series, and the total number of data series. If a focused data point contains a link, it can be activated by the `Enter` or the `Space` key. In particular, FusionCharts provides a facility to drill down into aggregate data points by following such a link.

Pressing the `Tab` key another time, the focus is moved to the first item of the legend, if one is available for the chart. Afterwards, the remaining legend items can again be navigated using the `Cursor` keys. The items are represented as buttons, which can be activated by the `Enter` or the `Space` key to hide the corresponding chart object (for example, a data series) and to unveil it again. As in the case of Highcharts, navigation with the `Cursor` keys is available in combination with most screen readers, only if they are not running in their default mode for web browsing, since otherwise these key presses are reserved for navigation functions of the screen reader itself. Demonstration charts created with the accessibility extension of FusionCharts are provided at [FusionCharts 2020]. An example usage of the ARIA and other attributes applied by FusionCharts is presented in Listing 3.2; other technical details are described in Subsection 4.5.9.

3.5.3 Semiotic

Semiotic is a free charting library based on the JavaScript frameworks React and D3. It is distributed with all its accessibility features already included by default [Meeks and Lu 2020]. In a similar way to Highcharts and FusionCharts, the dataset can be focused by means of the `Tab` key, where a running screen reader is informed about the chart type and the number of contained navigable items, that is, the data points

```
<svg class="highcharts-root" tabindex="0" version="1.1" xml:lang="en"
  lang="en" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink" viewBox="0 190 750 346">

<text class="highcharts-title" id="title">Line Chart of Sales</text>

<text class="highcharts-subtitle" id="desc">
  Chart of sales for 12 months in year 2012 for Salespersons A and B.
</text>

<!-- Y Axis -->

<g class="highcharts-axis highcharts-yaxis">
<line x1="98" y1="443" x2="98" y2="212" />
<text class="highcharts-axis-title" id="y-title"
  transform="matrix(0 -1 1 0 20 355)">
<tspan>Sales in €</tspan>
</text>
</g>


<g class="highcharts-axis-labels highcharts-yaxis-labels">

<g>
<text transform="matrix(1 0 0 1 38 215)">40,000</text>
<line x1="98" y1="212" x2="93" y2="212" />
</g>

<g>
<text transform="matrix(1 0 0 1 38 262)">35,000</text>
<line x1="98" y1="258" x2="93" y2="258" />
</g>

...
</g>

<!-- X Axis -->

<g class="highcharts-axis highcharts-xaxis">
<line x1="98" y1="443" x2="677" y2="443" />
<text class="highcharts-axis-title" id="x-title"
  transform="matrix(1 0 0 1 383 525)">
<tspan>Month</tspan>
</text>
</g>


<g class="highcharts-axis-labels highcharts-xaxis-labels">

<g>
<text id="x-2012-01" transform="matrix(0.7 0.7 -0.7 0.7 102 466)">
<tspan>January 2012</tspan>
</text>
<line x1="98" y1="443" x2="98" y2="448" />
</g>
```

Listing 3.1: Sample chart source code produced by Highcharts. The SVG chart elements are annotated with standard ARIA roles and properties, as well as being assigned dedicated class names. In addition, the root `<svg>` element and the data points have the `tabindex` attribute set to support keyboard navigation.

```

<g>
<text id="x-2012-02" transform="matrix(0.7 0.7 -0.7 0.7 155 466)">
<tspan>February 2012</tspan>
</text>
<line x1="150" y1="443" x2="150" y2="448" />
</g>

...
</g>


<g class="highcharts-series-group">

<!-- Data Points A -->

<g role="region" aria-label="Salesperson A, line 1 of 2 with 12 data points."
  class="highcharts-markers highcharts-series-0 highcharts-line-series"
  id="data-a">

<polyline points="
   98,319 150,331
  150,331 203,309
  203,309 255,280
  255,280 308,322
  308,322 361,303
  361,303 413,375
  413,375 466,382
  466,382 519,353
  519,353 571,315
  571,315 624,298
  624,298 677,246
  " />

<rect role="img" class="highcharts-point" aria-label="1. January 2012, 28366.
  Salesperson A" tabindex="-1" x="93" y="315" />

<rect role="img" class="highcharts-point" aria-label="2. February 2012, 27050.
  Salesperson A" tabindex="-1" x="146" y="327" />

...
</g>
</g>


<!-- Legend -->

<g class="highcharts-legend" id="legend">

<g class="highcharts-legend-item highcharts-series-1" id="legend-a">
<rect x="168" y="210" />
<text id="legend-text-a" x="185" y="220"
  font-family="Verdana" font-size="14">Salesperson A</text>
</g>

...
</g>

</svg>
```

Listing 3.1 (cont.)**:** Sample chart source code produced by Highcharts.

```
<svg role="application" class="raphael-group-1-parentgroup" tabindex="0"
  focusable="true" aria-label="This is a multi series line chart created
  with FusionCharts Suite XT. Title of the chart is Line Chart of Sales.
  Month is plotted on x-axis and Sales in € is plotted on y-axis"
  version="1.1" xml:lang="en" lang="en" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink" viewBox="0 190 750 346">

<g class="raphael-group-1-caption">
<text id="title">Line Chart of Sales</text>
</g>

<!-- Y Axis -->

<g>

<line x1="98" y1="443" x2="98" y2="212" />

<g class="raphael-group-1-dataset-axis-name">
<text id="y-title" transform="matrix(0 -1 1 0 20 355)">Sales in €</text>
</g>

<g class="raphael-group-1-dataset-axis">

<g>
<text transform="matrix(1 0 0 1 38 215)">40,000</text>
<line x1="98" y1="212" x2="93" y2="212" />
</g>

<g>
<text transform="matrix(1 0 0 1 38 262)">35,000</text>
<line x1="98" y1="258" x2="93" y2="258" />
</g>

...
</g>
</g>


<!-- X Axis -->

<g>

<line x1="98" y1="443" x2="677" y2="443" />

<g class="raphael-group-1-dataset-axis-name">
<text id="x-title" transform="matrix(1 0 0 1 383 525)">Month</text>
</g>

<g class="raphael-group-1-dataset-axis">

<g>
<text id="x-2012-01" transform="matrix(0.7 0.7 -0.7 0.7 102 466)">
  January 2012</text>
<line x1="98" y1="443" x2="98" y2="448" />
</g>
```

Listing 3.2: Sample chart source code produced by FusionCharts. The SVG chart elements are annotated with standard ARIA roles and properties, as well as being assigned dedicated class names. The data points and legend items have the tabindex and focusable attributes set to support keyboard navigation.

```

<g>
<text id="x-2012-02" transform="matrix(0.7 0.7 -0.7 0.7 155 466)">
  February 2012</text>
<line x1="150" y1="443" x2="150" y2="448" />
</g>

...
</g>
</g>


<!-- Data Points A -->

<g class="raphael-group-1-plot-group" id="data-a">

<polyline points="
   98,319 150,331
  150,331 203,309
  203,309 255,280
  255,280 308,322
  308,322 361,303
  361,303 413,375
  413,375 466,382
  466,382 519,353
  519,353 571,315
  571,315 624,298
  624,298 677,246" />

<rect aria-label="Sales in € of Salesperson A for Month January 2012 is 28366.
  Plot 1 of 12. Series 1 of 2" tabindex="0" focusable="true" x="93" y="315" />

<rect aria-label="Sales in € of Salesperson A for Month February 2012 is 27050.
  Plot 2 of 12. Series 1 of 2" tabindex="-1" focusable="true" x="146" y="327" />

...
</g>


<!-- Legend -->

<g class="raphael-group-1-legend" id="legend">

<text role="button" aria-label="Toggle the visibility of Salesperson A."
  tabindex="0" focusable="true" id="legend-text-a" x="185" y="220"
  font-family="Verdana" font-size="14">Salesperson A</text>

...
</g>

</svg>
```

Listing 3.2 (cont.)**:** Sample chart source code produced by FusionCharts.

when viewing a bar or a pie chart and the lines in the case of a line chart. Navigation amongst data points and data series is then achieved using the `Cursor` keys. In the case of bar and pie charts, single data points can be focused, exposing the name and value of the data point to the screen reader. Within line charts, by contrast, navigation is possible only between entire lines, supplying the screen reader with a summary about the focused line, including the total number of contained data points and the starting and ending values. The accessibility-related attributes used by Semiotic are illustrated in Listing 3.3. More details about the accessibility techniques applied by Semiotic are given in Subsection 4.5.7. Demonstration charts produced by Semiotic can be found at [Meeks and Lu 2020].

3.5.4 amCharts

The commercial library amCharts, too, generates accessible SVG content "out of the box" [amCharts 2020a]. The elements of a chart can be navigated by means of the `Tab` key, where the focused object is visually marked with a high-contrast outline and automatically-generated labels of the elements are conveyed to a running screen reader. As soon as an object of the chart receives focus, the title and the type of the chart is exposed. Whenever focus moves to a data point, the screen reader is informed about its name and value, for example, its x and y coordinates. However, if the number of data points is larger than a certain threshold (by default, 20 for bar charts and 50 for pie charts), focusing of single data points is deactivated in order to improve clarity for the user. For the same reason, data points belonging to line charts cannot be focused and have no label by default.

Controls for modifying the displayed content are accessible by `Tab` focus and screen reader as well. Actions usually triggered by dragging the mouse cursor, such as those for scroll bars, can alternatively be performed by focusing the respective object and then pressing the `Cursor` key corresponding to the direction the mouse would be moved in. Zoom buttons and legend items to toggle the visibility of chart elements can be activated by means of the `Enter` key. All the accessibility labels of the chart elements are automatically generated according to a default pattern, which can be customised by the author of the chart. Moreover, the author can assign accessible names and descriptions to specific chart elements, declare them focusable, and/or state that tooltips associated with them shall be displayed not only on mouse hovering but also on focus. The chart can be configured to fill chart elements with patterns which can be easily distinguished by users with low vision. Technical details about the attributes used by amCharts are described in Subsection 4.5.8. Sample charts produced by amCharts can be found at [amCharts 2020b].

3.5.5 AnyChart

The commercial software AnyChart offers two different accessibility modes, one of which can be chosen on chart creation [AnyChart 2020a]. Chart Elements accessibility mode is enabled by default, generating an overall summary of the chart with the chart type and title, the number of data series, and information about the axes. The pattern for this summary can be modified by the author of the chart. In addition, the author can activate accessibility mode for data series, which then generates labels for them, and can adapt these labels as well.

By contrast, in Data Table accessibility mode, the data are represented to screen readers as an invisible standard HTML table, where the overall summary of the chart is used as the title, data series correspond to columns of the table, and the x-axis items or categories to rows of the table. Information on the ARIA markup is given in Subsection 4.5.10, for demonstrations of accessible charts, see [AnyChart 2020a; AnyChart 2020b].

3.5.6 evoGraphs

The evoGraphs library [Sharif 2015a; Sharif and Forouraghi 2018] is based on the JavaScript framework jQuery, and creates bar charts and pie charts along with a textual description for screen reader users. This text contains the chart type and title, labels and values for all the data points, and some statistical characteristics, all expressed in full sentences. The description is visually hidden, while the created graphic is excluded from the accessibility tree using the `aria-hidden` property.

```
<svg class="visualization-layer" version="1.1" xml:lang="en" lang="en"
  xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink" viewBox="0 190 750 346">

<text class="frame-title" id="title">Line Chart of Sales</text>

<g role="group" class="data-visualization" aria-label="Visualization with
  a complex title. Use arrow keys to navigate elements.">

<g class="axis axis-labels">

<!-- Y Axis -->

<g class="axis left y" aria-label="left axis from 15,000 to 40,000">

<line x1="98" y1="443" x2="98" y2="212" />

<g class="axis-title axis left y">
<text id="y-title" transform="matrix(0 -1 1 0 20 355)">Sales in €</text>
</g>

<g>
<text class="axis-label" transform="matrix(1 0 0 1 38 215)">40,000</text>
<line x1="98" y1="212" x2="93" y2="212" />
</g>

<g>
<text class="axis-label" transform="matrix(1 0 0 1 38 262)">35,000</text>
<line x1="98" y1="258" x2="93" y2="258" />
</g>

...
</g>
</g>


<g role="group" class="lines" aria-label="2 lines in a line chart" tabindex="0">

<!-- Data Points A -->

<g tabindex="-1" id="data-a">

<polyline role="img" class="xyframe-line" aria-label="12 point line starting
  value 28,366 at January 2012 ending value 28,490 at December 2012" points="
   98,319 150,331
  150,331 203,309
  203,309 255,280
  255,280 308,322
  308,322 361,303
  361,303 413,375
  413,375 466,382
  466,382 519,353
  519,353 571,315
  571,315 624,298
  624,298 677,246" />
```

Listing 3.3: Sample chart source code produced by Semiotic. SVG chart elements are annotated with standard ARIA roles and properties, as well as being assigned dedicated class names. The `<g>` elements of the dataset and the data series are assigned the `tabindex` attribute to support keyboard navigation.

```

<rect x="93" y="315" />

<rect x="146" y="327" />

...
</g>

</g>
</g>


<!-- X Axis -->

<text id="x-title" transform="matrix(1 0 0 1 383 525)">Month</text>

<g class="ordinal-labels">

<line x1="98" y1="443" x2="677" y2="443" />

<g>
<text id="x-2012-01" transform="matrix(0.7 0.7 -0.7 0.7 102 466)">
  January 2012</text>
<line x1="98" y1="443" x2="98" y2="448" />
</g>

<g>
<text id="x-2012-02" transform="matrix(0.7 0.7 -0.7 0.7 155 466)">
  February 2012</text>
<line x1="150" y1="443" x2="150" y2="448" />
</g>

...
</g>


<!-- Legend -->

<g id="legend">

<g id="legend-a">
<rect x="168" y="210" class="marker-square" />
<text id="legend-text-a" x="185" y="220"
  font-family="Verdana" font-size="14">Salesperson A</text>
</g>

...
</g>

</svg>
```

Listing 3.3 (cont.)**:** Sample chart source code produced by Semiotic.

The library takes data in JavaScript Object Notation (JSON) format as input. Its functionality and the accessibility of its output was tested on various combinations of browsers and screen readers with sighted and visually impaired users, but the details and results of these tests are not specified. However, Mirri et al. [2017] applied both the evoGraphs system and the D3 library to automatically generate charts for scientific papers out of data in CSV format. As a basis they used the Research Articles in Simplified HTML (RASH) format and added extensions using either evoGraphs or D3. Comparative tests performed by a blind co-author of this paper on several combinations of different operating systems, browsers, and screen readers showed that evoGraphs produced positive results, whereas the charts generated by D3 were hardly accessible by screen reader. A demonstration of the evoGraphs System can be found at [Sharif 2015b].

3.5.7 ChartMaster

ChartMaster [Zou and Treviranus 2015] does not represent a self-contained charting library as such, but rather a software component which is intended to be integrated into existing systems for creating stock market charts. The tool lets the user navigate and search charts of past stock trading data by screen reader and keyboard. The web-based user interface consists of several drop-down menus enabling the user to access particular data points and statistical values and to choose the time frame for which the information should be displayed.

The user interface was designed in an iterative process involving usability tests with 18 participants, of whom 16 were blind or partially sighted. All of them appreciated the tool and stated that it helped them to understand the charts and to accomplish the given tasks. Moreover, the usability study confirmed the hypothesis that representing a complex chart only as a numeric table does not provide the same level of access for a screen reader user. The tool does not extract any information from the visual representation, but from the underlying data on chart generation.

3.6 Describler

Describler [Schepers 2015a; Schepers 2015b; Schepers 2017] is a client-side web application written in JavaScript, which lets the user explore charts and flow diagrams contained in SVG files by means of keyboard or mouse navigation in combination with speech synthesis. Using the web interface of Describler provided at [Schepers 2015a], an SVG file can either be chosen from a drop-down list of sample charts or opened from the local computer of the user. Once a file has been loaded, the application analyses the contained SVG structure with regard to ARIA roles, labels, and other ARIA properties. Describler identifies one or multiple charts contained in the SVG file, as well as certain objects of each chart, such as axes, legends, and data points, and extracts the information conveyed by these objects.

The software supports two-dimensional charts, such as pie charts, bar charts, and line charts. For the analysis, it assumes the presence of dedicated ARIA markup, as described in Subsection 4.5.1. It creates an array of all the focusable elements within the SVG document, that is, all the elements with the `tabindex` attribute set. Afterwards, the graphics document is shown within Describler's web interface, and its objects can be focused by keyboard or mouse. Whenever one of these objects is focused this way, Describler displays information on the object within a text box and reads it aloud. Throughout this subsection, this manner of output will be abbreviated as *speaking* or *reading aloud*. The navigation facilities and the speech synthesis work independently of any screen reader installed or running. To generate the speech output, Describler uses the services of the browser and the local operating system via the Web Speech API [MDN 2019].

Once an SVG document has been analysed completely, pressing the `Tab` key sets the focus to its `<svg>` root element. From this moment on, keyboard interaction with Describler is enabled. The user can navigate between those elements of the SVG having a `tabindex` of `0`. The order of the focused elements corresponds to the order in which they appear in the source code of the document. The user can move to the subsequent chart object by means of the `Tab` key and to the preceding object by pressing `Shift+Tab`.

If not blocked by assistive technologies, forward navigation is also possible using the `Cursor-Right` or the `Cursor-Down` key, backward navigation by the `Cursor-Left` or the `Cursor-Up` key. Independent of the keys used, navigation is cyclic, that is, moving forward from the last focusable SVG element or backward from the first focusable element sets the focus to the first or last element, respectively.

Whenever a chart element is focused, information about the chart is spoken. Afterwards, for most objects, a menu with context-sensitive options is available, where the user can choose an option by pressing a certain number key. All the menu items and their corresponding number keys are presented as full-sentence text prompts of the form "*Press. . . for/to* . . . ", which are also read aloud. As an alternative to keyboard interaction, the graphics elements can be focused by clicking with the left mouse key, either in the same order as for keyboard navigation using dedicated Forward and Backward buttons in the graphical user interface (GUI), or by directly clicking on the target object. The items in the context menu for the selected chart object can be chosen from a drop-down list.

The information spoken and the menu options offered by Describler vary by the kind of the focused element. By default, the content of the next descendant <title> element is spoken. If a descendant <desc> element is also present, the option "*more details*" can be chosen by pressing the key `1`, which causes the software to repeat the <title> and to speak the content of the <desc> element. With example texts taken from the sample charts provided at [Schepers 2017], the following subsections describe how specific chart elements behave.

3.6.1 Chart Root

The word "*chart*" and the chart title are spoken d times, where d is the number of datasets the chart contains. The following menu options are available:

1. "*Chart Statistics*": The word "*chart*" and the chart title are repeated. Afterwards, the chart type and statistical values of the first dataset are spoken. These include the total number of contained data points, the minimum and maximum data point values and the range between them, the average, the median, and the sum of all the values. If the chart contains multiple datasets, the text sequence is spoken for each dataset analogously.

2. "*Data Points from Lowest to Highest*": The word "*chart*" and the chart title are repeated. Afterwards, the phrase "*Lowest to highest:*" is spoken, and all data points of the first dataset are enumerated in increasing order by their values. For each data point, its name and value (for example, x and y coordinates) are spoken. If the chart contains multiple datasets, the text sequence is spoken for each dataset analogously.

3. "*Data Points from Highest to Lowest*": The word "*chart*" and the chart title are repeated. Afterwards, the phrase "*Highest to lowest:*" is spoken, and all data points of the first dataset are enumerated in decreasing order by their values. For each data point, its name and value (for example, x and y coordinates) are spoken. If the chart contains multiple datasets, the text sequence is spoken for each dataset analogously.

4. "*Trend Sonification*": The values of all the data points are sonified in the order they appear in the SVG source code. Sonification is implemented as a sine tone, whose frequency increases and decreases with the magnitude of the values.

3.6.2 Axes

The type of axis (for example, "*x-axis*") is spoken, followed by its title, number of axis items, first axis item, and the last axis item. The following menu options are available:

1. "*Axis Labels*": All labels of the axis are spoken in the order they appear in the SVG source code.

2. "*Select Data Points by Axis Label*" (only for x-axes): All labels of the x-axis are spoken as submenu prompts with associated number keys. When the user presses one of these keys, the focus is set to

that data point in the first dataset which refers to the corresponding x-axis item. The subsequent behaviour corresponds to the default when focusing a data point (see below).

3.6.3 Data Points

The index of the focused data point and the total number of data points in the dataset is spoken (for example: "*Data point 1 of 12.*"), followed by the name and the value of the data point (for instance, x and y coordinates). The following menu options are available:

1. "*More Details*": The name and the value of the data point are repeated. Afterwards, its value is compared to that of the data point previously focused: Describler announces "*This is an increase of*" (or "*decrease of*"), the absolute difference of the values of the two data points, and the name and value of the previous data point. Finally, the index number and the colours of the current data point are spoken.

   For example, "*This is a decrease of 1316 from the last value of 28366 for January 2012. This is the 2nd data point. The color is Contessa, and the outline is Contessa.*"

2. "*Comparison to All Other Data Points*": The name and the value of the data point are repeated. Afterwards, the relationship in percent between the value of the current data point and each of the other data points contained in the same dataset is spoken.

3. "*Comparison to a Specific Data Point*": The name and the value of the data point are repeated. Afterwards, all labels of the x-axis are spoken as submenu prompts with associated number keys. When the user presses one of these keys, the name and the value of the data point are repeated another time, and the focused data point is compared to the data point in the same dataset with the corresponding x-axis label. This comparison includes the absolute difference in combination with the word "*higher*" or "*lower*", and the relationship in percent.

4. "*Data Point Statistics*": The name and value of the current data point are repeated. Afterwards, its value is compared to statistical values of the containing dataset. In particular, the absolute differences to the average, median, minimum, and maximum value are spoken in combination with the word "*higher*" or "*lower*". Finally, Describler speaks the relationship in percent to the sum of all the values of the dataset.

   For example: "*May 2012: 28050. This is 268.33 below the mean value of 28318.33, and 554.00 below the median value of 28604. This is 6544.00 above the low value of 21506, and 8204.00 below the high value of 36254. May 2012 is 8.25% of the total value of 339820.*"

3.6.4 Legend Root

The word "*Legend*" is spoken, followed by the title of the legend and the number of contained items.

3.6.5 Legend Item

The word "*Legend item*" is spoken, followed by the index number of the item, the total number of items, and the title. For example, "*Legend item 1 of 3. Happy*". The following option is available:

1. "*A List of All Applicable Data Points*": All data points associated with the current legend item are spoken.

Chapter 4

Semantic Enrichment of SVG Charts

While the previous chapter presented various user interface solutions to enable visually impaired recipients to explore charts, this chapter discusses techniques to prepare digital charts such that they are accessible for computer users with disabilities. As in general web accessibility, the two essential elements are the embedding of underlying data and the assignment of semantics to objects of interest by annotating their internal representations with appropriate machine-readable information. The vast majority of solutions to this problem are based on Scalable Vector Graphics (SVG), a textual XML-based markup language for defining graphical documents, and the Accessible Rich Internet Applications (WAI-ARIA) suite introduced in Chapter 2.

Most of the solutions presented in this chapter still require special software to transform the resulting graphics into a representation meaningful for blind users. A long-term goal could be to develop an ARIA standard which can be implemented directly in common browsers and assistive technologies. It should be noted that the entire topic is currently work-in-progress and standardisation efforts are still ongoing. Unless stated otherwise, the following sections describe the situation at the time of writing this thesis.

4.1 Formats for Accessible Graphics

Scalable Vector Graphics (SVG) is an XML-based markup language for 2d vector graphics. SVG Version 1.1 (Second Edition) is specified by the W3C in [W3C 2011] and is supported by all modern web browsers. The more recent SVG 2 specification [W3C 2018c], on the other hand, is only partially supported.

In contrast to raster image formats like Joint Photographic Experts Group (JPEG), Graphics Interchange Format (GIF), and Portable Network Graphics (PNG), vector graphics are resolution-independent and can be freely resized without any degradation in quality such as pixellation artifacts [Dürnegger et al. 2010, page 28]. For this reason, SVG is often used in web development for the automated creation of charts, diagrams, maps, and other visualisations based on dynamically-changing data. Since they are text files, SVG documents typically have much smaller file sizes than their raster equivalents and are also much more amenable to compression.

In addition to these aspects, a consequence of the text-based format is that single SVG elements, including both text fragments and graphical objects, are machine-readable without any need to apply image recognition techniques. This, in turn, not only provides the possibility to access any object within a graphic via the browser's Document Object Model (DOM), but it also means that, in general, any SVG content can be exposed via accessibility application programming interfaces (APIs) and processed by assistive technologies. Moreover, SVG elements can receive keyboard focus and can be annotated with WAI-ARIA attributes. While the latter two features are specified only in SVG Version 2 [W3C 2018c], which has not yet been given the status of an official recommendation, they are already widely supported by most modern browsers.

4.1.1 Native Accessibility of SVG

SVG provides a set of dedicated elements for basic geometric shapes, such as lines, circles, rectangles, ellipses, and polygons. More irregular shapes such as curves can be created using the more general `<path>` element. Text labels are specified as plain character strings within a `<text>` element. Several related objects can be logically grouped by nesting them within a `<g>` container. Moreover, it is possible to duplicate an object or a group of objects once defined by means of the `<use>` element. The information conveyed by these elements might provide sufficient semantics to derive a textual representation of certain simple drawings without applying image recognition techniques. However, for more complex graphics like charts and diagrams, this information cannot be regarded as semantic but only as syntactic and structural [Salameh et al. 2014, page 219].

Beyond the elements described above, SVG also includes two elements to specify additional text information, namely `<title>` and `<desc>`. The `<title>` element is intended to provide a name for its immediate ancestor, and is typically rendered visually by current browsers as a tooltip for the corresponding graphical representation [Migliorisi 2016; Schepers 2019]. The `<desc>` element can be used to add a detailed description to its immediate ancestor. Both `<title>` and `<desc>` can be applied by appending them as direct descendants to the root `<svg>` element, as well as to any shape or container element of the graphics document. Thus, they provide the possibility not only to add an overall name and description to a whole graphic, but also to create a hierarchical structure of separate names and descriptions corresponding to individual graphical objects within a graphics document. This, in turn, forms the basis for interactively exploring parts of an SVG document using screen readers and similar assistive technologies.

SVG 2 also considers keyboard navigation between objects of an SVG document. While interactive elements like links and buttons are regarded as focusable by default, graphical elements can be assigned the `tabindex` attribute known from HTML in order to make them keyboard focusable.

4.1.2 Scientific Proposals for Semantic Enrichment

Dürnegger et al. [2010] introduce a set of 15 guidelines for creating accessible SVG graphics, based on the general Web Content Accessibility Guidelines (WCAG) by the W3C and present a Java-based tool for the automatic evaluation of SVG documents against these guidelines. Their guidelines follow an inclusive design approach, considering all potential target groups, and can be summarised as follows:

- An SVG document must be well-formed and valid according to the official XML and SVG specifications. All elements and attributes must be used as intended by the SVG standard [W3C 2011].
- Presentational information must be separated from content using style sheets or the SVG `presentation` attribute.
- Sizes must be specified using relative instead of absolute units.
- An SVG document, as well as all its graphical objects and containers with semantic relevance, must have a `<title>` providing a name and a `<desc>` element with a more detailed description, where the texts should be concise and avoid redundant phrases like "The image shows. . . ".
- Raster images should be avoided whenever possible. If needed, they should be included within the SVG document using the `<image>` element and assigned a meaningful accessible name and description.
- Graphical objects should be grouped by containers and ordered in a hierarchy which represents the structure of the graphical elements, taking into account that the linear representation of objects by assistive technologies corresponds to the order in which they appear in the source code. Whenever possible, objects should be reused instead of redundantly creating them multiple times.
- Information must not be conveyed by colour alone.

- Low contrast, colour combinations problematic for users with certain visual impairments, and animations with high frequency blinking or flashing effects which could cause photo-epileptic seizures must be avoided.
- Interactive elements must be usable not only by mouse, but independently of the input device, taking into account the lower speed and accuracy of certain alternative input devices. All important elements must be large enough to be accessible for users with visual or motor impairments.

Numerous other approaches towards adding semantics to graphics can be found, most of which are based on SVG but do not make use of the <title> and <desc> elements. Patil [2007] proposes a general format for placing descriptive text into the metadata section of image files. A more concrete variant of this idea is presented by Kopecek and Oslejsek [2008]. In their the GATE system (introduced in Subsection 3.2.3), all the objects of an SVG document to be annotated are assigned id attributes. Titles and semantic categories are stored within the <metadata> element of the SVG document, associated with the graphical objects by their id attributes, and expressed by a system based on the Web Ontology Language (OWL). The annotation of raster images is supported, too, by creating an invisible SVG structure representing the annotated objects at their location within the image. At the time of writing, the ontology used for the annotations is not a comprehensive system, but should rather be regarded as an open and extensible standard which is meant to grow by crowd contributions. The resulting annotated graphics document can be explored by synthetic speech and sonification using dedicated software created by the authors.

The strategy of associating annotations with their corresponding graphical elements by identifiers is also applied in two other solutions. Sorge [2016] presents a JavaScript library called DIAGcess which converts chemical diagrams embedded in web pages as raster graphics into accessible SVG representations. The raster image is analysed using image recognition techniques, and the elements of the diagram are stored in Chemistry Markup Language (CML), an XML-based format for chemical diagrams. The CML representation is further analysed and enriched with semantic information. Finally, the diagram is reproduced as an SVG document whose elements are associated with their counterparts in the CML representation. The SVG diagram can be navigated by keyboard, causing the semantics and labels associated to the focused element to be sent to an ARIA live region, so that they are read aloud by a screen reader. According to the author, the system can be extended to other types of diagrams and other fields of science.

The Digital Accessibility Information System (DAISY) is an XML-based standard for structuring information semantically and hierarchically so that it can easily be browsed by users with disabilities. It is widely applied to digital audio books provided by libraries for blind people, enabling quick navigation of the recordings by chapters, sections, paragraphs, and pages. Gardner and Bulatov [2010] introduce a DAISY extension for graphical content, where the SVG elements are logically grouped by linking them with so-called layers of the DAISY representation. Textual labels and references to supplementary multimedia content can be associated with a graphical object nesting them in an <actions> element as descendant of a corresponding <a> element. Moreover, the author can define several views which represent different parts of the graphics at a chosen zoom level. This way, the recipient can easily print and explore different sections of the graphics in more detail. The DAISY SVG system is used in the IVEO software (described in Section 3.3), where IVEO Creator uses it as its output format and IVEO Viewer ignores <title> and <desc> elements if a DAISY SVG structure is detected.

A different approach is pursued by Salameh et al. [2014]. The system is primarily intended to improve the automated search for and categorisation of images. It expresses the semantics of an SVG document in a structure based on the Resource Description Framework (RDF). As a first step, the graphical objects are transfered to an RDF graph, whose triples only contain the information directly extracted from the SVG elements and their attributes. This RDF graph is then compared to the entries of a knowledge base using similarity computation. The best-matching result is returned and can be revised by the author or transcriber of the graphic. After final confirmation, the RDF structure contains a semantic representation of the graphical objects and is then added to the knowledge base. The knowledge base is intended to grow by crowd contributions and can be extended to express the semantics of specific domains.

4.1.3 ARIA Enhancements to SVG

SVG 2 [W3C 2018c] provides the possibility to use numerous standard ARIA attributes in the default namespace to annotate SVG elements. According to this candidate recommendation, as an alternative to the native SVG <title> and <desc> elements, the ARIA properties aria-label or aria-labelledby can be used to specify a name and the property aria-describedby to reference to a description. Moreover, the SVG Accessibility Task Force, a collaboration between the Web Accessibility Initiative (WAI) and the SVG Working Group of the W3C, published several documents to further improve the accessibility of graphics in general and of SVG in particular. These include the definition of additional special ARIA roles for graphics, as well as recommendations how browsers should expose native elements and ARIA attributes of graphics to accessibility APIs.

The WAI-ARIA Graphics Module [W3C 2018e] extends the core WAI-ARIA specification [W3C 2017a] by three additional roles. In particular, it defines the roles graphics-document for a whole structured, navigable graphical unit, graphics-object for elements of a graphical structure, and graphics-symbol for objects whose visual appearance is less important than its meaning (like icons). For elements with role graphics-symbol or img (defined in [W3C 2017a]), all descendants should be considered presentational, that is, excluded from the accessibility tree representation. This recommendation is not explicitly bound to SVG, so that it can theoretically also be applied to any other graphics format which supports textual markup.

4.2 ARIA Guidelines for User Agents

The SVG Accessibility API Mappings [W3C 2018d] are an extension of the Core Accessibility API Mappings [W3C 2017b]. The document, which is currently only a working draft, specifies which of the SVG elements defined in [W3C 2018c] should be included in the accessibility tree and how they should be mapped to keywords in the known accessibility APIs of different platforms. In general, it recommends that the majority of SVG elements be omitted in order to keep the accessibility tree representation of a graphics document as simple as possible. This applies to all elements which are explicitly hidden or not directly rendered (but not necessarily all the invisible ones). Moreover, each shape element and each <use>, <g>, <image>, <mesh>, and <foreignObject> element should be excluded, unless the author has assigned it semantic significance, that is, it has any kind of text content, text alternative, or appropriate ARIA role. All the elements which can receive focus, trigger events, or are interactive in any way must be included in the accessibility tree without exception.

With regard to the SVG <title> element, the SVG Accessibility API Mappings document states that browsers must expose its content to accessibility APIs as the accessible name of its direct ancestor. However, if the latter is assigned an aria-label or an aria-labelledby attribute, this must be used instead, where aria-labelledby should be preferred over aria-label if both exist. Similarly, the <desc> element must be exposed as an accessible description of its direct ancestor, but if an aria-describedby attribute is also specified, this has higher priority. If neither an aria-describedby attribute nor a <desc> element is given and the accessible name is derived from an ARIA attribute, a possible direct descendant <title> element can also be exposed as an accessible description. In the case of a <text> element, the contained text should be considered if none of the former information is provided. In the case of a link, the title attribute should be considered where present.

The Graphics Accessibility API Mappings [W3C 2018b] represents a modular extension of the Core Accessibility API Mappings [W3C 2017b] and specifies how the three roles defined in the WAI-ARIA Graphics Module [W3C 2018e] should be mapped to keywords in the known accessibility APIs of different platforms. Like [W3C 2018e], this recommendation is not explicitly bound to SVG, so that it can theoretically also be applied to any other graphics format which supports textual markup.

4.3 SVG Accessibility in Practice

Many researchers have experimented with creating accessible SVGs, including Watson [2014], Migliorisi [2016], Watson [2018] and Fisher [2019]. All the authors recommend using SVG to create accessible graphics and come to the conclusion that inline SVG embedded within a web document provides the highest level of accessibility. By contrast, the content of external SVG files linked by an <img>, <object>, <iframe>, or <embed> element is not directly inserted into the DOM and, for this reason, is not entirely included within the accessibility tree.

Watson [2018] states that the content of both the aria-label attribute and the <title> element are exposed as accessible names by all the browsers tested. The <title> element is also used by all the screen readers under test [Fisher 2019]. In [Watson 2014; Fisher 2019], the aria-labelledby attribute is shown to be well supported for the computation of the accessible name by all browsers and screen readers. The content of the <desc> element, by contrast, is recognised only by Microsoft Internet Explorer with JAWS and Apple Safari with VoiceOver (on both macOS and iOS) [Fisher 2019]. According to the latter, the aria-describedby attribute is not widely supported either, namely by the platforms just mentioned as well as Google Chrome on Android with TalkBack.

In order to provide an accessible name and description compatible with as many browsers and screen readers as possible, Watson [2014] and Fisher [2019] recommend using the <title> element to specify an accessible name, <desc> for an accessible description, and associating both elements with the ancestor by referencing them with an aria-labelledby attribute. Unfortunately, this association causes the accessible name and/or the description to be read twice in some configurations. Moreover, using aria-labelledby for an accessible description is not semantically correct. However, this solution is regarded by the authors as the most reliable one at the moment. Setting the attribute to the id of the <title> element followed by that of the <desc> element produces a concatenation of both id strings delimited by a space character, ensuring that screen readers (at least, all those considered) present both the accessible name and the description in sequence.

4.4 Scientific Proposals for Accessible SVG Charts

As in the case of general purpose graphics (see Subsection 4.1.2), several scientific proposals for the semantic enhancement of digital charts have been published. Most of these, however, consider neither the native SVG elements for descriptive text nor WAI-ARIA attributes. For instance, Fredj and Duce [2006] argue that SVG 1 cannot store enough information on the structure and the semantics of a chart and, for this reason, present a software system for storing charts whose output is not an SVG document but two representations in more abstract, dedicated markup languages. Different visual and non-visual versions, such as can be SVG documents and textual descriptions, derived from these abstract representations by means of Extensible Stylesheet Language (XSL) transformations.

Similarly, the iGraph-Lite system proposed by Ferres et al. [2007], Ferres et al. [2013] and Ferres [2015] described in Subsection 3.2.1 stores the properties of a chart in an OWL structure detached from the graphical representation. The BrailleR extension by Fitzpatrick et al. [2017] and Godfrey et al. [2018] uses the DIAGcess library [Sorge 2016]. Statistical charts created in SVG format are enriched with semantic information on the chart elements expressed in a separate XML structure, whose elements are associated with SVG chart elements by referencing their id attributes.

By contrast, the SVG-Plott software [Engel et al. 2019; Harlan et al. 2019] described in Section 3.3 appends titles and descriptions to the SVG elements of the generated charts if multimodal output is chosen. Interactive regions are added so that speech output, if available, is triggered when touching elements of the chart on a touch-sensitive tactile device. In addition to the SVG chart, a separate SVG file containing the legend and an HTML file containing a short summary and listings of the data points are generated.

4.5 WAI-ARIA-Based Systems for Charts

At the time of writing this thesis, no dedicated WAI-ARIA taxonomy for charts has been standardised, nor has any de facto standard for enriching chart elements with ARIA roles and properties evolved. However, various approaches based on W3C standards have been published informally, that is, as part of web applications, charting libraries, wikis, or blog postings. These include comprehensive systems using accessible names and descriptions, ARIA roles and properties, and sometimes SVG class names to express the meaning of chart objects. In the following subsections, several of these proposals will be described in detail.

For most of the systems discussed here, the taxonomy of semantic keywords will be depicted as tables. In the case of ARIA roles and class names, the tables have the following structure: the first column, *Role*, names an ARIA role used in this system. If class names are also relevant in the respective system, these will be given in the second column, *Class*. The *Element* column states the SVG element the role and/or class name is assigned to. The *Ancestor* column states which chart object contains the element with the respective role and/or class name. The *Content* column names the chart objects or data type which the respective element includes as descendants. The final column, *Meaning*, describes the semantics of the role and/or class name.

Both *Ancestor* and *Content* describe the relations of chart objects within the hierarchical SVG structure of the system. This description is, however, not comprehensive. It includes only those object(s) which are assumed relevant for accessibility purposes. In other words, if a certain object contains objects without any textual information (for instance, `<g>` or `<path>` elements without any ARIA role or property), these objects will be omitted from the description of the relationship. Ancestors and descendants need not be direct; for instance, if an object contains a `<g>` element without any ARIA role or property which, in turn, contains another object with accessible information, the intermediate `<g>` element will not be mentioned.

Tables describing the usage of ARIA properties have the following structure: the first column, *Property*, names an ARIA property applied in the respective system. The second column, *Object*, states the chart object the property is attached to. The *Value* column lists the possible attribute values which can be assigned to this property. The final column, *Meaning*, describes the information the respective ARIA property conveys.

In addition to the terminology defined in the introductory chapter of this thesis, the following terms will be used throughout this section:

- *data object*: A dataset, data series, data group, data point, or statistical summary.
- *scale*: One dimension of a dataset, typically expressed as an axis or legend.
- *shape*: A graphical SVG element like `<circle>`, `<line>`, `<rect>`, or `<path>`, or a `<use>` element referencing such a graphical SVG element.
- *standard ARIA role/property*: An ARIA role or property defined in the core WAI-ARIA specification for web documents [W3C 2017a].

Moreover, with regard to the representation of ARIA roles and properties, the following notation will be used:

- Whenever the content of an object is stated as *text*, this means plain text and *not* the `<text>` element unless stated otherwise.
- A hyphen (-) means that the value has no importance or is not applicable in this system.
- A question mark (?) means that the ancestor or content has not been precisely specified in this system.
- Whenever the content of elements or values for properties are stated within quotation marks (), they are meant verbatim. Otherwise, they are descriptive.

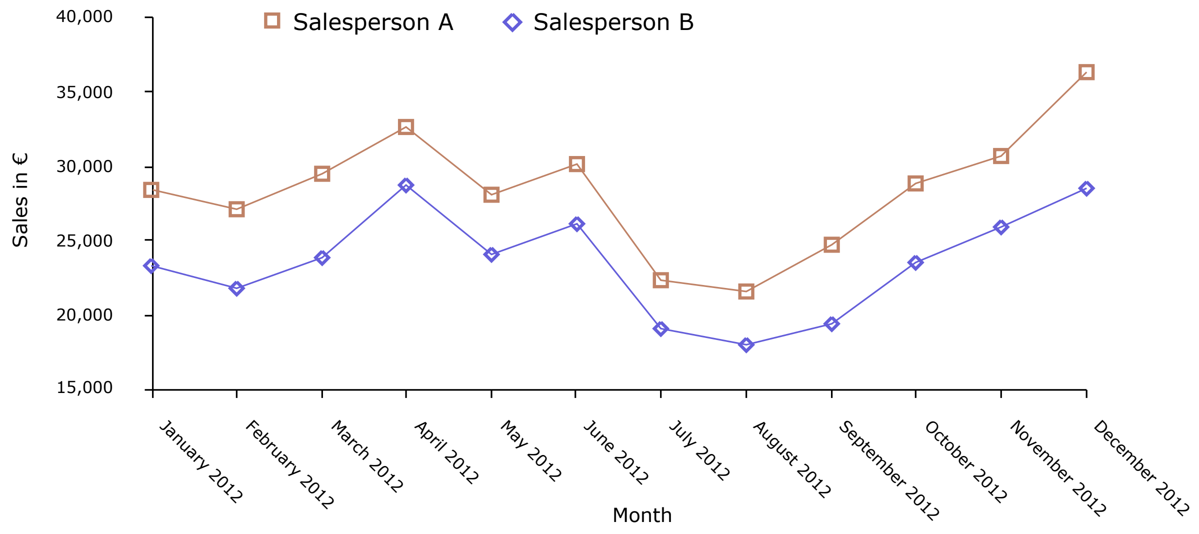


Figure 4.1: Basic sample line chart with two data series. It corresponds to the SVG code excerpt shown in Listing 4.1. [Created and used with kind permission by Keith Andrews, ISDS, Graz University of Technology.]

For each of the systems presented in the following subsections, its application will be illustrated by an excerpt of sample SVG code. The chart taken for this purpose is shown in Figure 4.1. The initial version of the corresponding SVG source code excerpt is given in Listing 4.1. This version does not include any accessibility-related annotations apart from a <title> and a <desc> element appended to the chart root. In all the subsequent SVG code listings, those attributes which fulfil accessibility purposes were deliberately placed first within the element in order to improve readability. It should be noted that the values of all the id attributes were chosen arbitrarily and do not form a part of any of the described systems. The same applies to the class names, unless stated otherwise.

The systems presented in this section can be grouped into two groups. Firstly, those systems will be described which have been explicitly published as such and, therefore, are assumed to be fully specified. Some of these systems do not define all the roles and/or properties necessary to completely express the semantics of all three main chart types covered in this thesis (line, bar, and pie charts). These missing roles and/or properties are discussed at the end of the respective subsection under the term “necessary extensions”. Roles and/or properties for other chart types are discussed under future work in Section 8.1.

Secondly, systems which are applied by some of the charting libraries introduced in Section 3.5 will be summarised. Since the information about the roles, class names, and properties used by these systems was derived from the SVG source code of demonstration charts provided on the web site of the respective libraries, the resulting taxonomies are derived from observation and cannot be considered exhaustive or complete. For this reason, a discussion of missing roles, class names, and properties is not considered appropriate in these cases.

```
<svg version="1.1" xml:lang="en" lang="en" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink" viewBox="0 190 750 346">

<g>

<title id="title">Line Chart of Sales</title>

<desc id="desc">
  Chart of sales for 12 months in year 2012 for Salespersons A and B.
</desc>


<!-- Y Axis -->

<g>

<line x1="98" y1="443" x2="98" y2="212" class="chart-line" />
<text id="y-title" transform="matrix(0 -1 1 0 20 355)"
  class="chart-title">Sales in €</text>

<g>
<text transform="matrix(1 0 0 1 38 215)" class="chart-label">40,000</text>
<line x1="98" y1="212" x2="93" y2="212" class="chart-line" />
</g>

<g>
<text transform="matrix(1 0 0 1 38 262)" class="chart-label">35,000</text>
<line x1="98" y1="258" x2="93" y2="258" class="chart-line" />
</g>

...
</g>


<!-- X Axis -->

<g>

<line x1="98" y1="443" x2="677" y2="443" class="chart-line" />
<text id="x-title" transform="matrix(1 0 0 1 383 525)"
  class="chart-title">Month</text>

<g>
<text id="x-2012-01" transform="matrix(0.7 0.7 -0.7 0.7 102 466)"
  class="chart-label">January 2012</text>
<line x1="98" y1="443" x2="98" y2="448" class="chart-line" />
</g>

<g>
<text id="x-2012-02" transform="matrix(0.7 0.7 -0.7 0.7 155 466)"
  class="chart-label">February 2012</text>
<line x1="150" y1="443" x2="150" y2="448" class="chart-line" />
</g>

...
</g>
```

Listing 4.1: Basic sample SVG source code of the chart shown in Figure 4.1. Apart from a <title> and a <desc> element as descendants of the chart root, it contains no accessibility markup. The class names are arbitrary and used only for styling.

```


<!-- Data Points A -->

<g id="data-a">

<polyline points="
   98,319 150,331
  150,331 203,309
  203,309 255,280
  255,280 308,322
  308,322 361,303
  361,303 413,375
  413,375 466,382
  466,382 519,353
  519,353 571,315
  571,315 624,298
  624,298 677,246
  " class="chart-lineA" />

<g>
<rect x="93" y="315" class="marker-square" />
</g>

<g>
<rect x="146" y="327" class="marker-square" />
</g>

...

</g>


<!-- Legend -->

<g id="legend">

<g id="legend-a">
<rect x="168" y="210" class="marker-square" />
<text id="legend-text-a" x="185" y="220"
  font-family="Verdana" font-size="14">Salesperson A</text>
</g>

...

</g>

</g>
</svg>
```

Listing 4.1 (cont.)**:** Basic sample SVG source code of the chart shown in Figure 4.1.

Role	Element	Ancestor	Content	Meaning
chart	<svg> / <g>	- / <svg>	all objects	chart root element
heading*	<title> / <text>	chart / scale	text	chart/scale title
chartarea	<rect>	chart	-	chart background (used to detect visual hight and width)
xaxis	<g>	chart	axis title and labels	x-axis
yaxis	<g>	chart	axis title and labels	y-axis
axislabel	<title> / <text>	axis	text	label of an axis item
legend	<g>	chart	legend title and items	legend
legenditem	<g>	legend	<text> element	legend item
dataset	<g>	chart	data points	dataset
datagroup	?	?	?	collection of related data points
datapoint	<g> / shape	dataset	data value	data point
datavalue	<title>	data point	text	y-value of a data point

Table 4.1: ARIA roles defined for the Describler software. An asterisk (*) indicates that this role is defined in an official ARIA specification.

Property	Object	Value	Meaning
aria-charttype	chart root	bar / pie / line	type of a chart
aria-axistype	axis	category	states if an axis has discrete values
aria-valuemin*	numerical axis	number	minimum value
aria-valuemax*	numerical axis	number	maximum value
aria-labelledby*	data value	id of x-axis label / legend item	references the name (x-axis label / legend item) associated with a data point

Table 4.2: ARIA properties defined for the Describler software. An asterisk (*) indicates that this property is defined in an official ARIA specification.

4.5.1 Describler

The Describler software presented in Section 3.6 introduces a custom taxonomy of non-standard ARIA roles and properties for charts and chart objects [Schepers 2015a; Schepers 2017]. This taxonomy is used within both the source code of the application and the sample SVG files provided along with the software. All the chart objects in these sample graphics include the tabindex attribute set to `0`, so that they can be navigated using the `Tab` key. The ARIA roles defined in this system are presented in Table 4.1; the defined ARIA properties can be found in Table 4.2. The source code of a sample SVG document annotated according to this system is given in Listing 4.2.

A necessary extension to the Describler taxonomy would be to define roles and titles for data series.

```
<svg version="1.1" xml:lang="en" lang="en" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink" viewBox="0 190 750 346">

<g role="chart" aria-charttype="line" tabindex="0">

<title role="heading" id="title">Line Chart of Sales</title>

<desc id="desc">
  Chart of sales for 12 months in year 2012 for Salespersons A and B.
</desc>


<!-- Y Axis -->

<g role="yaxis" aria-valuemin="15000" aria-valuemax="40000" tabindex="0">

<line x1="98" y1="443" x2="98" y2="212" class="chart-line" />
<text role="heading" id="y-title" transform="matrix(0 -1 1 0 20 355)"
  class="chart-title">Sales in €</text>

<g>
<text role="axislabel" transform="matrix(1 0 0 1 38 215)"
  class="chart-label">40,000</text>
<line x1="98" y1="212" x2="93" y2="212" class="chart-line" />
</g>

<g>
<text role="axislabel" transform="matrix(1 0 0 1 38 262)"
  class="chart-label">35,000</text>
<line x1="98" y1="258" x2="93" y2="258" class="chart-line" />
</g>

...
</g>


<!-- X Axis -->

<g role="xaxis" aria-axistype="category" tabindex="0">

<line x1="98" y1="443" x2="677" y2="443" class="chart-line" />
<text role="heading" id="x-title" transform="matrix(1 0 0 1 383 525)"
  class="chart-title">Month</text>

<g>
<text role="axislabel" id="x-2012-01" class="chart-label"
  transform="matrix(0.7 0.7 -0.7 0.7 102 466)">January 2012</text>
<line x1="98" y1="443" x2="98" y2="448" class="chart-line" />
</g>

<g>
<text role="axislabel" id="x-2012-02" class="chart-label"
  transform="matrix(0.7 0.7 -0.7 0.7 155 466)">February 2012</text>
<line x1="150" y1="443" x2="150" y2="448" class="chart-line" />
</g>
```

Listing 4.2: Sample SVG code annotated with ARIA roles and properties according to the system introduced by the Describler software. Certain chart objects are assigned the attribute `tabindex=0` for `Tab` navigation. The class names are arbitrary and used only for styling.

```

...
</g>


<!-- Data Points A -->

<g role="dataset" id="data-a">

<polyline points="
   98,319 150,331
  150,331 203,309
  203,309 255,280
  255,280 308,322
  308,322 361,303
  361,303 413,375
  413,375 466,382
  466,382 519,353
  519,353 571,315
  571,315 624,298
  624,298 677,246
  " class="chart-lineA" />

<g role="datapoint" tabindex="0">
<title role="datavalue" aria-labelledby="x-2012-01">28366</title>
<rect x="93" y="315" class="marker-square" />
</g>

<g role="datapoint" tabindex="0">
<title role="datavalue" aria-labelledby="x-2012-02">27050</title>
<rect x="146" y="327" class="marker-square" />
</g>

...
</g>


<!-- Legend -->

<g role="legend" tabindex="0" id="legend">

<g role="legenditem" tabindex="0" id="legend-a">
<rect x="168" y="210" class="marker-square" />
<text id="legend-text-a" x="185" y="220"
  font-family="Verdana" font-size="14">Salesperson A</text>
</g>

...
</g>

</g>
</svg>
```

Listing 4.2 (cont.)**:** Sample SVG code annotated with the ARIA roles and properties according to the system introduced by the Describler software.

4.5.2 WAI-ARIA Graphics Roles

Schepers [2019] recommends using the three roles defined in the WAI-ARIA Graphics Module [W3C 2018e] (see Subsection 4.1.3) in the following manner:

- graphics-document: for the chart root element.
- graphics-object: for objects which contain sub-components, such as axes.
- graphics-symbol: for atomic objects, such as data points.

According to this proposal, a more detailed classification of the objects should then be made using the ARIA property aria-roledescription. As an example, the author suggests aria-roledescription=bar for a data point in a bar chart. Moreover, it is recommended that all objects of interest be given a tabindex of 0, so that they can be navigated to using the Tab key, and that each name and value of a data point be exposed as an accessible name. The latter should be achieved either by grouping a <text> or a <title> element with the data point's element(s) or using one of the properties aria-label and aria-labelledby. For the widest browser and screen reader compatibility, the author recommends using a combination of both a <text> or <title> element and the property aria-labelledby pointing to this element. No complete example is provide for the application of this system. A possible solution is given in Listing 4.3.

Necessary extensions to this scheme include:

- Values for aria-roledescription to indicate charts and other chart objects.
- Values for aria-roledescription to indicate data points of other chart types, such as line and pie charts.
- Properties for expressing the type of a chart or an axis.

```
<svg version="1.1" xml:lang="en" lang="en" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink" viewBox="0 190 750 346">

<g role="graphics-document" aria-roledescription="line chart">

<title id="title">Line Chart of Sales</title>

<desc id="desc">
  Chart of sales for 12 months in year 2012 for Salespersons A and B.
</desc>


<!-- Y Axis -->

<g role="graphics-object" aria-roledescription="y-axis" tabindex="0">

<line x1="98" y1="443" x2="98" y2="212" class="chart-line" />
<text id="y-title" transform="matrix(0 -1 1 0 20 355)"
  class="chart-title">Sales in €</text>

<g role="graphics-symbol" aria-roledescription="axis item">
<text transform="matrix(1 0 0 1 38 215)" class="chart-label">40,000</text>
<line x1="98" y1="212" x2="93" y2="212" class="chart-line" />
</g>

<g role="graphics-symbol" aria-roledescription="axis item">
<text transform="matrix(1 0 0 1 38 262)" class="chart-label">35,000</text>
<line x1="98" y1="258" x2="93" y2="258" class="chart-line" />
</g>

...
</g>


<!-- X Axis -->

<g role="graphics-object" aria-roledescription="x-axis" tabindex="0">

<line x1="98" y1="443" x2="677" y2="443" class="chart-line" />
<text id="x-title" transform="matrix(1 0 0 1 383 525)"
  class="chart-title">Month</text>

<g role="graphics-symbol" aria-roledescription="axis item">
<text id="x-2012-01" transform="matrix(0.7 0.7 -0.7 0.7 102 466)"
  class="chart-label">January 2012</text>
<line x1="98" y1="443" x2="98" y2="448" class="chart-line" />
</g>

<g role="graphics-symbol" aria-roledescription="axis item">
<text id="x-2012-02" transform="matrix(0.7 0.7 -0.7 0.7 155 466)"
  class="chart-label">February 2012</text>
<line x1="150" y1="443" x2="150" y2="448" class="chart-line" />
</g>

...
</g>
```

Listing 4.3: Sample SVG code annotated with the roles defined in the WAI-ARIA Graphics Module in conjunction with the property aria-roledescription. Objects of interest are assigned a tabindex=0 for Tab navigation. The class names are arbitrary and used only for styling.

```


<!-- Data Points A -->

<g role="graphics-object" aria-roledescription="line" tabindex="0" id="data-a">

<polyline points="
   98,319 150,331
  150,331 203,309
  203,309 255,280
  255,280 308,322
  308,322 361,303
  361,303 413,375
  413,375 466,382
  466,382 519,353
  519,353 571,315
  571,315 624,298
  624,298 677,246
  " class="chart-lineA" />

<g role="graphics-symbol" aria-roledescription="data point"
  tabindex="0" aria-labelledby="x-2012-01 datapoint-a0">
<title id="datapoint-a0">28366</title>
<rect x="93" y="315" class="marker-square" />
</g>

<g role="graphics-symbol" aria-roledescription="data point"
  tabindex="0" aria-labelledby="x-2012-02 datapoint-a1">
<title id="datapoint-a1">27050</title>
<rect x="146" y="327" class="marker-square" />
</g>

...
</g>


<!-- Legend -->

<g role="graphics-object" aria-roledescription="legend" tabindex="0"
  id="legend">

<g role="graphics-symbol" aria-roledescription="legend item" id="legend-a">
<rect x="168" y="210" class="marker-square" />
<text id="legend-text-a" x="185" y="220"
  font-family="Verdana" font-size="14">Salesperson A</text>
</g>

...
</g>

</g>
</svg>
```

Listing 4.3 (cont.)**:** Sample SVG code annotated with the roles defined in the WAI-ARIA Graphics Module in conjunction with the property aria-roledescription.

Role	Ancestor	Content	Meaning
graphics-datachart	<svg>	all objects	chart root element
graphics-axis	chart	axis items	axis
graphics-legend	chart	legend items	legend
graphics-tick	scale	label	item of a continuous (numerical) scale
graphics-category	scale	label	item of a discrete (categorical / ordinal) scale
graphics-note**	chart / data object	text	annotation with additional context / information for the ancestor
graphics-datagroup	?	?	collection of related data points
graphics-dataline	chart	data points	data series
graphics-dataunit	data series	?	data point
graphics-dataregion	?	?	complex 2-dimensional feature / contour
graphics-summarydata	chart / data group	?	statistical object, such as trend / mean line

Table 4.3: Non-abstract ARIA roles for charts of tabular data defined in the W3C proposal [W3C 2015b]. The proposal does not state which particular SVG element(s) to annotate with a certain role. An asterisk (*) indicates that this role is defined in an official ARIA specification. Roles marked with two asterisks (**) are labelled as an "issue" in the wiki and are still the subject of discussion.

4.5.3 W3C Proposal

W3C [2015b] proposes a system of ARIA roles and properties for various types of visualisations, such as charts, network diagrams, and maps. This system is a hierarchical taxonomy partly consisting of abstract and superclass roles. Tables 4.3 and 4.4 present the non-abstract subset applicable for charts of tabular data. It should be noted that this proposal has no official status yet and is only available in the form of a wiki.

No complete code example is provided for the application of this system. A possible solution can be seen in Listing 4.4. Since no values for the property aria-roledescription are specified with regard to line charts, the values in this example are only based on assumptions.

A necessary extension to this scheme would be a specified way to express the variable of an axis.

Property	Object	Value	Meaning
aria-roledescription*	any	text	type of chart/object (e.g. piechart, slice)
aria-orientation*	axis	horizontal / vertical / depth / other	visual orientation
aria-dataunit	scale	text	unit of numerical values
aria-datatype**	scale	token (examples: category / ordinal / number)	scale type
aria-valuemin*,**	numerical scale	number	minimum value
aria-valuemax*,**	numerical scale	number	maximum value
aria-valuenow*,**	scale item	string / number	machine-readable value of label
aria-valuetext*	scale item	text	human-readable text of label, defaults to accessible name
aria-label*	any	text	title of an object
aria-labelledby*	any	id	reference to title of object
aria-describedby*	any	id	reference to description of object
aria-live*	any	true	indicates data which are updated
aria-datascales	data object	id list	associates a data object with scales
aria-datavariables	data object	text list	human-readable names of scales associated with data object, defaults to scale titles
aria-datavalues	data object	text / number list	names and values of data object in order of its associated scales (example: x-name,y-value)
aria-datavaluearray	data group / series	array of type aria-datavalues	values of all data points associated with data group or data series
aria-dataproperty**	data object	property / attribute names	indicates which style property/attribute are modified for each data variable
aria-haspopup*	any	?	for objects with author-generated pop-up
aria-owns*	any	id list	associates referenced objects with host element (such as data points with a data group)
aria-posinset*	data point / group	number	position within underlying dataset (if chart only shows portion at a time)
aria-setsize*	data point / group	number	size of complete dataset (if chart only shows portion at a time)

Table 4.4: ARIA properties for charts of tabular data defined in the W3C proposal [W3C 2015b]. An asterisk (*) indicates that this property is defined in an official ARIA specification. Properties marked with two asterisks (**) are labelled as an "issue" in the wiki and are still the subject of discussion.

```
<svg version="1.1" xml:lang="en" lang="en" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink" viewBox="0 190 750 346">

<g role="graphics-datachart" aria-roledescription="line chart"
  aria-labelledby="title" aria-describedby="desc">

<title id="title">Line Chart of Sales</title>

<desc id="desc">
  Chart of sales for 12 months in year 2012 for Salespersons A and B.
</desc>

<!-- Y Axis -->

<g role="graphics-axis" aria-valuemin="15000" aria-valuemax="40000"
  aria-orientation="vertical" aria-dataunit="€" aria-datatype="number"
  aria-roledescription="y-axis" aria-labelledby="y-title" id="yscale">

<line x1="98" y1="443" x2="98" y2="212" class="chart-line" />
<text id="y-title" transform="matrix(0 -1 1 0 20 355)"
  class="chart-title">Sales in €</text>

<g role="graphics-tick" aria-valuenow="40000">
<text transform="matrix(1 0 0 1 38 215)" class="chart-label">40,000</text>
<line x1="98" y1="212" x2="93" y2="212" class="chart-line" />
</g>

<g role="graphics-tick" aria-valuenow="35000">
<text transform="matrix(1 0 0 1 38 262)" class="chart-label">35,000</text>
<line x1="98" y1="258" x2="93" y2="258" class="chart-line" />
</g>

...
</g>

<!-- X Axis -->

<g role="graphics-axis" aria-orientation="horizontal" aria-datatype="category"
  aria-roledescription="x-axis" aria-labelledby="x-title" id="xscale">

<line x1="98" y1="443" x2="677" y2="443" class="chart-line" />
<text transform="matrix(1 0 0 1 383 525)" id="x-title"
  class="chart-title">Month</text>

<g role="graphics-category" aria-valuenow="2012-01">
<text id="x-2012-01" transform="matrix(0.7 0.7 -0.7 0.7 102 466)"
  class="chart-label">January 2012</text>
<line x1="98" y1="443" x2="98" y2="448" class="chart-line" />
</g>

<g role="graphics-category" aria-valuenow="2012-02">
<text id="x-2012-02" transform="matrix(0.7 0.7 -0.7 0.7 155 466)"
  class="chart-label">February 2012</text>
<line x1="150" y1="443" x2="150" y2="448" class="chart-line" />
</g>

...
</g>
```

Listing 4.4: Sample SVG code annotated with ARIA roles and properties for charts according to the system proposed by the W3C. The values chosen for aria-roledescription are not explicitly part of this proposal. The class names are arbitrary and used only for styling.

```


<!-- Data Points A -->

<g role="graphics-dataline" aria-roledescription="line"
  aria-datascales="legend" aria-datavariables="Salespersons"
  aria-labelledby="legend-text-a" aria-datavaluearray="[
  'January 2012' 28366, 'February 2012' 27050, ...]" id="data-a">

<polyline points="
   98,319 150,331
  150,331 203,309
  203,309 255,280
  255,280 308,322
  308,322 361,303
  361,303 413,375
  413,375 466,382
  466,382 519,353
  519,353 571,315
  571,315 624,298
  624,298 677,246
  " class="chart-lineA" />

<g role="graphics-dataunit" aria-datascales="xscale yscale"
  aria-datavariables="x y" aria-datavalues="'January 2012' 28366">
<rect x="93" y="315" class="marker-square" />
</g>

<g role="graphics-dataunit" aria-datascales="xscale yscale"
  aria-datavariables="x y" aria-datavalues="'February 2012' 27050">
<rect x="146" y="327" class="marker-square" />
</g>

...
</g>


<!-- Legend -->

<g role="graphics-legend" aria-datatype="category" id="legend">

<g role="graphics-category" id="legend-a">
<rect x="168" y="210" class="marker-square" />
<text id="legend-text-a" x="185" y="220"
  font-family="Verdana" font-size="14">Salesperson A</text>
</g>

...
</g>

</g>
</svg>
```

Listing 4.4 (cont.)**:** Sample SVG code annotated with the ARIA roles and properties for charts according to the system proposed by the W3C.

Role	Element	Ancestor	Content	Meaning
table	<g>	<svg>	all objects	chart root element
row	<g>	chart	data point names / values	group of data point names (examples: x-axis / legend) / data series
columnheader	<text>	group of data point names	text	names group title (examples: x-axis / legend title) / data point name
rowheader	<g>	data series	visual data object	data series title
cell	<g>	data series	visual data object	data point
img	shape	data series title / data point	<title>	visual data object (example: <line>)

Table 4.5: Standard ARIA roles for tables, used to mark up line charts, as proposed by Watson [2017].

4.5.4 SVG Pseudo-Table

Watson [2017] recommends using the standard ARIA roles for tables defined in the core WAI-ARIA specification [W3C 2017a] to mark up the data points included in line charts. A variant of the system is applied by the charting library D3plus [D3plus 2019]. The usage of the roles is presented in Table 4.5. ARIA properties are not used in the original proposal.

Note that the elements with role rowheader (that is, the representations of the data series titles) contain a shape element with role img which, in turn, contains a <title> element with the actual title of the data series. Similarly, the elements with role cell (the data points) contain a shape element with role img which, in turn, contains a <title> element with the value of the data point. This implementation is chosen because on most browsers, screen readers do not display the content of the <title> element if it is not a direct descendant of a visible element [Watson 2017].

Necessary extensions to this scheme include roles to unambiguously indicate axes, axis titles, and labels.

```
<svg version="1.1" xml:lang="en" lang="en" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink" viewBox="0 190 750 346">

<g role="table">

<title>Line Chart of Sales</title>

<desc>
  Chart of sales for 12 months in year 2012 for Salespersons A and B.
</desc>


<!-- Y Axis -->

<g>

<line x1="98" y1="443" x2="98" y2="212" class="chart-line" />
<text id="y-title" transform="matrix(0 -1 1 0 20 355)"
  class="chart-title">Sales in €</text>

<g>
<text transform="matrix(1 0 0 1 38 215)" class="chart-label">40,000</text>
<line x1="98" y1="212" x2="93" y2="212" class="chart-line" />
</g>

<g>
<text transform="matrix(1 0 0 1 38 262)" class="chart-label">35,000</text>
<line x1="98" y1="258" x2="93" y2="258" class="chart-line" />
</g>

...
</g>


<!-- X Axis -->

<g role="row">

<line x1="98" y1="443" x2="677" y2="443" class="chart-line" />
<text role="columnheader" id="x-title" transform="matrix(1 0 0 1 383 525)"
  class="chart-title">Month</text>

<g>
<text role="columnheader" id="x-2012-01" class="chart-label"
  transform="matrix(0.7 0.7 -0.7 0.7 102 466)">January 2012</text>
<line x1="98" y1="443" x2="98" y2="448" class="chart-line" />
</g>

<g>
<text role="columnheader" id="x-2012-02" class="chart-label"
  transform="matrix(0.7 0.7 -0.7 0.7 155 466)">February 2012</text>
<line x1="150" y1="443" x2="150" y2="448" class="chart-line" />
</g>

...
</g>
```

Listing 4.5: Sample SVG code annotated with the standard ARIA roles for tables, as proposed by Watson [2017]. Since a data series title needs to be a descendant element of the corresponding data series in this system, it has been shifted accordingly in this example. The class names are arbitrary and used only for styling.

```


<!-- Data Points A -->

<g role="row" id="data-a">

<g role="rowheader" id="legend-a">
<rect x="168" y="210" class="marker-square" />
<text id="legend-text-a" x="185" y="220"
  font-family="Verdana" font-size="14">Salesperson A</text>
</g>

<polyline points="
   98,319 150,331
  150,331 203,309
  203,309 255,280
  255,280 308,322
  308,322 361,303
  361,303 413,375
  413,375 466,382
  466,382 519,353
  519,353 571,315
  571,315 624,298
  624,298 677,246
  " class="chart-lineA" />

<g role="cell">
<rect role="img" x="93" y="315" class="marker-square">
<title id="datapoint-a0">28366</title>
</rect>
</g>

<g role="cell">
<rect role="img" x="146" y="327" class="marker-square">
<title id="datapoint-a1">27050</title>
</rect>
</g>

...
</g>

</g>
</svg>
```

Listing 4.5 (cont.)**:** Sample SVG code annotated with the standard ARIA roles for tables, as proposed by Watson [2017].

Role	Element	Ancestor	Content	Meaning
group	<svg>	-	all objects	chart root element
list	<g>	chart	data points	data series
listitem	<g>	data series	<text> element with data point values in form `NAME-VALUE`; e.g. Jaws-44\%	data point

Table 4.6: Standard ARIA roles for lists, used for bar charts as proposed by Migliorisi [2016], Migliorisi [2019] and Kopacz [2019].

4.5.5 SVG Pseudo-List

Migliorisi [2016], Migliorisi [2019] and Kopacz [2019] propose to apply the standard ARIA roles for lists to indicate data series and data points. The usage of these roles is presented in Table 4.6.

The standard ARIA properties aria-labelledby and aria-describedby are attached to the root <svg> element, pointing to <text> elements which contain a chart title and description, respectively. Moreover, the <g> element representing a data series is assigned the aria-label property with the title of the data series. A sample SVG code excerpt is presented in Listing 4.6.

Necessary extensions for this scheme include roles for axes, axis titles, and axis labels.

```
<svg role="group" aria-labelledby="title" aria-describedby="desc"
  version="1.1" xml:lang="en" lang="en" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink" viewBox="0 190 750 346">

<title id="title">Line Chart of Sales</title>

<desc id="desc">
  Chart of sales for 12 months in year 2012 for Salespersons A and B.
</desc>


<!-- Y Axis -->

<g>

<line x1="98" y1="443" x2="98" y2="212" class="chart-line" />
<text id="y-title" transform="matrix(0 -1 1 0 20 355)"
  class="chart-title">Sales in €</text>

<g>
<text transform="matrix(1 0 0 1 38 215)" class="chart-label">40,000</text>
<line x1="98" y1="212" x2="93" y2="212" class="chart-line" />
</g>

<g>
<text transform="matrix(1 0 0 1 38 262)" class="chart-label">35,000</text>
<line x1="98" y1="258" x2="93" y2="258" class="chart-line" />
</g>

...
</g>


<!-- X Axis -->

<g>

<line x1="98" y1="443" x2="677" y2="443" class="chart-line" />
<text id="x-title" transform="matrix(1 0 0 1 383 525)"
  class="chart-title">Month</text>

<g>
<text id="x-2012-01" transform="matrix(0.7 0.7 -0.7 0.7 102 466)"
  class="chart-label">January 2012</text>
<line x1="98" y1="443" x2="98" y2="448" class="chart-line" />
</g>

<g>
<text id="x-2012-02" transform="matrix(0.7 0.7 -0.7 0.7 155 466)"
  class="chart-label">February 2012</text>
<line x1="150" y1="443" x2="150" y2="448" class="chart-line" />
</g>

...
</g>
```

Listing 4.6: Sample SVG code annotated with the standard ARIA roles for lists and the properties aria-label, aria-labelledby, and aria-describedby, as proposed by Migliorisi [2016], Migliorisi [2019] and Kopacz [2019]. The class names are arbitrary and used only for styling.

```


<!-- Data Points A -->

<g role="list" aria-label="Salesperson A" id="data-a">

<polyline points="
   98,319 150,331
  150,331 203,309
  203,309 255,280
  255,280 308,322
  308,322 361,303
  361,303 413,375
  413,375 466,382
  466,382 519,353
  519,353 571,315
  571,315 624,298
  624,298 677,246
  " class="chart-lineA" />

<g role="listitem">
<rect x="93" y="315" class="marker-square" />
<text id="datapoint-a0" x="97" y="318" text-ancor="middle">
  January 2012 - 28366</text>
</g>

<g role="listitem">
<rect x="146" y="327" class="marker-square" />
<text id="datapoint-a1" x="150" y="330" text-ancor="middle">
  February 2012 - 27050</text>
</g>

...
</g>


<!-- Legend -->

<g id="legend">

<g id="legend-a">
<rect x="168" y="210" class="marker-square" />
<text id="legend-text-a" x="185" y="220"
  font-family="Verdana" font-size="14">Salesperson A</text>
</g>

...
</g>

</svg>
```

Listing 4.6 (cont.)**:** Sample SVG code annotated with the standard ARIA roles for lists and the properties aria-label, aria-labelledby, and aria-describedby, as proposed by Migliorisi [2016], Migliorisi [2019] and Kopacz [2019].

4.5.6 Highcharts

The Accessibility module of the Highcharts library [Highcharts 2021b; Highcharts 2021a] described in Subsection 3.5.1 assigns standard ARIA roles to data series and data points. Moreover, the semantics of certain objects are clearly expressed by a dedicated taxonomy of class names. The resulting system of role and class name combinations is shown in Table 4.7. All the information in this subsection is derived from the sample charts provided at [Highcharts 2021c].

The SVG chart is embedded in a `<div>` element with a tabindex of `0`. Data points and buttons are assigned a tabindex of `-1`. Thus, the chart as a whole can be focused using the `Tab` key and, afterwards, focus can be programmatically set to particular chart elements.

The standard ARIA property aria-label is attached to various chart objects. Its content is automatically generated according to a pattern which can be customised by the author of the chart. The formats used for the sample charts are as follows:

- *Data Series*: `SERIES-TITLE`, `SERIES-TYPE` `SERIES-INDEX` of `TOTAL-SERIES` with `TOTAL-POINTS` data points.

  For example: “VoiceOver, line 3 of 6 with 6 data points.”

- *Data Points*: `POINT-INDEX`. `NAME`, `VALUE`. `SERIES-TITLE`.

  For example: “4. July 2015, 41.4%. NVDA.”

The `SERIES-TYPE` is “line” for line charts, and “series” otherwise. Some sample SVG code annotated according to the Highcharts system is shown in Listing 4.7.

Apart from the SVG structure, several `<div>` elements are inserted for accessibility purposes and are grouped in a `<figure>` element along with the `<div>` container for the chart. They are visually hidden and contain automatically-composed texts about chart type, title, axes, and the number of data series, an optional HTML table representation of the data, and an overall chart description, if provided by the author of the chart. Moreover, two other `<div>` elements are present with the aria-live property, used to make a screen reader announce the information about a chart object interactively while navigating.

Role	Class	Element	Ancestor	Content	Meaning
-	highcharts-root	<svg>	-	all objects	chart root element
-	highcharts-title	<text>	chart	<tspan> element	chart title
-	highcharts-subtitle	<text>	chart	<tspan> element	chart subtitle
-	highcharts-caption	<text>	chart	?	chart caption (no content in all examples)
-	highcharts-credits	<text>	chart	text	chart credits statement
-	highcharts-axis highcharts-xaxis	<g>	chart	axis title and line	x-axis
-	highcharts-axis highcharts-yaxis	<g>	chart	axis title and line	y-axis
-	highcharts-axis-title	<text>	axis	<tspan> element	axis title
-	highcharts-axis-labels highcharts-xaxis-labels	<g>	chart	<text> elements with nested <tspan>	labels of all x-axis items
-	highcharts-axis-labels highcharts-xaxis-labels highcharts-navigator-xaxis	<g>	chart	<text> elements with nested <tspan>	labels of all items of a navigable x-axis
-	highcharts-axis-labels highcharts-yaxis-labels	<g>	chart	<text> elements	labels of all y-axis items
-	highcharts-axis-labels highcharts-yaxis-labels highcharts-navigator-yaxis	<g>	chart	<text> elements	labels of all items of a navigable y-axis
-	highcharts-range-selector-group	<g>	chart		group of controls to select the displayed range of a dataset
-	highcharts-range-selector-buttons	<g>	range selection group	range selection buttons	group of buttons to select the displayed range of a dataset
button	highcharts-button highcharts-button-pressed	<g>	range selection buttons group	<text> element	button to display a predefined range of the dataset, currently selected

Table 4.7: System of standard ARIA role and class name combinations used by Highcharts. N denotes a zero-based index of the data series. (continues on next page)

Role	Class	Element	Ancestor	Content	Meaning
button	highcharts-button highcharts-button-normal	<g>	range selection buttons group	<text> element	button to display a predefined range of the dataset, currently not selected
-	highcharts-input-group	<g>	range selection group	values of selected range	selection values group, labels with the start and end values of the range currently displayed
-	highcharts-label highcharts-range-label	<g>	range values group	<text> element	label for the subsequent range value element; e.g. from
-	highcharts-label highcharts-range-input	<g>	range values group	<text> element with nested <tspan>	range value associated with the preceding label element; e.g. Mar23,2020
-	highcharts-legend	<g>	chart	legend items	legend
-	highcharts-legend-item highcharts-series-N	<g>	legend	<text> and <tspan> elements	legend item
-	highcharts-series-group	<g>	chart	data series	collection of all data series
-	highcharts-series highcharts-series-N highcharts-pie-series	<g>	chart	data points	data series of pie chart
region	highcharts-series highcharts-series-N highcharts-N-series	<g>	chart	data points	data series of other chart type
region	highcharts-markers highcharts-series-N highcharts-spline-series / highcharts-line-series	<g>	chart	data points	data series of line chart
img	highcharts-point	<path>	data series	-	data point

Table 4.7: System of standard ARIA role and class name combinations used by Highcharts. N denotes a zero-based index of the data series.

```
<svg class="highcharts-root" tabindex="0" version="1.1" xml:lang="en"
  lang="en" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink" viewBox="0 190 750 346">

<text class="highcharts-title" id="title">Line Chart of Sales</text>

<text class="highcharts-subtitle" id="desc">
  Chart of sales for 12 months in year 2012 for Salespersons A and B.
</text>

<!-- Y Axis -->

<g class="highcharts-axis highcharts-yaxis">
<line x1="98" y1="443" x2="98" y2="212" />
<text class="highcharts-axis-title" id="y-title"
  transform="matrix(0 -1 1 0 20 355)">
<tspan>Sales in €</tspan>
</text>
</g>


<g class="highcharts-axis-labels highcharts-yaxis-labels">

<g>
<text transform="matrix(1 0 0 1 38 215)">40,000</text>
<line x1="98" y1="212" x2="93" y2="212" />
</g>

<g>
<text transform="matrix(1 0 0 1 38 262)">35,000</text>
<line x1="98" y1="258" x2="93" y2="258" />
</g>

...
</g>

<!-- X Axis -->

<g class="highcharts-axis highcharts-xaxis">
<line x1="98" y1="443" x2="677" y2="443" />
<text class="highcharts-axis-title" id="x-title"
  transform="matrix(1 0 0 1 383 525)">
<tspan>Month</tspan>
</text>
</g>


<g class="highcharts-axis-labels highcharts-xaxis-labels">

<g>
<text id="x-2012-01" transform="matrix(0.7 0.7 -0.7 0.7 102 466)">
<tspan>January 2012</tspan>
</text>
<line x1="98" y1="443" x2="98" y2="448" />
</g>

```

Listing 4.7: Sample SVG code annotated with standard ARIA roles and properties as well as dedicated class names according to the system used by Highcharts. The root `<svg>` element and the data points have the `tabindex` attribute set for keyboard navigation.

```
<g>
<text id="x-2012-02" transform="matrix(0.7 0.7 -0.7 0.7 155 466)">
<tspan>February 2012</tspan>
</text>
<line x1="150" y1="443" x2="150" y2="448" />
</g>

...
</g>


<g class="highcharts-series-group">

<!-- Data Points A -->

<g role="region" aria-label="Salesperson A, line 1 of 2 with 12 data points."
  class="highcharts-markers highcharts-series-0 highcharts-line-series"
  id="data-a">

<polyline points="
   98,319 150,331
  150,331 203,309
  203,309 255,280
  255,280 308,322
  308,322 361,303
  361,303 413,375
  413,375 466,382
  466,382 519,353
  519,353 571,315
  571,315 624,298
  624,298 677,246
  " />

<rect role="img" class="highcharts-point" aria-label="1. January 2012, 28366.
  Salesperson A" tabindex="-1" x="93" y="315" />

<rect role="img" class="highcharts-point" aria-label="2. February 2012, 27050.
  Salesperson A" tabindex="-1" x="146" y="327" />

...
</g>
</g>


<!-- Legend -->

<g class="highcharts-legend" id="legend">

<g class="highcharts-legend-item highcharts-series-1" id="legend-a">
<rect x="168" y="210" />
<text id="legend-text-a" x="185" y="220"
  font-family="Verdana" font-size="14">Salesperson A</text>
</g>

...
</g>

</svg>
```

Listing 4.7 (cont.)**:** Sample SVG code annotated with standard ARIA roles and properties as well as dedicated class names according to the system used by Highcharts.

4.5.7 Semiotic

A system of standard ARIA roles and properties in combination with dedicated class names is also used by the charting library Semiotic [Meeks and Lu 2020] (see Subsection 3.5.3). The resulting taxonomy is presented in Table 4.8. The element representing the whole dataset, in this case, the set of all the data series, is assigned a `tabindex` of `0`, so that it can be focused using the `Tab` key. The ancestor elements for bars and lines have the `tabindex` attribute set to `-1`, which means that they can be focused programmatically. All the details and examples presented in this subsection are based on demonstration charts provided at [Meeks and Lu 2020].

The standard ARIA property `aria-label` is applied to various chart objects with texts generated automatically according to predefined patterns. Examples taken from the demonstration charts include:

- *group of dataset and continuous axes*:
  Visualization with a complex title. Use arrow keys to navigate elements.

- *dataset of bar chart*:
  `TOTAL-POINTS` bars in a bar chart.

- *dataset of line chart*:
  `TOTAL-SERIES` lines in a line chart.

- *data points of bar chart*:
  `x-VALUE` bar value `y-VALUE`

  For example: “Jason bar value 10”

- *data series of line chart*:
  `TOTAL-POINTS` point line starting value `FIRST-VALUE` at `FIRST-NAME`
  ending value `LAST-VALUE` at `LAST-NAME`

  For example: “15 point line starting value 0.01k at 1 ending value 0.018k at 15”

- *continuous axis*:
  `POSITION` axis from `FIRST-AXIS-LABEL` to `LAST-AXIS-LABEL`

  For example: “left axis from 0 to 1”

A complete example of SVG code annotated according to the Semiotic system is shown in Listing 4.8.

Role	Class	Element	Ancestor	Content	Meaning
-	visualization-layer	<svg>	-	all objects	chart root element
-	frame-title	<text>	chart root	text	chart title
group	data-visualization	<g>	dataset and continuous axes	group of dataset and continuous axes	
group	pieces	<g>	group of dataset and continuous axes	dataset for bar charts	
group	lines	<g>	group of dataset and continuous axes	data series	dataset for line charts
img	-	<rect>	dataset	-	data point for bar charts
img	xyframe-line	shape	dataset	-	data series for line charts
-	axisaxis-labels	<g>	group of dataset and continuous axes	continuous axes	group of continuous axes
-	[POS y]	<g>	group of continuous axes	axis labels	continuous y-axis for bar charts
-	axis POS y	<g>	group of continuous axes	axis labels	continuous y-axis for line charts
-	axisx POS	<g>	group of continuous axes	axis labels	continuous x-axis
-	axis-label	<text>	continuous axis	text	label for an item of a continuous axis
-	axis-title [POS y]	<g>	continuous y-axis	<text> element with possible <tspan> elements	title of a continuous y-axis for bar charts
-	axis-titleaxis POS y	<g>	continuous y-axis	<text> element with possible <tspan> elements	title of a continuous y-axis for line charts
-	axis-titleaxis POS x	<g>	continuous x-axis	<text> element with possible <tspan> elements	title of a continuous x-axis for line charts
-	ordinal-labels	<g>	chart root	<text> elements with axis labels	discrete x-axis

Table 4.8: System of standard ARIA role and class name combinations used by Semiotic. Class names in square brackets ([]) are only applied for stacked and grouped bar charts. POS denotes the position of an axis, such as left or bottom.

```
<svg class="visualization-layer" version="1.1" xml:lang="en" lang="en"
  xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink" viewBox="0 190 750 346">

<text class="frame-title" id="title">Line Chart of Sales</text>

<g role="group" class="data-visualization" aria-label="Visualization with
  a complex title. Use arrow keys to navigate elements.">

<g class="axis axis-labels">

<!-- Y Axis -->

<g class="axis left y" aria-label="left axis from 15,000 to 40,000">

<line x1="98" y1="443" x2="98" y2="212" />

<g class="axis-title axis left y">
<text id="y-title" transform="matrix(0 -1 1 0 20 355)">Sales in €</text>
</g>

<g>
<text class="axis-label" transform="matrix(1 0 0 1 38 215)">40,000</text>
<line x1="98" y1="212" x2="93" y2="212" />
</g>

<g>
<text class="axis-label" transform="matrix(1 0 0 1 38 262)">35,000</text>
<line x1="98" y1="258" x2="93" y2="258" />
</g>

...
</g>
</g>


<g role="group" class="lines" aria-label="2 lines in a line chart" tabindex="0">

<!-- Data Points A -->

<g tabindex="-1" id="data-a">

<polyline role="img" class="xyframe-line" aria-label="12 point line starting
  value 28,366 at January 2012 ending value 28,490 at December 2012" points="
   98,319 150,331
  150,331 203,309
  203,309 255,280
  255,280 308,322
  308,322 361,303
  361,303 413,375
  413,375 466,382
  466,382 519,353
  519,353 571,315
  571,315 624,298
  624,298 677,246" />
```

Listing 4.8: Sample SVG code annotated with standard ARIA roles and properties as well as dedicated class names as used in Semiotic. The `<g>` element of the dataset and that of the data series have been assigned a tabindex attribute for keyboard navigation.

```
<rect x="93" y="315" />

<rect x="146" y="327" />

...
</g>

</g>
</g>


<!-- X Axis -->

<text id="x-title" transform="matrix(1 0 0 1 383 525)">Month</text>

<g class="ordinal-labels">

<line x1="98" y1="443" x2="677" y2="443" />

<g>
<text id="x-2012-01" transform="matrix(0.7 0.7 -0.7 0.7 102 466)">
  January 2012</text>
<line x1="98" y1="443" x2="98" y2="448" />
</g>

<g>
<text id="x-2012-02" transform="matrix(0.7 0.7 -0.7 0.7 155 466)">
  February 2012</text>
<line x1="150" y1="443" x2="150" y2="448" />
</g>

...
</g>


<!-- Legend -->

<g id="legend">

<g id="legend-a">
<rect x="168" y="210" class="marker-square" />
<text id="legend-text-a" x="185" y="220"
  font-family="Verdana" font-size="14">Salesperson A</text>
</g>

...
</g>

</svg>
```

Listing 4.8 (cont.)**:** Sample SVG code annotated with standard ARIA roles and properties as well as dedicated class names as used in Semiotic.

Role	Element	Ancestor	Content	Meaning
group	`<svg>`	-	chart(s)	`<svg>` root element
region	`<g>`	`<svg>`	all objects	chart root element
menu	`<g>`	chart	data points	data series
menuitem	`<g>`	data series	data value element	data point

Table 4.9: Standard ARIA roles used by the charting library amCharts.

4.5.8 amCharts

The charting library amCharts [amCharts 2020a] described in Subsection 3.5.4 uses the standard ARIA roles shown in Table 4.9. The roles for data series and points were originally intended to be used for menus within a web application. The data points have the `tabindex` attribute set to `0` and the `focusable` attribute set to `true` so that they can be navigated using the `Tab` key. Other chart objects can also be declared focusable by the author of the chart. The information and examples given in this subsection are derived from sample charts found at amCharts [2020b].

In addition to the roles described in Table 4.9, the ARIA property `aria-describedby` is used to convey some information too, pointing to `<desc>` elements and assigned to the chart root and to the data series. The label of a data point is attached programmatically at runtime when it receives keyboard focus. All accessible descriptions are customisable by the author of the chart and default to the following patterns:

- *chart root*: corresponds to the visible chart title if specified, otherwise defaults to `Chart`.
- *data series*: corresponds to the data series title, if specified, otherwise no name or description.
- *data points*: `x-VALUE y-VALUE`

  For example: "Research & Development 1200"

  No accessible names or descriptions are assigned to data points of line charts.

Other chart objects, such as the data points of a line chart, can additionally be given an accessible name and description by the author. A sample SVG code excerpt from amCharts is presented in Listing 4.9.

```
<svg role="group" version="1.1" xml:lang="en" lang="en" viewBox="0 190 750 346"
  xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">

<desc>JavaScript chart by amCharts</desc>

<g role="region" aria-describedby="desc">

<desc id="desc">Line Chart of Sales</desc>

<!-- Y Axis -->

<g>

<line x1="98" y1="443" x2="98" y2="212" class="chart-line" />
<text id="y-title" transform="matrix(0 -1 1 0 20 355)"
  class="chart-title">Sales in €</text>

<g>
<text transform="matrix(1 0 0 1 38 215)" class="chart-label">40,000</text>
<line x1="98" y1="212" x2="93" y2="212" class="chart-line" />
</g>

<g>
<text transform="matrix(1 0 0 1 38 262)" class="chart-label">35,000</text>
<line x1="98" y1="258" x2="93" y2="258" class="chart-line" />
</g>

...
</g>

<!-- X Axis -->

<g>

<line x1="98" y1="443" x2="677" y2="443" class="chart-line" />
<text id="x-title" transform="matrix(1 0 0 1 383 525)"
  class="chart-title">Month</text>

<g>
<text id="x-2012-01" transform="matrix(0.7 0.7 -0.7 0.7 102 466)"
  class="chart-label">January 2012</text>
<line x1="98" y1="443" x2="98" y2="448" class="chart-line" />
</g>

<g>
<text id="x-2012-02" transform="matrix(0.7 0.7 -0.7 0.7 155 466)"
  class="chart-label">February 2012</text>
<line x1="150" y1="443" x2="150" y2="448" class="chart-line" />
</g>

...
</g>
```

Listing 4.9: Sample SVG code annotated with standard ARIA roles and the aria-describedby property in amCharts. The data points have a tabindex of 0 and the focusable attribute for Tab navigation. The class names are arbitrary and used only for styling.

```


<!-- Data Points A -->

<g role="menu" aria-describedby="legend-text-a" id="data-a">

<desc id="legend-text-a">Salesperson A</desc>

<polyline points="
    98,319 150,331
   150,331 203,309
   203,309 255,280
   255,280 308,322
   308,322 361,303
   361,303 413,375
   413,375 466,382
   466,382 519,353
   519,353 571,315
   571,315 624,298
   624,298 677,246
  " class="chart-lineA" />

<g role="menuitem" focusable="true" tabindex="0">
<rect x="93" y="315" class="marker-square" />
</g>

<g role="menuitem" focusable="true" tabindex="0">
<rect x="146" y="327" class="marker-square" />
</g>

...
</g>


<!-- Legend -->

<g id="legend">

<g id="legend-a">
<rect x="168" y="210" class="marker-square" />
<text id="legend-text-a" x="185" y="220"
  font-family="Verdana" font-size="14">Salesperson A</text>
</g>

...
</g>

</g>
</svg>
```

Listing 4.9 (cont.)**:** Sample SVG code annotated with standard ARIA roles and the aria-describedby property in amCharts.

Class	Element	Ancestor	Content	Meaning
raphael-group-N-parentgroup	<g>	<svg>	all objects	chart root element
raphael-group-N-background	<g>	chart root	-	chart background
raphael-group-N-dataset-axis-name	<g>	chart root	<text> element	axis title, associated with subsequent axis labels
raphael-group-N-dataset-axis	<g>	chart root	<text> elements	axis labels, associated with preceding axis title
raphael-group-N-dataset-top-label	<g>	chart root	<text> elements	labels of angular gauge chart
raphael-group-N-plot-group	<g>	chart root	shape elements as data points	data series
raphael-group-N-legend	<g>	chart root	<text> elements as legend items	legend
raphael-group-N-caption	<g>	chart root	<text> element(s)	caption

Table 4.10: Selected class names used by the charting library FusionCharts. `N` denotes a variable number.

4.5.9 FusionCharts

In charts created by the library FusionCharts with its accessibility extension [FusionCharts 2020], as described in Subsection 3.5.2, the <svg> element is assigned the standard ARIA role application. This role is intended to be used to indicate web applications. A screen reader switches to application mode in JAWS or focus mode in NVDA, as soon as the SVG receives keyboard focus. In combination with the attributes tabindex and focusable attached to the root <svg> element and certain chart objects, the chart can then be navigated by keyboard.

Moreover, the standard ARIA role button is used for <text> elements which represent legend items, denoting the titles of the data series. They can also be used to toggle the visibility of a particular data series. All the legend items are assigned the tabindex and focusable attributes. In the case of the first legend item, tabindex is set to `0`, whereas for all the other items the attribute is set to `-1`. The result is that navigation by the `Tab` key always sets the keyboard focus to the first item. Subsequent items can be focused programmatically. The same system of focus management is applied to the shape elements representing the data points. Furthermore, certain objects of a chart can be identified by dedicated class names. Those which are regarded as relevant for expressing semantic information are listed in Table 4.10. The details and examples given in this subsection are taken from the demonstration charts available at [FusionCharts 2020].

In addition, the chart root element, data points, and legend items are assigned an aria-label property with titles, which are generated automatically according to customisable patterns. The default string patterns for bar and line charts are:

- *chart root*: This is a `CHART-TYPE` chart created with FusionCharts Suite XT. Title of the chart is `CAPTION`. `x-AXIS-TITLE` is plotted on x-axis and `y-AXIS-TITLE` is plotted on y-axis.

  For example: “This is a multi series line chart created with FusionCharts Suite XT. Title of the chart is Social Media Platforms Popularity. Years is plotted on x-axis and Popularity is plotted on y-axis.”

- *data points*: `y-AXIS-TITLE` of `SERIES-TITLE` for `x-AXIS-TITLE NAME` is `y-VALUE`. Plot `POINT-INDEX` of `TOTAL-POINTS`. Series `SERIES-INDEX` of `TOTAL-SERIES`.

  For example: “Popularity of Facebook for Years 2012 is 62%. Plot 1 of 5. Series 1 of 4.”

- *legend items*: Toggle the visibility of `ITEM_LABEL`.

  For example: "Toggle the visibility of Twitter."

Some example SVG code annotated according to the system used by FusionCharts is presented in Listing 4.10. For non-cartesian charts, the default string patterns are:

- *chart root*: This is a `CHART-TYPE` chart created with FusionCharts Suite XT. `CAPTION` for `SUBCAPTION` is plotted.

  For example: "This is a pie chart created with FusionCharts Suite XT. Web Servers Market Share for 2015—16 is plotted."

- *data points*: `POINT-INDEX. LABEL, VALUE.`

  For example: 4.Other,18.67M.

- *legend items*: Toggle the slicing of `LABEL`.

  For example: "Toggle the slicing of Other."

```
<svg role="application" class="raphael-group-1-parentgroup" tabindex="0"
  focusable="true" aria-label="This is a multi series line chart created
  with FusionCharts Suite XT. Title of the chart is Line Chart of Sales.
  Month is plotted on x-axis and Sales in € is plotted on y-axis"
  version="1.1" xml:lang="en" lang="en" xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink" viewBox="0 190 750 346">

<g class="raphael-group-1-caption">
<text id="title">Line Chart of Sales</text>
</g>

<!-- Y Axis -->

<g>

<line x1="98" y1="443" x2="98" y2="212" />

<g class="raphael-group-1-dataset-axis-name">
<text id="y-title" transform="matrix(0 -1 1 0 20 355)">Sales in €</text>
</g>

<g class="raphael-group-1-dataset-axis">

<g>
<text transform="matrix(1 0 0 1 38 215)">40,000</text>
<line x1="98" y1="212" x2="93" y2="212" />
</g>

<g>
<text transform="matrix(1 0 0 1 38 262)">35,000</text>
<line x1="98" y1="258" x2="93" y2="258" />
</g>

...
</g>
</g>


<!-- X Axis -->

<g>

<line x1="98" y1="443" x2="677" y2="443" />

<g class="raphael-group-1-dataset-axis-name">
<text id="x-title" transform="matrix(1 0 0 1 383 525)">Month</text>
</g>

<g class="raphael-group-1-dataset-axis">

<g>
<text id="x-2012-01" transform="matrix(0.7 0.7 -0.7 0.7 102 466)">
  January 2012</text>
<line x1="98" y1="443" x2="98" y2="448" />
</g>
```

Listing 4.10: Sample SVG code annotated with standard ARIA roles and properties as well as dedicated class names in FusionCharts. The data points and legend items have both tabindex and focusable attributes set for keyboard navigation.

```

<g>
<text id="x-2012-02" transform="matrix(0.7 0.7 -0.7 0.7 155 466)">
  February 2012</text>
<line x1="150" y1="443" x2="150" y2="448" />
</g>

...
</g>
</g>


<!-- Data Points A -->

<g class="raphael-group-1-plot-group" id="data-a">

<polyline points="
   98,319 150,331
  150,331 203,309
  203,309 255,280
  255,280 308,322
  308,322 361,303
  361,303 413,375
  413,375 466,382
  466,382 519,353
  519,353 571,315
  571,315 624,298
  624,298 677,246" />

<rect aria-label="Sales in € of Salesperson A for Month January 2012 is 28366.
  Plot 1 of 12. Series 1 of 2" tabindex="0" focusable="true" x="93" y="315" />

<rect aria-label="Sales in € of Salesperson A for Month February 2012 is 27050.
  Plot 2 of 12. Series 1 of 2" tabindex="-1" focusable="true" x="146" y="327" />

...
</g>


<!-- Legend -->

<g class="raphael-group-1-legend" id="legend">

<text role="button" aria-label="Toggle the visibility of Salesperson A."
  tabindex="0" focusable="true" id="legend-text-a" x="185" y="220"
  font-family="Verdana" font-size="14">Salesperson A</text>

...
</g>

</svg>
```

Listing 4.10 (cont.)**:** Sample SVG code annotated with standard ARIA roles and properties as well as dedicated class names in FusionCharts.

4.5.10 AnyChart

Charts created by the library AnyChart, described in Subsection 3.5.5, assign the standard ARIA role `presentation` to the `<svg>` root element by default [AnyChart 2020a; AnyChart 2020b]. This role is defined in the core WAI-ARIA specification [W3C 2017a] for graphics, but was intended for those graphics without semantic significance. Within the SVG, the chart is wrapped in a `<g>` container element with the standard role `article` and the standard property `aria-label`. The latter holds a description which can be customised by the author of the chart and defaults to the following string pattern:

- *chart*: `CHART-TYPE` chart entitled `CHART-TITLE`, with `TOTAL-SERIES` `CHART-TYPE` series, . Y-scale minimum value is `MIN-y-VALUE` , maximum value is `MAX-y-VALUE`. X-scale with `TOTAL-x-AXIS-LABELS` categories: `x-AXIS-LABEL1`, `x-AXIS-LABEL2`, `x-AXIS-LABEL3`, . . . , .

  For example: “line chart entitled Trend of Sales of the Most Popular Products of ACME Corp., with 3 line series, . Y-scale minimum value is 0 , maximum value is 28. X-scale with 24 categories: 1986, 1987, 1988, 1989, 1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, .”

These details were derived from the samples at AnyChart [2020b].

```
<svg role="presentation" version="1.1" xml:lang="en" lang="en"
  xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink" viewBox="0 190 750 346">

<g role="article" aria-label="line chart entitled Line Chart of Sales, with 2
  line series, . Y-scale minimum value is 15,000 , maximum value is 40,000.
  X-scale with 12 categories: January 2012, February 2012, March 2012, April
  2012, May 2012, June 2012, July 2012, August 2012, September 2012, October
  2012, November 2012, December 2012, .">

<!-- Y Axis -->

<g>

<line x1="98" y1="443" x2="98" y2="212" class="chart-line" />
<text id="y-title" transform="matrix(0 -1 1 0 20 355)"
  class="chart-title">Sales in €</text>

<g>
<text transform="matrix(1 0 0 1 38 215)" class="chart-label">40,000</text>
<line x1="98" y1="212" x2="93" y2="212" class="chart-line" />
</g>

<g>
<text transform="matrix(1 0 0 1 38 262)" class="chart-label">35,000</text>
<line x1="98" y1="258" x2="93" y2="258" class="chart-line" />
</g>

...
</g>


<!-- X Axis -->

<g>

<line x1="98" y1="443" x2="677" y2="443" class="chart-line" />
<text id="x-title" transform="matrix(1 0 0 1 383 525)"
  class="chart-title">Month</text>

<g>
<text id="x-2012-01" transform="matrix(0.7 0.7 -0.7 0.7 102 466)"
  class="chart-label">January 2012</text>
<line x1="98" y1="443" x2="98" y2="448" class="chart-line" />
</g>

<g>
<text id="x-2012-02" transform="matrix(0.7 0.7 -0.7 0.7 155 466)"
  class="chart-label">February 2012</text>
<line x1="150" y1="443" x2="150" y2="448" class="chart-line" />
</g>

...
</g>
```

Listing 4.11: Sample SVG code annotated with standard ARIA roles and properties according to the system used by AnyChart. The class names are arbitrary and used only for styling.

```


<!-- Data Points A -->

<g id="data-a">

<polyline points="
   98,319 150,331
  150,331 203,309
  203,309 255,280
  255,280 308,322
  308,322 361,303
  361,303 413,375
  413,375 466,382
  466,382 519,353
  519,353 571,315
  571,315 624,298
  624,298 677,246
  " class="chart-lineA" />

<g>
<rect x="93" y="315" class="marker-square" />
</g>

<g>
<rect x="146" y="327" class="marker-square" />
</g>

...
</g>


<!-- Legend -->

<g id="legend">

<g id="legend-a">
<rect x="168" y="210" class="marker-square" />
<text id="legend-text-a" x="185" y="220"
  font-family="Verdana" font-size="14">Salesperson A</text>
</g>

...
</g>

</g>
</svg>
```

Listing 4.11 (cont.)**:** Sample SVG code annotated with standard ARIA roles and properties according to the system used by AnyChart.

Chapter 5

Accessible Charts with AChart

The AChart (Accessible Chart) project was launched at Graz University of Technology's Institute of Interactive Systems and Data Science (ISDS) in 2019. Its goal is to provide open-source software solutions for producing and interpreting accessible charts and data visualisations in Scalable Vector Graphics (SVG) format. For this purpose, two complementary software systems were built. Firstly, AChart Creator can be used to automatically generate accessible SVG charts from tabular data. Its modular source code was designed in a way that it can be extended to other types of visualisations with relatively low effort. The functionality and implementation of AChart Creator will be described in Chapter 6.

Secondly, AChart Interpreter can interpret and read out the accessible charts produced by AChart Creator, in a manner similar to Describler [Schepers 2015a]. Its split-screen GUI displays a chart both visually and in textual form. The closely related spin-off command-line tool AChart Summariser produces a solely textual summary of an accessible chart to the console. A detailed description of AChart Interpreter and AChart Summariser will be given in Chapter 7.

The current chapter covers general aspects which are relevant for the AChart project as a whole. In addition to the work done in the context of this thesis, other developers also contributed to the AChart software. Table 5.1 lists all participants of the AChart project along with their contributions.

A web-based approach was chosen for the implementation of the AChart software. The programmes were written in TypeScript [Microsoft 2020b], a class-based superset of JavaScript. The TypeScript source code needs to be *transpiled* into JavaScript for execution. For transpilation, the TypeScript compiler version 3.4.5 was used, set to ECMAScript version 5 as its target output. An automated tool chain for transpilation, copying, and execution was facilitated by the task runner gulp.js [Bublitz and Schoffstall 2021] and the implementation of appropriate gulp scripts.

Contributions	Participants
AChart Creator:	
First proof-of-concept.	Alexander Grass, Lea Novak, and Danica Radulovic [Grass et al. 2019]
Command-line options for chart, axis titles, and descriptions.	Inti Gabriel Mendoza Estrada
New, class-based design, standalone version, automatic titles, multi-line charts, data column selection, sorting of data points, creation of legends, accessibility enhancements, visual corrections.	Christopher Alexander Kopel
AChart Interpreter:	
Main implementation.	Christopher Alexander Kopel
Visually optimised user interface, synchronised highlighting of chart objects.	Inti Gabriel Mendoza Estrada; Lukas Bodner, Daniel Geiger, and Lorenz Leitner [Bodner et al. 2020a]
Enhanced speech synthesis detection.	Lukas Bodner, Daniel Geiger, and Lorenz Leitner [Bodner et al. 2020a]
AChart Summariser:	
Main implementation.	Christopher Alexander Kopel

Table 5.1: Contributors to the AChart project at the time of writing.

5.1 Motivation

As described in Chapter 3, numerous solutions have been proposed with the goal of giving visually impaired recipients access to data visualisations and charts. However, many of these approaches only exist as laboratory prototypes and are not publicly available. Solely tactile representations do not provide a suitable means to rapidly understand complex visual information for most users, as explained in Section 3.1. Moreover, all the tactile and most of the multimodal solutions require special hardware, which leads to increased cost for the potential recipients and is disadvantageous in terms of portability. All the systems presented in Sections 3.1 to 3.3 are self-contained and do not interact with any screen reader, with the consequence that the user needs to learn new commands and, in some cases, to deactivate the screen reader in order to avoid conflicts. Lastly, many of the solutions can be classified as closed systems, which do not interact with any graphics software or format, but try to recognise the information from an image, which always introduces the risk of erroneous or unsuitable results.

Both iGraph-Lite by Ferres et al. [2007], Ferres et al. [2013] and Ferres [2015] (see Subsection 3.2.1) and Plot Explorer by Revnitski [2005] (see Section 3.3) require use of a special software library or plug-in to be be addressed by the graphics application. For example, iGraph-Lite uses a visual description based on the Web Ontology Language (OWL) as an intermediate format.

Several other proposals, by contrast, use annotated SVG formats for embedding accessible information in graphics documents. The Graphics Accessible To Everyone (GATE) system by Kopecek and Oslejsek [2008] (see Subsection 3.2.3) and IVEO by Bulatov et al. [2005], Gardner and Bulatov [2006], Gardner and Bulatov [2010] and Gardner [2016] both use special data structures in addition to the SVG code. Rotard et al. [2004]'s prototype system, the SVG-Plott software by Engel et al. [2019] (see Section 3.3), and the Access2Graphics approach by Dürnegger et al. [2010] and Altmanninger and Wöß [2008] (see Section 3.4) all use annotated SVG formats, but solely make use of SVG's native `<title>` and `<desc>`

elements to provide accessible names and descriptions. None of them explicitly defines a taxonomy for identifying the type of chart components, such as axes, data series, and data points. Expressing such type information within accessible names and/or descriptions is possible, but has the disadvantage that more computation is necessary to identify these types and, thus, to perform any further processing of the underlying data. In contrast, the BrailleR extension by Fitzpatrick et al. [2017], Godfrey et al. [2018] and Sorge [2016] described in Section 3.4 expresses such semantics in an XML structure associated with the SVG elements of the visual chart.

The AChart project aims to develop a flexible software solution which can be used in conjunction with common screen readers, applies an open SVG-based system for unambiguously identifying chart objects, and provides the possibility of being easily extensible to other systems and visualisation types. While most of the charting libraries discussed in Section 3.5 generally fulfil the criteria of screen reader compatibility and semantic SVG enrichment, they do not provide any analytical features helping to rapidly understand the data, such as comparisons of data points or statistical summaries. Moreover, in many cases, they express the types of chart objects through dedicated CSS class names. An essential aspect for the design of the AChart software, by contrast, was to embed accessible information as recommended by the World Wide Web Consortium (W3C), that is, by means of native SVG elements in combination with roles and properties based on the Accessible Rich Internet Applications (ARIA) system. Since, at the time of writing, no ARIA taxonomy for charts has been standardised, it was decided to apply one of the proposals listed in Section 4.5, which will be described in the following section.

The AChart Interpreter software was inspired by the Describler application [Schepers 2015a; Schepers 2017] in two ways. Firstly, Describler uses SVG along with a taxonomy of ARIA roles and properties for declaring chart objects (see Subsection 4.5.1). Moreover, it provides a user interface which facilitates the visual and auditory exploration of annotated SVG charts by means of the mouse or keyboard, as described in Section 3.6. However, Describler is perhaps best regarded as an inspiring proof-of-concept, since it is somewhat limited in its functionality. For example, the software does not fully handle charts with multiple data series: if more than one data series is present, all data points are listed, but the different data series are not reported as such to the user, and most of the analytical functions are supported only for the first series. With regard to user interaction, no key command for rapidly leaving a data series is provided; the only way is to step through all remaining data points of the series from the object currently focused. Finally, Describler does not interact with screen readers in any way: the recognised information is available through the integrated speech output or the read-only text field, which can be accessed by a screen reader user only after moving the focus out of the chart. In addition, all the menu items need to be invoked by the number keys, which are usually blocked by the navigation facilities of most screen readers [Vispero 2020; NV Access 2020]. AChart Interpreter was carefully designed to provide an optimised user experience for both sighted and blind users.

5.2 Developing an ARIA-Based System for Charts

Various proposals for enriching SVG charts with ARIA-based roles and properties were introduced in Section 4.5. Each of these approaches exhibit certain advantages and drawbacks. The Describler system [Schepers 2015a; Schepers 2017] presented in Subsection 4.5.1 represents an intuitive and logical strategy to clearly express the semantics of chart objects. However, it lacks a role for identifying a data series and its title; all sample charts provided with the Describler software contain only one data series with no explicit title. Furthermore, this system does not consider chart types other than bar, line, and pie charts. It defines no roles for axes apart from the x- and y-axis, no way to express the values for data point of higher dimensions, and no other values for the aria-charttype property than bar, line, and pie. The name for a data point is given by referencing an x-axis label or legend item using the standard aria-labelledby property attached to the data value object, which is problematic in the case of a continuous axis as it might not contain an item for each data point. Lastly, using the role heading to identify a title for chart objects of different types has the consequence that the reading software needs to determine which object the title belongs to, introducing unnecessary computational effort and uncertainties. The same applies for the inconsistent definition of the chart root object either as a separate <g> or as the root <svg> element.

The main advantage of the approach to use the standard ARIA roles for tables, as proposed by Watson [2017] and described in Subsection 4.5.4, is that such standard ARIA roles are recognised and conveyed in a meaningful way by most modern browsers and screen readers. For charts of tabular data, the table format might appear intuitive and logical, and the data can be navigated by the special screen reader commands for reading tables. However, a table does not necessarily contain counterparts for all possible objects of such a chart. In the case of bar charts, the x-axis may contain the names of all data points and, thus, be represented by the first row of the pseudo-table, but this analogy is not valid for data points which do not correspond to an item of the x-axis. In addition, assigning the role row to the x-axis uses the same role assigned to a data series. It can therefore only be distinguished from a data series by its position within the table and its contained objects. The y-axis and the chart type are not represented at all. A solution to these problems could be to combine the pseudo-table representation of the data with other, possibly non-standard, roles and properties. This would mean the document no longer completely conforms to the W3C standards, but should not interfere with the interpretation of the table by screen readers.

Most of the positive and negative aspects just discussed also apply for the pseudo-list approach by Migliorisi [2016] and Kopacz [2019] presented in Subsection 4.5.5. In addition, the original approach does not identify the names and values of a data point, but concatenates them to one string, which complicates any further processing of the data. Another disadvantage compared to the pseudo-table approach is that screen readers usually do not provide any special navigation mechanisms for lists like they do for tables.

A different strategy complying with the W3C recommendations is the application of the three roles defined in the WAI-ARIA Graphics Module [W3C 2018e], proposed by Schepers [2019] and described in Subsection 4.5.2. In fact, the W3C specification recommends to use these three roles for identifying three different layers in the hierarchical structure of a graphics document. Nevertheless, the roles only convey the semantic information that the annotated objects are graphical and express their structural relationship within the document, but do not give any further details about the meaning of the graphics. More specific semantics could be exposed through the aria-label, aria-labelledby, and/or aria-describedby properties or embedded within descendant SVG elements for text content, but this would prevent a clear distinction between the types on the one hand and the accessible name and description on the other hand.

In a proposal for further defining the semantics of chart objects, Schepers [2019] recommends combining the three ARIA graphics roles with the property aria-roledescription, but provides neither a taxonomy of values for this property nor any sample chart annotated according to this proposal. A possible strategy could be to transfer the Describler taxonomy of roles to a system of role descriptions using aria-roledescription. Another question would be how to express the chart and axis types without non-standard ARIA attributes.

In this context, it should be noted that the correct usage and interpretation of the aria-roledescription property is not unambiguously defined. The WAI-ARIA Graphics Module lists it among the inherited states and properties of all the three roles, but explicitly recommends it only in combination with the role graphics-symbol where it "can be used to name the symbol type separately from the name and description for the particular instance of the symbol" [W3C 2018e, Subsection 4.1]. The core WAI-ARIA document says that the property defines "a human-readable, author-localized description for the role of an element" [W3C 2017a, Subsection 6.6]. This implies that its original purpose is providing a text which is conveyed verbatim to the user, rather than specifying a string for automated processing.

Based on the various aspects just discussed, it is reasonable to claim that a system restricted to the current ARIA roles, states, and properties standardised by the W3C cannot sufficiently meet the special requirements for comprehensively enriching all chart objects with appropriate semantics. The taxonomy developed by the W3C SVG Accessibility Task Force [W3C 2015b] (see Subsection 4.5.3) may be regarded as a promising solution. However, as discovered in research for this thesis, this work has not been continued since 2015, and the authors neither provide any example SVG charts annotated with these attributes, nor has the system been applied in any way so far. In addition, it may be argued that the prefix graphics-* for all the defined roles unnecessarily lengthens attribute names and increases the probability of syntax errors. The systems used for the different charting libraries to identify and mark up chart objects, described in Subsections 4.5.6 to 4.5.10, have not publicly been documented and, according to the analysis for this thesis, partly rely on non-standard attributes beyond any ARIA context.

5.3 AChart Taxonomy of Roles and Properties

For the reasons given above, a taxonomy of roles and properties is defined for AChart, which extends and builds upon Describler's approach, since the latter is considered to be clear and consistent. The system overcomes some of the insufficiencies of the original approach, but ensures the highest possible compatibility with the existing Describler software. In particular, the following changes have been made to the Describler taxonomy:

- The standard ARIA role graphics-document is assigned to the root <svg> element.
- The chart root object is specified as a <g> element with the defined role chart.
- A title object is given the standard ARIA role heading and is referenced by the standard property aria-labelledby of the associated ancestor object.
- A single data series is identified using the role dataset.
- The title of a data series is identified using the standard ARIA role heading.
- The name of a data point is embedded as a descendant element and is assigned the standard ARIA role of heading. It need not be given, if the data point name is present as an axis label or legend item.

The resulting AChart taxonomy of roles is shown in Table 5.2, the AChart system of properties is shown in Table 5.3. A sample SVG chart annotated according to the AChart taxonomy can be seen in Listing 5.1.

Role	Element	Ancestor	Content	Meaning
graphics-document	<svg>	-	all charts	graphics root element
chart	<g>	<svg>	all objects	chart root element
heading*	<title>/<text>	chart/scale/ data series	text	title of chart/scale/data series
chartarea	<rect>	chart	-	chart outline (used to detect visual height and width)
xaxis	<g>	chart	axis title and labels	x-axis
yaxis	<g>	chart	axis title and labels	y-axis
axislabel	<title>/<text>	axis	text	label of axis item
legend	<g>	chart	legend title and items	legend
legenditem	<g>	legend	<text> element	legend item
dataset	<g>	chart	data points	data series
datagroup	<g>	data series	data group title and data points	collection of related data points
datapoint	<g>/shape	data series / data group	data point names and values	data point
heading*	<title>/<text>	data point	text	name of data point (optional, if already given by scale item)
datavalue	<title>/<text>	data point	text	value of data point

Table 5.2: Taxonomy of ARIA roles used by AChart, based on those of Describler. An asterisk (*) indicates that this role is defined in an official ARIA specification.

Property	Object	Value	Meaning
aria-charttype	chart root	bar/pie/line	type of chart
aria-axistype	axis	category	indicates axis has discrete values
aria-valuemin*	numerical axis	number	minimum value
aria-valuemax*	numerical axis	number	maximum value
aria-labelledby*	chart/chart object	id of associated title	references title of chart or chart object
aria-labelledby*	data point	id of name	references name of data point

Table 5.3: Taxonomy of ARIA properties used by AChart, based on those of Describler. An asterisk (*) indicates that this property is defined in an official ARIA specification.

```
<svg role="graphics-document" version="1.1" xml:lang="en" lang="en"
  xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink" viewBox="0 190 750 346">

<g role="chart" aria-charttype="line" aria-labelledby="title" tabindex="0">

<title role="heading" id="title">Line Chart of Sales</title>

<desc id="desc">
  Chart of sales for 12 months in year 2012 for Salespersons A and B.
</desc>

<!-- Y Axis -->

<g role="yaxis" aria-valuemin="15000" aria-valuemax="40000"
  aria-labelledby="y-title" aria-orientation="vertical" tabindex="0">

<line x1="98" y1="443" x2="98" y2="212" class="chart-line" />
<text role="heading" id="y-title" transform="matrix(0 -1 1 0 20 355)"
  class="chart-title">Sales in €</text>

<g>
<text role="axislabel" transform="matrix(1 0 0 1 38 215)"
  class="chart-label">40,000</text>
<line x1="98" y1="212" x2="93" y2="212" class="chart-line" />
</g>

<g>
<text role="axislabel" transform="matrix(1 0 0 1 38 262)"
  class="chart-label">35,000</text>
<line x1="98" y1="258" x2="93" y2="258" class="chart-line" />
</g>

...
</g>

<!-- X Axis -->

<g role="xaxis" aria-axistype="category" aria-labelledby="x-title"
  aria-orientation="horizontal" tabindex="0">

<line x1="98" y1="443" x2="677" y2="443" class="chart-line" />
<text role="heading" id="x-title" transform="matrix(1 0 0 1 383 525)"
  class="chart-title">Month</text>

<g>
<text role="axislabel" id="x-2012-01" class="chart-label"
  transform="matrix(0.7 0.7 -0.7 0.7 102 466)">January 2012</text>
<line x1="98" y1="443" x2="98" y2="448" class="chart-line" />
</g>

<g>
<text role="axislabel" id="x-2012-02" class="chart-label"
  transform="matrix(0.7 0.7 -0.7 0.7 155 466)">February 2012</text>
<line x1="150" y1="443" x2="150" y2="448" class="chart-line" />
</g>
```

Listing 5.1: Sample SVG code annotated with the ARIA roles and properties defined by AChart, based on those of Describler. Certain chart objects are assigned a `tabindex` of `0` for `Tab` navigation. The class names are arbitrary and used only for styling.

```

...
</g>

<!-- Data Points A -->

<g role="dataset" aria-labelledby="legend-text-a" id="data-a">

<polyline points="
  98,319 150,331
  150,331 203,309
  203,309 255,280
  255,280 308,322
  308,322 361,303
  361,303 413,375
  413,375 466,382
  466,382 519,353
  519,353 571,315
  571,315 624,298
  624,298 677,246
  " class="chart-lineA" />

<g role="datapoint" aria-labelledby="x-2012-01" tabindex="0">
<title role="datavalue" aria-labelledby="yScale">28366</title>
<rect x="93" y="315" class="marker-square" />
</g>

<g role="datapoint" aria-labelledby="x-2012-02" tabindex="0">
<title role="datavalue" aria-labelledby="yScale">27050</title>
<rect x="146" y="327" class="marker-square" />
</g>

...
</g>

<!-- Legend -->

<g role="legend" tabindex="0" id="legend">

<g role="legenditem" tabindex="0" id="legend-a">
<rect x="168" y="210" class="marker-square" />
<text id="legend-text-a" x="185" y="220"
  font-family="Verdana" font-size="14">Salesperson A</text>
</g>

...
</g>

</g>
</svg>
```

Listing 5.1 (cont.)**:** Sample SVG code annotated with the ARIA roles and properties defined by AChart, based on those of Describler.

Chapter 6

AChart Creator

Accessible Chart Creator (AChart Creator) is a command-line programme which produces charts in Scalable Vector Graphics (SVG) format and enriches them with machine-readable semantic information. It reads tabular data from files in Comma-Separated Values (CSV) format and transforms them into a visualisation, where the user can choose one of multiple chart types. The semantics are embedded into native SVG elements, such as `<title>`, `<desc>`, and `<text>`, in combination with roles and properties based on the Accessible Rich Internet Applications (ARIA) system according to the AChart taxonomy of roles and properties described in Section 5.3. The result is saved to an SVG file, which can then be viewed by means of any SVG rendering engine, as well as explored in Describler (see Section 3.6) and AChart Interpreter (see Chapter 7). The programme is launched and controlled via command line arguments. Messages are sent to the standard output stream (stdout) or, in case of an error, to the standard error stream (stderr).

The development of AChart Creator was motivated by several aspects. The JavaScript library D3 [Bostock 2021] is widely used to create SVG visualisations within web documents. However, despite extensive research, Grass et al. [2019, page 5] were, at that time, unable to find any software tools capable of creating SVG *files* using D3. While producing an SVG file could be accomplished by creating SVG within a web page and then saving the resulting graphics by means of the browser, this strategy is time-consuming, can introduce baggage into the resulting file, and does not support the automated generation of multiple SVG documents. The source code of AChart Creator and its build scripts provide a tool chain for graphics authors working with D3 to easily produce SVG files with custom visualisations. In addition, they demonstrate how to create accessible SVG charts with D3. Finally, AChart Creator serves as a first solution for producing charts which conform to the ARIA taxonomy proposed in this thesis.

The core visualisation functionality of the software was originally developed by Grass et al. [2019]. In the context of this thesis, the programme was extended by several functions and bundled into a user-friendly standalone command-line tool. The software is open-source under an MIT licence and is available at [Kopel, Andrews, Mendoza Estrada, Grass et al. 2021]. The first section of this chapter describes the functionality and behaviour of AChart Creator. Afterwards, an overview of the implementation will be given, comparing the previous versions to the current one and describing the enhancements.

6.1 User Interaction

AChart Creator can be started by calling its executable binary file from the console. If the executable file is located in the current working directory or in a directory included within the search path of the platform, then the command `acreate` or `./acreate` will typically launch the programme. The command has the syntax shown in Listing 6.1.

All options are treated as case-insensitive and can be used in arbitrary order. In case an argument other than the above options is given, the programme will exit with an error message stating that an invalid option has been used. Similarly, if one of the above options requiring an associated argument is used

```
acreate [--chart] CHART-TYPE [--dataset CSV-FILENAME]
        [--output SVG-FILENAME] [--chart-title TITLE]
        [--chart-desc DESCRIPTION] [--x-axis-title TITLE]
        [--y-axis-title TITLE] [--legend-title TITLE] [--target SOFTWARE]
        [--column DATA-COLUMN] [--svg-precision PLACES] [--no-sort]
        [--no-legend] [--no-tooltips] [--no-bar-values]
        [--no-segment-values] [--no-segment-percentages]
        [--segment-percentage-precision PLACES] [--version] [--help]
```

Listing 6.1: The command line syntax and options of AChart Creator.

without the argument, the programme will exit with an appropriate error message. The specification of a chart type is mandatory. It can either be given as the first argument or at any position if preceded by `--chart`. If this parameter is missing or if no valid chart type is given, the programme will display a corresponding error message and exit. In all error cases just described, a standard help text is displayed along with the respective error message (see Appendix A).

In the current implementation, the following chart types can be specified as valid arguments:

- `line`: Creates a line chart with an x-axis, y-axis, one or more colour-coded data series, and, in the case of multiple data series, an optional legend.
- `bar`: Creates a bar chart with an x-axis, y-axis, one data series, and optional labels for the bars.
- `pie`: Creates a colour-coded pie chart with one data series, an optional legend, and optional labels for the pie segments.

The user can specify the input CSV file, from which the data for the chart shall be read, with the option `--dataset`. Without this option, AChart Creator loads a default CSV file as dataset, whose name is determined within the source code by a string constant in the main class. The option `--output` is used to specify the name of an output file for the SVG chart. If this option is not given, the chart will be saved in the directory of the input CSV file, and the name of the resulting SVG file is equal to that of the input file apart from the extension. If the input file name ends with `.csv`, this extension is replaced by `.svg` for the output file name, otherwise `.svg` is appended to the name of the input file. Both the input and the output file names may include a path. If no path or a relative path is specified, the programme will assume the current working directory as a base.

When producing the SVG output, the markup generated by D3 is made more human-readable by inserting line breaks and indentations. To avoid unnecessarily many digits of decimal precision, all the numbers calculated for the position and lengths of SVG elements are rounded by default to a precision of three decimal places. The number of decimal places can be specified using the command-line option `--svg-precision`. The option `--help` causes AChart Creator to display its standard help text, as listed in Appendix A, then exit. Similarly, the option `--version` makes the programme print its version information, then exit.

By default, AChart Creator tries to infer the titles of charts, axes, legends, and data series from the column headers provided in the CSV data. This automatic extraction of titles depends on the chart type and is explained in the subsections below. Using dedicated command-line options, however, the user can set most of the named titles manually, which always takes precedence over the default values. For this purpose, the following options are supported:

- `--chart-title`: Specifies the title of the chart.

- `--chart-desc`: Specifies an additional description of the chart (no default value available).
- `--x-axis-title`: Specifies the title of the x-axis for a line chart or bar chart. This option can be abbreviated to `--x-title`.
- `--y-axis-title`: Specifies the title of the y-axis for a line chart or bar chart. This option can be abbreviated to `--y-title`.
- `--legend-title`: Specifies the title of the legend for a multi-series line chart or a pie chart.

In most cases, all these titles are embedded into a <text> element and assigned the ARIA role heading. Thus, they are visible and can also be recognised by screen readers and AChart Interpreter. By contrast, the chart description is placed into a <desc> element and is only relevant for Describler, AChart Interpreter, and screen readers.

As explained in Section 5.2, the AChart taxonomy of ARIA roles and properties was composed with the intention of achieving the highest possible compatibility with Describler. Moreover, as Describler relies on setting the focus on the chart root and chart objects of interest, the corresponding elements produced by AChart Creator have their tabindex attribute set to `0`. Nevertheless, certain inconsistencies could not be avoided. For instance, Describler does not recognise any dataset or data series titles. For this reason, in the case of charts with multiple data series, each <g> element containing a data series is assigned the tabindex attribute as well, so it can be focused.

However, for each focusable element not known to Describler, the software retrieves the next descendant <title> element, not considering a possible higher-level <text> element or any element referenced with the aria-labelledby property. Navigating to a data series element therefore causes Describler not to read the data series title but instead the information of a descendant chart object, such as a data point value. To avoid this behaviour, a <title> element with the corresponding data series title is redundantly appended to each data series object, even if the title is already given by a legend item. The handling by browsers in combination with common screen readers was taken into account as well. As browsers do not recognise most of the ARIA roles and properties used by AChart Creator, the chart root and every chart object apart from axis items are assigned the standard ARIA property <aria-roledescription> with a short human-interpretable description in English.

This strategy produces charts which comply with the AChart system, but are also highly compatible with Describler, as well as most combinations of modern browsers and screen readers. Nevertheless, certain conflicts could not be resolved. For this reason, the command-line option `--target` can be used to optimise the accessibility markup for the desired assistive technology. When the option is set to `describler`, the chart title is redundantly appended as a <title> descendant to the root <svg> element, since Describler expects a <title> element at this position. In addition, a placeholder title is appended to the y-axis and the data series in case there is no title available. This placeholder title is “Values” for the y-axis and “Data series” for a data series. Lastly, in the case of multi-series line charts with a legend, each data point value is given an aria-labelledby property pointing to the legend item of the data series. This enables Describler to provide an enumeration of all the data points which are associated with a legend item. Setting the option `--target` to `screen-reader`, too, inserts the placeholder titles for the y-axis and the data series as just described. Moreover, it sets the aria-labelledby property of every data point differently, so that it references not only the elements with its names but also those with its values. This way, both the data point names and values can be recognised by screen readers when navigating to a data point. Setting `--target` to `achart` yields the default output format, that is, the AChart system with compatibility extensions as described above.

Certain pieces of information are embedded by a <title> element, which causes them to be displayed as a tooltip when setting the mouse pointer on the visible area of its respective ancestor element. This mainly affects the compatibility extensions for Describler explained above as well as tooltips intentionally generated for sighted users. All possible tooltips can be suppressed by specifying the command-line option `--no-tooltips`. This takes precedence over the `--target` option and therefore impairs the compatibility with Describler. Those data which are inserted only for accessibility purposes and should neither be permanently visible nor appear as tooltips are placed into `<desc>` elements.

In the following explanations, all rows and columns of the CSV input will be denoted by an index number starting with 0 and increasing from top to bottom or from left to right, respectively. For the case that the cells are not separated by a comma, the software tries to detect which character is used as the delimiter. In general, the CSV structure of the input file is parsed as follows. Row 0 is assumed to be a header row and column 0 is assumed to be a header column. In other words, column 0 contains the names of the data points, which are then used for the x-axis of a line or bar chart and for the legend of a pie chart, where the cell in row 0 is considered for the default x-axis or legend title. All subsequent columns are interpreted as one data series each, where the respective cell in row 0 represents the title of that dimension (data series). Row 0 is considered a header row, and all subsequent rows represent records (data points).

If the CSV structure contains only two columns, column 1 is visualised as a single data series. If more than two columns are present, the default behaviour depends on the chart type. In any case, the option `--column` can be used to determine a single column which shall be regarded as a single data series to visualise. This option expects a positive integer argument denoting the column, where `1` stands for column 1, that is, the first data series. If a non-positive argument or a value greater than the number of data series is given, the programme exits with a corresponding error message.

In the case of line and bar charts, the data points are plotted from left to right in increasing order by their names. If this order is not desired, the sorting can be suppressed by the option `--no-sort`, which will cause the data points to be plotted in the order that they are listed in the input CSV. In the following three subsections, the exact behaviour for line, bar, and pie charts is described.

```
Year,Price in Austria [€],Price in Germany [€],Price in Spain [€]
2011,230,250,150
2012,239,255.8,145.9
2013,241,261,143.4
2014,260,266.1,151
2015,270,275,159
2016,277,281.3,175.9
2017,280,285.2,189.4
2018,300,293,200
2019,310,305.3,213.1
```

Listing 6.2: Some sample data in CSV format used by AChart Creator as input to create the line charts shown in Figures 6.1 and 6.2. The CSV file contains nine data points in three dimensions, with both a header row and a header column. The data concerns prices for Austria, Germany, and Spain for each of nine years from 2011 to 2019.

6.1.1 Creating Line Charts

By default, all the data series contained in the given CSV file are visualised with one polyline (sequence of line segments connecting the data points) per series, where each of the polylines has a different colour. Each data point is drawn as a circle and contains a `<title>` element with its name and value shown as a tooltip when moving the mouse pointer over the circle. Moreover, two `<desc>` elements are included for exposing the name and the value as separate information to AChart Interpreter and Describler. If only one data series is present or the `--column` option is used, the value of the respective cell in row 0 is taken as the default visible y-axis title, and the data series title is empty. If multiple data series are visualised, each data series is labelled by a `<title>` element with the role `heading` and the content of the respective cell in row 0. This title is recognised by AChart Interpreter and becomes visible as a tooltip when moving the mouse pointer over a polyline. In this case, the y-axis is not given a default title. In addition, for multi-series line charts, a legend is created at the right margin of the chart. It shows one short line for each data series in its colour, along with the corresponding data series title. The title of the legend itself can be specified with the command-line option `--legend-title`; if this option is not given, it defaults to the word "Legend". The legend can be suppressed by specifying the command-line option `--no-legend`.

The x-axis title is embedded as a `<text>` element with role `heading`. The items of the y-axis are interpolated from the minimum and the maximum value of all data points, taking into account all data series shown in the chart. The chart title is composed as follows: for all columns represented in the chart, their respective cells in row 0 are concatenated with a comma and a space character (", "). Afterwards, the string " by ", and the x-axis title are appended. For example, "Amount 2015 by Fruit" for a single-series chart and "Amount 2013, Amount 2014, Amount 2015 by Fruit" for a three-series chart.

Listing 6.2 shows some fictitious sample data in CSV format concerning prices for Austria, Germany, and Spain for each of nine years from 2011 to 2019. Listing 6.3 shows (part of) the SVG source code of a single line chart created by AChart Creator from column 3 of the CSV file (Price in Spain) and Figure 6.1 shows the resulting graphic.

Listing 6.4 shows the relevant parts of the SVG source code of a multi-line chart created from all three data series in Listing 6.2, and Figure 6.2 shows the resulting graphic.

```
<svg xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1"
  viewBox="0 0 750 600" role="graphics-document">
  <style type="text/css"> ... </style>
  <rect id="backdrop" width="750" height="600" fill="#fff"/>

  <g id="ChartRoot" role="chart" tabindex="0" transform="translate(100,100)"
    aria-labelledby="title" aria-charttype="line"
    aria-roledescription="Line Chart">
    <rect role="chartarea" width="600" height="400" fill="none"/>
    <text id="title" role="heading" text-anchor="middle" font-size="14"
      x="275" y="-25">Price in Spain [€] by Year</text>

    <g id="xScale" role="xaxis" aria-roledescription="x-Axis"
      aria-labelledby="x-title" tabindex="0" aria-valuemin="2011"
      aria-valuemax="2019" transform="translate(0,400)" fill="none"
      font-size="10" font-family="sans-serif" text-anchor="middle">
      <text y="50" x="300" text-anchor="middle" fill="black" font-size="12"
        role="heading" id="x-title">Year</text>
      <path class="domain" stroke="currentColor" d="M0.5,6V0.5H600.5V6"/>

      <g class="tick" opacity="1" transform="translate(0.5,0)">
        <line stroke="currentColor" y2="6"/>
        <text fill="currentColor" y="9" dy="0.71em" role="axislabel"
          id="x1">2011</text>
      </g>

      <g class="tick" opacity="1" transform="translate(75.5,0)">
        <line stroke="currentColor" y2="6"/>
        <text fill="currentColor" y="9" dy="0.71em" role="axislabel"
          id="x2">2012</text>
      </g>

      ...
    </g>

    <g id="yScale" role="yaxis" aria-roledescription="y-Axis" tabindex="0"
      aria-valuemin="72" aria-valuemax="320" aria-labelledby="y-title"
      fill="none" font-size="10" font-family="sans-serif" text-anchor="end">
      <text transform="rotate(-90)" y="-38" x="-200" text-anchor="middle"
        fill="black" font-size="12" role="heading" id="y-title">
        Price in Spain [€]</text>
      <path class="domain" stroke="currentColor" d="M-6,400.5H0.5V0.5H-6"/>

      <g class="tick" opacity="1" transform="translate(0,387.597)">
        <line stroke="currentColor" x2="-6"/>
        <text fill="currentColor" x="-9" dy="0.32em" role="axislabel"
          id="y80">80</text>
      </g>
```

Listing 6.3: SVG code excerpt of a line chart with one data series created from column 3 of the CSV in Listing 6.2. Apart from `--column 3`, no other command-line options have been used. The accessibility markup corresponds to the AChart taxonomy with additional aria-roledescription properties and tooltips for the data points. Additional line breaks have manually been inserted to improve readability.

```

    <g class="tick" opacity="1" transform="translate(0,355.339)">
      <line stroke="currentColor" x2="-6"/>
      <text fill="currentColor" x="-9" dy="0.32em" role="axislabel"
        id="y100">100</text>
    </g>

    ...
  </g>

  <g id="dataarea1" role="dataset" aria-roledescription="Data Series">
    <path class="line" d="M0,274.194C25,276.613,50,279.032,75,280.806C100,
      282.581,125,284.839,150,284.839C175,284.839,200,276.774,225,
      272.581C250,268.387,275,266.371,300,259.677C325,252.984,350,
      240.591,375,232.419C400,224.247,425,217.124,450,210.645C475,204.167,
      500,199.919,525,193.548C550,187.177,575,179.798,600,172.419"
      stroke="#66c2a5"/>

    <g tabindex="0" role="datapoint" aria-labelledby="name1-1">
      <title>2011: 150</title>
      <desc role="heading" id="name1-1">2011</desc>
      <circle class="dot" cx="0" cy="274.194" r="5" fill="#66c2a5"/>
      <desc role="datavalue" id="value1-1">150</desc>
    </g>

    <g tabindex="0" role="datapoint" aria-labelledby="name1-2">
      <title>2012: 145.9</title>
      <desc role="heading" id="name1-2">2012</desc>
      <circle class="dot" cx="75" cy="280.806" r="5" fill="#66c2a5"/>
      <desc role="datavalue" id="value1-2">145.9</desc>
    </g>

    ...
  </g>

 </g>
</svg>
```

Listing 6.3 (cont.)**:** SVG code excerpt of a line chart with one data series created from column 3 of the CSV in Listing 6.2.

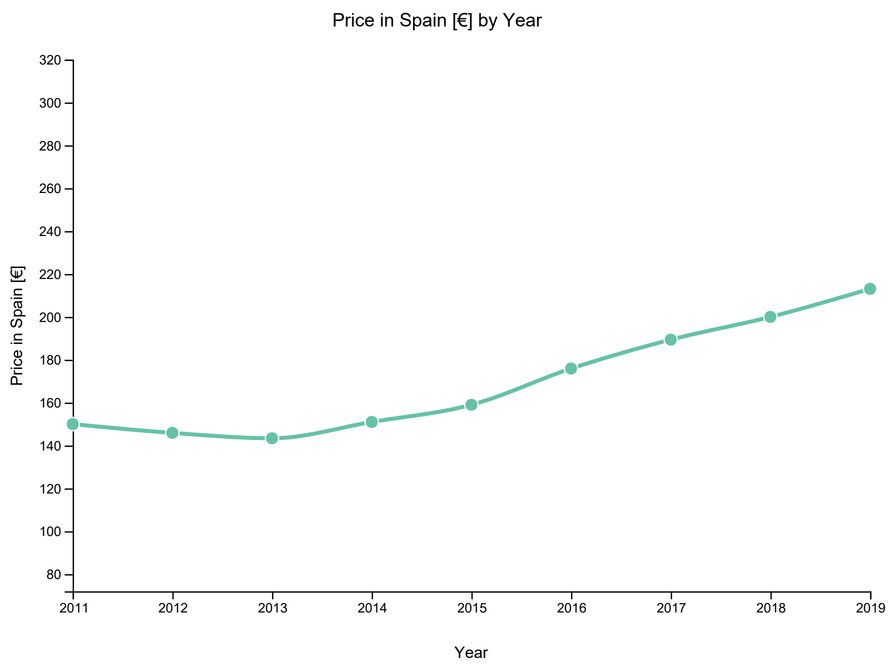


Figure 6.1: Line chart with one data series created from column 3 of the CSV in Listing 6.2. The chart, x-axis, and y-axis titles are visible. The data points are drawn as small circles. [Produced by the author of this thesis using AChart Creator.]

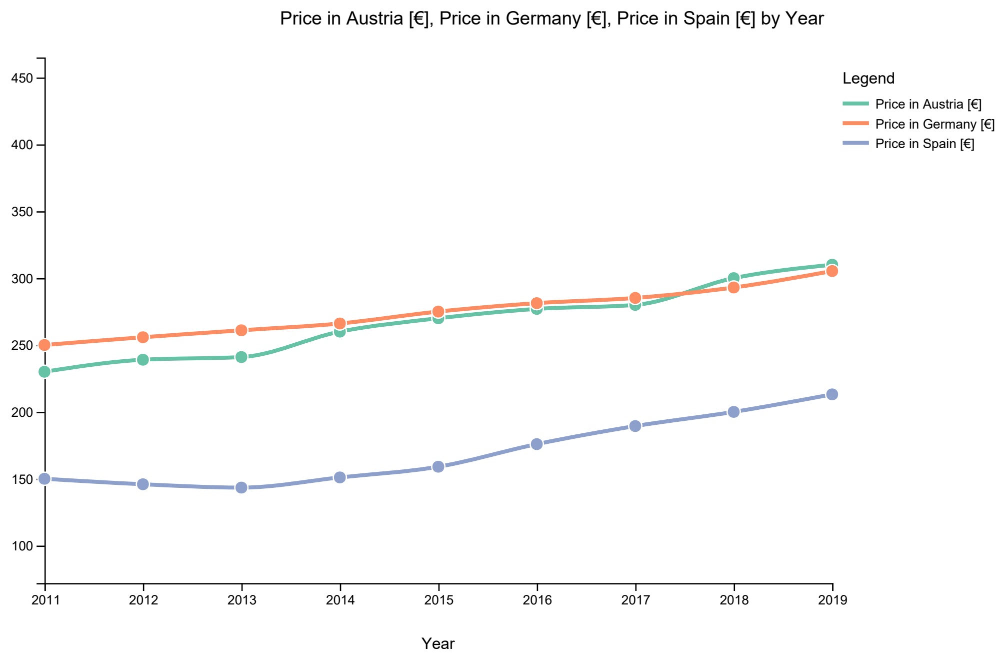


Figure 6.2: Line chart with three data series created from the CSV shown in Listing 6.2. The three lines have different colours which are assigned to the three corresponding data series titles in the legend to the right. The chart, x-axis, and legend titles are visible. The data points are drawn as small circles. [Produced by the author of this thesis using AChart Creator.]

```
<svg xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1"
  viewBox="0 0 750 600" role="graphics-document">
  ...
  <g id="ChartRoot" role="chart" tabindex="0" transform="translate(75,100)"
    aria-labelledby="title" aria-charttype="line" aria-roledescription="Line Chart">
    <rect role="chartarea" width="600" height="400" fill="none"/>
    <text id="title" role="heading" text-anchor="middle" font-size="14"
      x="387.5" y="-25">
      Price in Austria [€], Price in Germany [€], Price in Spain [€] by Year
    </text>

    <g id="xScale" role="xaxis" aria-roledescription="x-Axis"
      aria-labelledby="x-title" tabindex="0" aria-valuemin="2011"
      aria-valuemax="2019" transform="translate(0,400)" fill="none"
      font-size="10" font-family="sans-serif" text-anchor="middle">
      <text y="50" x="300" text-anchor="middle" fill="black" font-size="12"
        role="heading" id="x-title">Year</text>
      <path class="domain" stroke="currentColor" d="M0.5,6V0.5H600.5V6"/>

      <g class="tick" opacity="1" transform="translate(0.5,0)">
        <line stroke="currentColor" y2="6"/>
        <text fill="currentColor" y="9" dy="0.71em" role="axislabel"
          id="x1">2011</text>
      </g>

      <g class="tick" opacity="1" transform="translate(75.5,0)">
        <line stroke="currentColor" y2="6"/>
        <text fill="currentColor" y="9" dy="0.71em" role="axislabel"
          id="x2">2012</text>
      </g>

      ...
    </g>

    <g id="yScale" role="yaxis" aria-roledescription="y-Axis" tabindex="0"
      aria-valuemin="72" aria-valuemax="465" fill="none" font-size="10"
      font-family="sans-serif" text-anchor="end">
      <path class="domain" stroke="currentColor" d="M-6,400.5H0.5V0.5H-6"/>

      <g class="tick" opacity="1" transform="translate(0,372.001)">
        <line stroke="currentColor" x2="-6"/>
        <text fill="currentColor" x="-9" dy="0.32em" role="axislabel"
          id="y100">100</text>
      </g>

      <g class="tick" opacity="1" transform="translate(0,321.111)">
        <line stroke="currentColor" x2="-6"/>
        <text fill="currentColor" x="-9" dy="0.32em" role="axislabel"
          id="y150">150</text>
      </g>

      ...
    </g>
```

Listing 6.4: SVG code excerpt of a line chart with three data series created from the CSV in Listing 6.2. No command-line options have been used. The accessibility markup corresponds to the AChart taxonomy with additional aria-roledescription properties and tooltips for the data points. Additional line breaks have been manually inserted to improve readability.

```

    <g id="dataarea1" role="dataset" aria-roledescription="Data Series"
      tabindex="0" aria-labelledby="dataset-title1">
      <title role="heading" id="dataset-title1">Price in Austria [€]</title>
      <path class="line" d="M0,239.186C25,235.284,50,231.383,75,230.025C100,
        228.668,125,229.347,150,227.99C175,226.633,200,213.571,225,208.651C250,
        203.732,275,201.357,300,198.473C325,195.589,350,193.045,375,191.349C400,
        189.652,425,190.331,450,188.295C475,186.26,500,173.028,525,167.939C550,
        162.85,575,160.305,600,157.761" stroke="#66c2a5"/>

      <g tabindex="0" role="datapoint" aria-labelledby="name1-1">
        <title>2011: 230</title>
        <desc role="heading" id="name1-1">2011</desc>
        <circle class="dot" cx="0" cy="239.186" r="5" fill="#66c2a5"/>
        <desc role="datavalue" id="value1-1">230</desc>
      </g>

      <g tabindex="0" role="datapoint" aria-labelledby="name1-2">
        <title>2012: 239</title>
        <desc role="heading" id="name1-2">2012</desc>
        <circle class="dot" cx="75" cy="230.025" r="5" fill="#66c2a5"/>
        <desc role="datavalue" id="value1-2">239</desc>
      </g>

      ...
    </g>

    <g id="dataarea2" role="dataset" aria-roledescription="Data Series"
      tabindex="0" aria-labelledby="dataset-title2">
      <title role="heading" id="dataset-title2">Price in Germany [€]</title>

      ...
    </g>

    ...

    <g role="legend" aria-roledescription="Legend" font-size="10"
      font-family="sans-serif" text-anchor="start" tabindex="0"
      aria-labelledby="legend-title" transform="translate(608, 20)">
      <text role="heading" font-size="12" id="legend-title">Legend</text>

      <g role="legenditem" id="legenditem1" transform="..." tabindex="0">
        <line x2="20" style="stroke-width: 3;" stroke="#66c2a5"/>
        <text x="25" alignment-baseline="middle">Price in Austria [€]</text>
      </g>

      <g role="legenditem" id="legenditem2" transform="..." tabindex="0">
        <line x2="20" style="stroke-width: 3;" stroke="#fc8d62"/>
        <text x="25" alignment-baseline="middle">Price in Germany [€]</text>
      </g>

      ...
    </g>

  </g>
</svg>
```

Listing 6.4 (cont.)**:** SVG code excerpt of a line chart with three data series created from the CSV in Listing 6.2.

```
Fruit,Amount 2013,Amount 2014, Amount 2015
Apples,9,8,10
Bananas,20,22,28
Grapefruits,30,25,35
Lemons,8,14,50
Oranges,12,4,6
```

Listing 6.5: Some sample data in CSV format used by AChart Creator as input to create the bar chart shown in Figure 6.3 and the pie charts shown in Figures 6.4 and 6.5. The CSV file contains five data points in three dimensions, with both a header row and a header column. The data concerns amounts of five types of fruit in the years 2013, 2014, and 2015.

6.1.2 Creating Bar Charts

By default, column 1 of the CSV file is visualised, column 0 is considered to be a header column. If the `--column` option is used, the specified column is rendered instead. Each data point is represented by a vertical bar with its length proportional to its magnitude, along with a visible text label denoting its exact value. The label can be suppressed with the command-line option `--no-bar-values`. In this case, the data point value is available as a tooltip when moving the mouse over the corresponding bar. The y-axis title corresponds to the cell in the visualised column and row 0, and the data series title is empty. The items of the y-axis are interpolated from the minimum and maximum value of the data series. The chart title is then composed of the cell in the used column and row 0, concatenated with the string “ by ” and the x-axis title. An SVG code excerpt of an example chart generated from the fictitious CSV data in Listing 6.5 is presented in Listing 6.6. The resulting graphic can be seen in Figure 6.3.

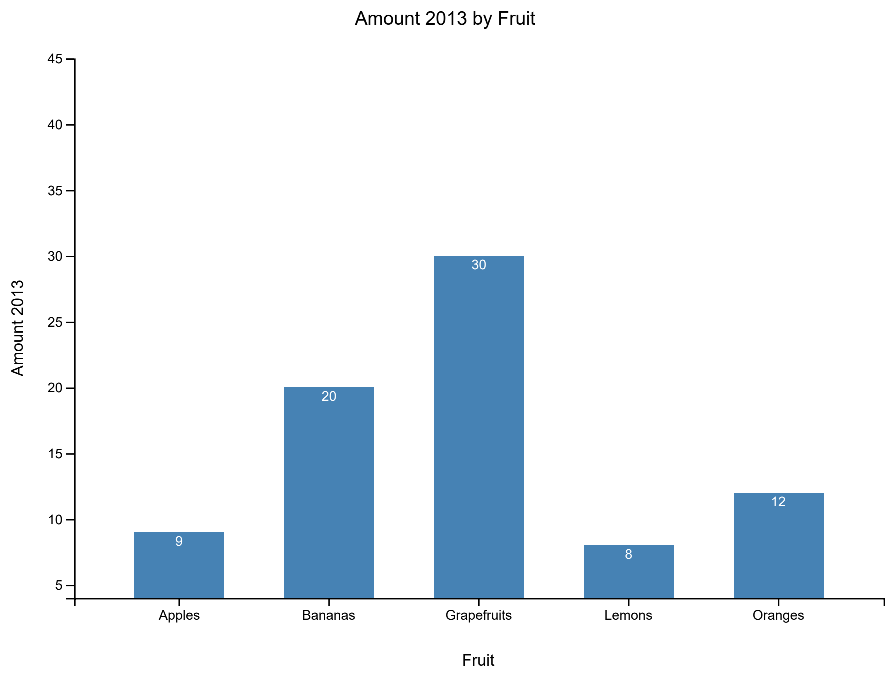


Figure 6.3: Bar chart with one data series created from column 1 of the CSV shown in Listing 6.5. The chart, x-axis, and y-axis titles are visible. The data points are drawn as vertical bars with a length proportional to their magnitudes and are annotated by their exact values. [Produced by the author of this thesis using AChart Creator.]

```
<svg xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1"
  viewBox="0 0 750 600" role="graphics-document">
  <style type="text/css">.bar {fill: steelblue; }</style>
  <rect id="backdrop" width="750" height="600" fill="#fff"/>

  <g id="ChartRoot" role="chart" tabindex="0" transform="translate(100,100)"
    aria-labelledby="title" aria-charttype="bar"
    aria-roledescription="Bar Chart">
    <rect role="chartarea" width="600" height="400" fill="none"/>
    <text id="title" role="heading" text-anchor="middle" font-size="14"
      x="275" y="-25">Amount 2013 by Fruit</text>

    <g id="xScale" role="xaxis" aria-axistype="category"
      aria-roledescription="x-Axis" aria-labelledby="x-title" tabindex="0"
      transform="translate(0,400)" fill="none" font-size="10"
      font-family="sans-serif" text-anchor="middle">
      <text y="50" x="300" text-anchor="middle" fill="black" font-size="12"
        role="heading" id="x-title">Fruit</text>
      <path class="domain" stroke="currentColor" d="M0.5,6V0.5H600.5V6"/>

      <g class="tick" opacity="1" transform="translate(77.778,0)">
        <line stroke="currentColor" y2="6"/>
        <text fill="currentColor" y="9" dy="0.71em" role="axislabel"
          id="x1">Apples</text>
      </g>

      <g class="tick" opacity="1" transform="translate(188.889,0)">
        <line stroke="currentColor" y2="6"/>
        <text fill="currentColor" y="9" dy="0.71em" role="axislabel"
          id="x2">Bananas</text>
      </g>

      ...
    </g>

    <g id="yScale" role="yaxis" aria-roledescription="y-Axis" tabindex="0"
      aria-valuemin="4" aria-valuemax="45" aria-labelledby="y-title"
      fill="none" font-size="10" font-family="sans-serif" text-anchor="end">
      <text transform="rotate(-90)" y="-38" x="-200" text-anchor="middle"
        fill="black" role="heading" id="y-title" font-size="12">
        Amount 2013</text>
      <path class="domain" stroke="currentColor" d="M-6,400.5H0.5V0.5H-6"/>

      <g class="tick" opacity="1" transform="translate(0,390.744)">
        <line stroke="currentColor" x2="-6"/>
        <text fill="currentColor" x="-9" dy="0.32em" role="axislabel"
          id="y1">5</text>
      </g>

      <g class="tick" opacity="1" transform="translate(0,341.963)">
        <line stroke="currentColor" x2="-6"/>
        <text fill="currentColor" x="-9" dy="0.32em" role="axislabel"
          id="y2">10</text>
      </g>
```

Listing 6.6: SVG code excerpt of a bar chart with one data series created from column 1 of the CSV shown in Listing 6.5. No command-line options have been used. The accessibility markup corresponds to the AChart taxonomy with additional aria-roledescription properties. Additional line breaks have manually been inserted to improve readability.

```

      ...
    </g>

    <g id="dataarea" role="dataset">

      <g tabindex="0" transform="translate(44.444,351.22)"
        role="datapoint" aria-labelledby="x1">
        <rect class="bar" width="66.667" height="48.78"/>
        <text x="33.334" y="10" text-anchor="middle" font-size="10"
          fill="white" role="datavalue" id="value1">9</text>
      </g>

      <g tabindex="0" transform="translate(155.556,243.902)"
        role="datapoint" aria-labelledby="x2">
        <rect class="bar" width="66.667" height="156.098"/>
        <text x="33.334" y="10" text-anchor="middle" font-size="10"
          fill="white" role="datavalue" id="value2">20</text>
      </g>

      ...
    </g>

  </g>
</svg>
```

Listing 6.6 (cont.)**:** SVG code excerpt of a bar chart with one data series created from column 1 of the CSV shown in Listing 6.5.

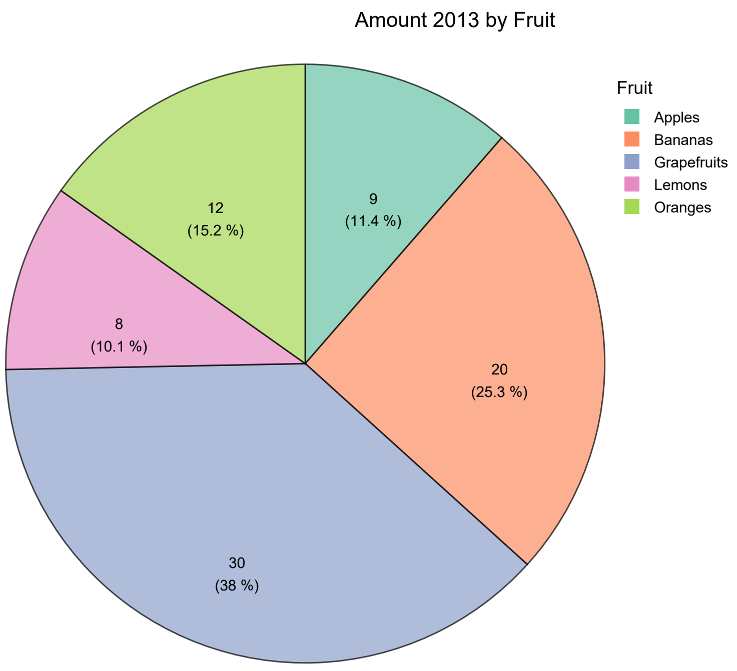


Figure 6.4: Pie chart with one data series and a legend created from column 1 of the CSV shown in Listing 6.5. The chart title is visible. The data points are depicted as pie segments of different colours with a size proportional to their magnitudes. Within each segment, the exact value and percentage are displayed. To the right of the chart, the legend shows the mapping of pie segment colours to corresponding data point names. The legend title "Fruit" is also visible. [Produced by the author of this thesis using AChart Creator.]

6.1.3 Creating Pie Charts

By default, column 1 of the CSV file is visualised, column 0 is considered to be a header column. If the `--column` option is used, the specified column is rendered instead. The data points are rendered as pie segments of different colours with sizes proportional to their magnitudes. The exact value of each data point is displayed using a `<text>` label within its corresponding segment. These labels can be visually hidden by using the command-line option `--no-segment-values`. In this case, each segment contains a descendant `<title>` element which displays the value of the data point as a tooltip when moving the mouse pointer over the pie segment. Below each label of a data point value, the corresponding percentage is shown, rounded to one decimal place by default. A different number of decimal places can be specified with the option `--segment-percentage-precision`. The percentage labels are suppressed with the option `--no-segment-percentages`, causing the percentages to be shown as tooltips.

To the right of the pie chart, a legend is generated which represents each pie segment with a small square in its colour next to the data point name. The legend title is visible and corresponds to the header cell of CSV column 0, unless overridden by the user. An SVG code excerpt of an example pie chart with a legend is shown in Listing 6.7. The resulting graphic can be seen in Figure 6.4.

The legend can be suppressed with the command-line option `--no-legend`. In this case, the names of the data points are added as labels to their respective pie segments, immediately above the positions of the labels for the data point values, as can be seen in Figure 6.5. However, this additional annotation is semantically treated as legend with its title embedded in a `<title>` element, as shown in Listing 6.8.

The data series title is always present only as a `<title>` element and corresponds to the cell in row 0 of the used column. The chart title is composed of the data series title, concatenated with the string " by " and the legend title.

```
<svg xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1"
  viewBox="0 0 750 600" role="graphics-document">
  <rect id="backdrop" width="700" height="550" fill="#fff"/>

  <g id="ChartRoot" role="chart" tabindex="0" transform="translate(50,100)"
    aria-labelledby="title" aria-charttype="pie"
    aria-roledescription="Pie Chart" font-size="10">
    <rect role="chartarea" width="400" height="400" fill="none"/>
    <text id="title" role="heading" text-anchor="middle" font-size="14"
      x="300" y="-25">Amount 2013 by Fruit</text>

    <g id="pie-chart" transform="translate(200,200)" role="dataset"
      aria-labelledby="dataset-title" tabindex="0">
      <title role="heading" id="dataset-title">Amount 2013</title>

      <g role="datapoint" tabindex="0" aria-labelledby="legenditem1">
        <path d="M1.225e-14,-200A200,200,0,0,1,131.245,-150.913L0,0Z"
          fill="#66c2a5" stroke="black" style="opacity: 0.7;"/>
        <text transform="translate(45.54,-121.762)" dy="15"
          style="text-anchor: middle;" role="datavalue" id="value1">9</text>
        <text id="percentage1" transform="translate(45.54,-121.762)"
          dy="30" style="text-anchor: middle;">(11.4 %)</text>
      </g>

      <g role="datapoint" tabindex="0" aria-labelledby="legenditem2">
        <path d="M131.245,-150.913A200,200,0,0,1,148.274,134.22L0,0Z"
          fill="#fc8d62" stroke="black" style="opacity: 0.7;"/>
        <text transform="translate(129.769,-7.75)" dy="15"
          style="text-anchor: middle;" role="datavalue" id="value2">20</text>
        <text id="percentage2" transform="translate(129.769,-7.75)"
          dy="30" style="text-anchor: middle;">(25.3 %)</text>
      </g>
      ...
    </g>

    <g role="legend" aria-roledescription="Legend" font-size="10"
      font-family="sans-serif" text-anchor="start" tabindex="0"
      aria-labelledby="legend-title" transform="translate(408, 20)">
      <text role="heading" font-size="12" id="legend-title">Fruit</text>

      <g role="legenditem" id="legenditem1"
        transform="translate(0,15)" tabindex="0">
        <rect x="5" y="-5" width="10" height="10" fill="#66c2a5"/>
        <text x="25" alignment-baseline="middle">Apples</text>
      </g>
      ...
    </g>

  </g>
</svg>
```

Listing 6.7: SVG code excerpt of a pie chart with one data series and a legend created from column 1 of the CSV shown in Listing 6.5. No command-line options have been used. The accessibility markup corresponds to the AChart taxonomy with additional aria-roledescription properties. Additional line breaks have manually been inserted to improve readability.

```
<svg xmlns="http://www.w3.org/2000/svg"
  xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1"
  viewBox="0 0 750 600" role="graphics-document">
  <rect id="backdrop" width="500" height="550" fill="#fff"/>

  <g id="ChartRoot" role="chart" tabindex="0" transform="translate(50,100)"
    aria-labelledby="title" aria-charttype="pie"
    aria-roledescription="Pie Chart" font-size="10">
    <rect role="chartarea" width="400" height="400" fill="none"/>
    <text id="title" role="heading" text-anchor="middle" font-size="14"
      x="200" y="-25">Amount 2013 by Fruit</text>

    <g id="pie-chart" transform="translate(200,200)" role="dataset"
      aria-labelledby="dataset-title" tabindex="0">
      <title role="heading" id="dataset-title">Amount 2013</title>

      <g role="datapoint" tabindex="0" aria-labelledby="legenditem1">
        <path d="M1.225e-14,-200A200,200,0,0,1,131.245,-150.913L0,0Z"
          fill="#66c2a5" stroke="black" style="opacity: 0.7;"/>
        <text transform="translate(45.54,-121.762)" dy="15"
          style="text-anchor: middle;" role="datavalue" id="value1">9</text>
        <text id="percentage1" transform="translate(45.54,-121.762)"
          dy="30" style="text-anchor: middle;">(11.4 %)</text>
      </g>

      <g role="datapoint" tabindex="0" aria-labelledby="legenditem2">
        <path d="M131.245,-150.913A200,200,0,0,1,148.274,134.22L0,0Z"
          fill="#fc8d62" stroke="black" style="opacity: 0.7;"/>
        <text transform="translate(129.769,-7.75)" dy="15"
          style="text-anchor: middle;" role="datavalue" id="value2">20</text>
        <text id="percentage2" transform="translate(129.769,-7.75)"
          dy="30" style="text-anchor: middle;">(25.3 %)</text>
      </g>
      ...
    </g>

    <g role="legend" aria-roledescription="Legend"
      aria-labelledby="legend-title" tabindex="0"
      transform="translate(200,200)">
      <desc role="heading" id="legend-title">Fruit</desc>

      <g role="legenditem" tabindex="0" id="legenditem1"
        transform="translate(45.54,-121.762)" style="text-anchor: middle;">
        <text>Apples</text>
      </g>

      <g role="legenditem" tabindex="0" id="legenditem2"
        transform="translate(129.769,-7.75)" style="text-anchor: middle;">
        <text>Bananas</text>
      </g>
      ...
    </g>

  </g>
</svg>
```

Listing 6.8: SVG code excerpt of a pie chart with one data series created from column 1 of the CSV shown in Listing 6.5. The chart has no visible legend. Instead, each pie segment is additionally labelled with its data point name, and these names are given the ARIA roles for legends as well. Apart from `--no-legend`, no other command-line options have been used. The accessibility markup corresponds to the AChart taxonomy with additional aria-roledescription properties. Additional line breaks have manually been inserted to improve readability.

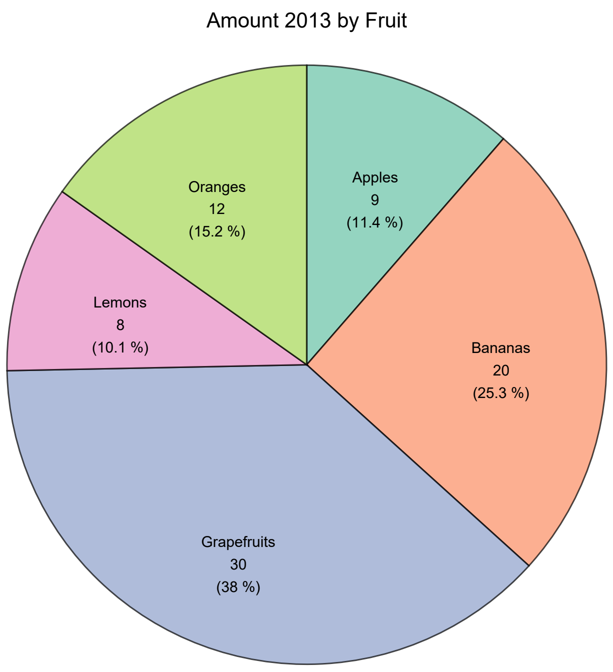


Figure 6.5: Pie chart with one data series created from column 1 of the CSV shown in Listing 6.5. The chart title is visible. The chart has no legend. The data points are depicted as pie segments of different colours with a size proportional to their magnitudes. Within each segment, the name, exact value, and percentage are displayed. [Produced by the author of this thesis using AChart Creator.]

6.2 Implementation

Chart authoring in D3 [Bostock 2021] takes place at a lower level of abstraction than is the case with the charting libraries described in Section 3.5. The chart author composes SVG structures like marks and axes into a chart, rather than specify the parameters of a chart object. For this reason, D3 cannot provide pre-canned accessibility features by itself, as the charting libraries do. Instead, it is the responsibility of the chart author to apply the D3 methods such that the resulting SVG is accessible, with a logical structure, textual elements at appropriate places, and ARIA attributes attached to suitable SVG elements. An example based on the SVG pseudo-list approach (see Subsection 4.5.5) is presented by Kopacz [2019]. The implementation of AChart Creator is another solution using D3 to create accessible charts.

To create SVG elements, D3 assumes that it is running inside a web browser with access to the browser's document object model (DOM). AChart Creator is written in TypeScript and, in its current form, uses Node.js [OpenJS 2021b] as its runtime environment instead of a browser. The `jsdom` library [jsdom 2021] is used to emulate a DOM tree inside Node.js, so that D3 can be used to compose a hierarchical structure of SVG nodes. The `nexe` [Boyd et al. 2020] package is then used to create self-contained binary executable files for various platforms.

The general procedure applied by AChart Creator can be summarised as follows. The specified CSV file is opened and read using the Node.js library `csvtojson`, whose method `fromFile()` is asynchronous and invokes a given callback function when the operation has been completed. The callback function is passed an array whose entries represent the rows of the CSV structure, starting with row 1. Each entry of the array contains a data structure whose member variables represent the cells in the respective row. The name of the variable is determined by the header of the cell's column, that is, the cell in the same column and row 0. The value of the variable corresponds to the content of the cell. The order in which these member variables are stored within each structure corresponds to the order of the CSV cells from left to right. This means, calling the `Object.keys()` method on one of these data structures returns an array whose entry 0 holds the content of row 0, column 0; entry 1 holds the content of row 0, column 1; and so forth. This way, all the columns can be programmatically referenced in their original order within the CSV.

After instantiating the `JSDOM` class and initialising the emulated DOM tree with a root `<svg>` element, AChart Creator obtains this root element by means of the d3 `select` method. Using the `attr()` method of D3, certain attributes of the element are set, such as `viewBox`, `version`, `xmlns`, and `role`. By means of D3's `append()` method, a `<g>` element for the chart root is appended to the root `<svg>` element and, in turn, the elements for the chart objects to the chart root. For each of the appended elements, certain attributes are set, including ARIA roles and properties, as well as `tabindex` and `id`. The text content of an element is specified with D3's `text()` method. Special D3 graphics methods are applied for drawing axes, lines, arcs, curves, and other shapes and for mapping the input data to their labels. After composing the desired SVG chart within the DOM tree, its content is serialised to a string of SVG markup and written to a file.

6.2.1 SVG Generator

The predecessor to AChart Creator was called SVG Generator, and was developed by Grass et al. [2019]. It creates accessible single-series line, bar, and pie charts in SVG format from CSV data by means of D3. This software comprises a set of three self-contained TypeScript modules named `line.ts`, `bar.ts`, and `pie.ts`, where each module creates one particular chart type. To initiate the creation of a chart, the module for the desired chart type is transpiled to a JavaScript file, which is then executed with Node.js using the given input CSV and output SVG file names. The base path for the CSV file is set to the subdirectory `source/data/csv/` of the project directory; the path for the SVG file is fixed to `build/svg/`. To automate the process of transpiling the TypeScript and launching Node.js with the resulting JavaScript, the authors provide a set of scripts for the task runner gulp.js [Bublitz and Schoffstall 2021]. The syntax for creating a chart by means of gulp is as follows:

```
gulp --compile TYPESCRIPT-FILE --dataset CSV-FILE
```

Afterwards, the created chart is located in the directory `build/svg/` and has the name `CHART-TYPE.svg`, where the output file name is determined by the gulp script. For example, the command:

```
gulp --compile line.ts --dataset fruit.csv
```

produces the SVG output file `build/svg/line.svg`, provided that the file `source/data/csv/fruit.csv` exists and contains valid CSV data.

The software considers column 0 of the input CSV structure as data point names and column 1 as the data series; any other columns are ignored. The cells of row 0 are used only to programmatically reference the columns, but not for any text output. The root `<svg>` element is given the ARIA role graphics-document. Apart from this assignment, the chart roots, axes, datasets, and data points are structured and annotated exactly according to the Describler system introduced in Subsection 4.5.1. In the case of pie charts, however, the legend represents a descendant element of the dataset, and the data points are appended to the chart root element. All the text labels for axis and legend items as well as for data points are automatically derived from the CSV data. The markup for the titles of the chart, axes, and legends as well as for a chart description is also inserted as in the Describler system. However, the text content is hard-coded within the source code and cannot be specified from the command line.

Although written in TypeScript, the three modules are designed as script files without any class, function, or type declarations. They each have a similar structure. First, the emulated DOM tree is initialised, the input and output file names are determined from the first two command-line arguments, and the CSV file is read. The entire data extraction and chart composition is placed into an anonymous callback function passed to the asynchronous fromFile() method for reading the CSV file. In addition to the three modules for line, bar, and pie charts, the authors provide a template file which can be used to write modules for other chart types. It contains the basic structure just described and can be filled with the type-specific instructions for drawing a particular kind of visualisation.

6.2.2 AChart Creator Version 1.0

While the original implementation of SVG Generator is capable of creating SVG charts with the expected visual representation and accessibility markup, the software architecture exhibits several drawbacks. Firstly, the transpilation of the source code at every invocation of the programme increases execution time and requires a TypeScript compiler to be installed on the user's platform. To automate the steps of transpiling, copying, and executing, gulp.js needs to be installed as well. The application of gulp scripts for the mere execution, in turn, complicates the compilation of the Node.js project to a standalone binary file, since gulp.js is mainly intended as a task runner during development.

Moreover, considerable blocks of source code are duplicated across the three modules, such as those for initialising the DOM tree, reading the CSV, extracting the column headers, and creating the elements of the basic SVG structure. In the context of this thesis, this architecture was transferred to a class-based design, where the duplicated code blocks mentioned above were moved to dedicated central classes used for all chart types. The main class AChartCreator handles all command-line arguments and creates an instance for one of the three supported chart types. The abstract class Chart defines values and elements common among all the chart types and inherited by their classes. This way, AChart Creator can be transpiled to one JavaScript project, which can then be executed without gulp or any repeated transpilations and can be compiled into a binary executable file.

With regard to the handling of CSV data, SVG Generator only considers the columns 0 and 1. In order to facilitate the creation of charts from other columns as well, the `--column` option was introduced, which lets the user choose an arbitrary column from those available to be interpreted as a data series. In addition, the production of line charts with multiple data series was implemented, where each CSV column starting from 1 is represented as a separate polyline within the same coordinate system and encoded with a different colour. Each of these polylines is assigned the ARIA role dataset and is given a descendant `<title>` element with the ARIA role heading for the data series title. In addition, the creation of a legend was implemented, which lists the titles of all the data series along with a short line in its colour. The same kind of legend was also added to the pie charts, where squares in the colour of each pie segment represent the corresponding data points and are given a label with the name. The segments of a pie chart are not only labelled with the absolute value of the data point, but also with its percentage. Several command-line options were added which let the user choose the exact visual appearance of the charts with regard to legends and annotations of data points.

Tests with several different CSV files revealed two inconsistencies of SVG Generator. For example, it drew the data points from left to right according to the order they are listed in the CSV file from top to bottom, which assumes that the CSV rows are sorted in a suitable manner. If, however, the rows are listed in an arbitrary order, this behaviour may lead to an unintended result, especially in the case of numerical or time-based names. In AChart Creator version 1.0, the data points are therefore sorted in increasing order by their names before generating the SVG. The sorting includes both numerical values and strings, which are sorted alphabetically. This is especially useful in the case of dates or times expressed as string values. If desired, the original order can be applied by giving the command-line option `--no-sort`.

Another problematic issue in SVG Generator was that the `id` values for x-axis labels, legend items, and data points were automatically generated, including the verbatim content of the respective CSV cells. While this solution works if the cells contain only alphanumerical strings, other characters such as the full stop (.) or white space ( ) can cause errors when querying for these ids using the JavaScript selector methods. For this reason, the generation of `id` attributes was changed so that increasing index numbers independent of the cell contents were used instead.

Moreover, SVG Generator did not provide any useful titles or descriptions, neither visible nor hidden, but only the necessary markup to embed them within the graphics document, filled with constant strings. The titles for the x-axis of bar charts and the y-axis of line charts were not visible; in the former case because it was only included by the `<title>` element, in the latter case because the `<text>` element was positioned outside the margins of the SVG document. The feature to set the chart title and description via command-line options was introduced by Inti Gabriel Mendoza Estrada as a part of a seminar project, along with the possibility to specify the x-axis and y-axis titles in the case of line charts. In the context of this thesis, the command-line setting of axis titles was then extended to bar charts, and the possibility to denote a legend title was added. Furthermore, the automatic derivation of default titles for charts, axes, legends, and/or data series from the CSV column headers was implemented according to the specification in Section 6.1. Lastly, all axis titles were made visible by embedding them in `<text>` elements and positioning them within the boundaries of the SVG document.

6.2.3 AChart Creator Version 2.0

While the accessibility markup of the line and bar charts created by SVG Generator and AChart Creator version 1.0 mostly conform to the Describler system, the pie charts lack the roles for the legend and its titles. For version 2.0, these roles were added, along with the role `heading` for the legend title. Moreover, the `<path>` elements representing the data points are appended to the chart root element instead of the dataset, which prevents them from being found by Describler or AChart Interpreter. For this reason, the data points were moved into the `<g>` element with the role `dataset`. In addition, the changes to the Describler system proposed in Section 5.3 were applied.

The current version 2.0 of AChart Creator consists of seven classes. The classes `LineChart`, `BarChart`, and `PieChart` are used to produce the three corresponding chart types. All three are derived from the abstract superclass `Chart`, which declares several common instance variables and provides the method `init()` for preparing the data and generating a basic structure of the SVG document. Each of the three subclasses implements a method named `create()` which performs the main tasks of composing the chart. Both methods expect an array of extracted CSV rows, a structure of chart metadata, and the root `<svg>` element of the DOM tree as arguments. The chart metadata structure contains members representing all the parameters which can be specified from the command line, such as the chart title and description, the x-axis, y-axis, and legend titles, the column to visualise, the target accessibility software, the precisions for rounding decimal numbers, and several booleans expressing whether a particular feature should be present. It should be noted that none of these classes have a constructor method, because at the time of instantiation, the CSV data are not yet available.

In the `Chart` superclass, instance constants are first defined for the height, width, margins, and certain positions, and variables are declared for other distances as well as for possible global CSS definitions and for storing certain elements of the basic SVG structure. The instance variable `names_columns` of type `string` array is intended to store the header of all the names columns, that is, in version 2.0, the CSV content of the cell in row 0 and column 0. The variable `values_columns` of type `string` array holds the headers of all data series to visualise, that is, the contents of the cells in row 0 and one or more of the columns with numbers greater than 0. The headers serve two purposes: firstly, they are used to reference the member variables in a data structure which represents a CSV row, secondly they provide the basis for automatically composing the default titles.

The method `init()` of the `Chart` class extracts the column headers from the passed data array and stores them into the instance variables just described. If a particular column has been specified for visualisation, the header of this column is stored as the only entry in the `values_columns` array; otherwise, `values_columns` is set to the headers of all extracted columns starting at 1. Afterwards, the `init()` method tests each title to see if it has been set by the user and, if not, infers the particular default title from the CSV headers. The next step is to test for all the names and values of each data point whether they are numeric or contain other characters. In the former case, the `string` content is converted to the type `number`, which is necessary for correctly sorting the data points and for calculating the minimum and maximum values of numerical axis. If among the data point values any non-numerical content is detected, the programme exits with an error message, since it was decided that all values need to be numerical. Finally, the array of CSV rows is sorted by their names, unless the user has requested not to do so with the `--no-sort` option.

After these steps of data preparation, it is determined whether to print a legend to the right of the chart. This is the case if there are multiple data series or the chart is a pie chart and if the user has not suppressed the legend with the `--no-legend` command-line flag. Depending on this decision, the horizontal position of the chart is calculated: if no legend is present, the chart is centred within the SVG document, otherwise it is shifted to the left. Then, a basic structure of the graphics document is created, including the background, the chart root, as well as the chart title and description, and certain attributes are set which are independent of the chart type. Apart from the `init()` method just described, the `Chart` class contains a method for creating the legend and methods for rounding numbers according to the desired number of decimal places.

Each of the `LineChart`, `BarChart`, and `PieChart` classes starts with the initialisation of other instance constants for the dimensions and positions of the chart; some also define global CSS. As one of its first steps, the method `create()` of the respective instance calls the `init()` method inherited from the `Chart` superclass just described. In the case of bar and pie charts, however, it first tests whether the user has specified a particular data column. If not, the corresponding member variable of the chart metadata is set to 1, so that the superclass method only considers this column for title derivation. This is necessary since both classes currently support only visualising a single data series. After the superclass `init()` method has completed, certain attributes are added to the chart elements already created, for example the ARIA property `aria-charttype` to the chart root, and the D3 methods for drawing the chart are applied. Although the DOM structures and the processing of data exhibit many similarities across the different chart types, it was decided to leave these parts of the implementation in separate classes, as they differ slightly in the graphics methods used. Finally, the methods return a string containing the serialised SVG structure.

Apart from the classes for creating charts, there are three central classes for performing general tasks. The main class is called `AChartCreator` and is part of the module `achart-creator`. In this module, the emulated DOM tree is first initialised. Then, the class `AChartCreator` is instantiated. This class declares two string constants for the default input file names (one for line charts and a different one for other chart types) as well as instance variables for the input and output file names given by the user, the requested chart type, and the specified chart metadata (titles, description, CSV column and others) as described above. Moreover, the variable `chart` of type `Chart` is declared to hold an instance of the class for the selected chart type.

The AChartCreator constructor calls the method parseCommandLine() of the same instance as a first step. In a switch block, this method evaluates the command-line arguments, sets the respective instance variables and chart metadata members accordingly, and outputs a message in case of syntax errors. All messages are composed of string constants defined in the static class Text. If no syntax errors have been detected, the argument of the mandatory --chart parameter is evaluated in a separate switch block. In the default case of no known chart type, the programme exits with an error message. Otherwise, an instance of the respective class for the chosen chart type is created and stored in the instance variable chart.

After handling the command-line input, the constructor outputs an initial message to stdout and then calls the method loadFile() of the static class FileHelper. This method first tests if an input and an output file name have been specified by the user. If no output file name has been given, it is derived from the input file name by detecting whether the latter ends with the extension .csv, removing such an extension if present, and appending the extension .svg. Afterwards, FileHelper.loadFile() handles the interaction with the csvtojson library, outputs a text message if the CSV file has been successfully loaded, and then passes the array with the CSV data to the method createChart() of the AChartCreator instance. This method eventually initiates chart creation: it calls the create() method of the instance stored in the variable chart, passing it the CSV data and the chart metadata specified by the user. The returned SVG markup string is handed to the method FileHelper.writeSvg(), which outputs a final confirmation message to stdout and saves the markup string to an SVG file with the appropriate name. Since the jsdom library serialises the DOM content without any line breaks or indentations, the markup is passed to the library xml-formatter before saving in order to improve its readability.

Chapter 7

AChart Interpreter

Accessible Chart Interpreter (AChart Interpreter) is a web application for analysing charts in Scalable Vector Graphics (SVG) format and presenting the results of this analysis in some form of textual output, such as synthetic speech. In other words, AChart Interpreter is a kind of screen reader for charts. For correct interpretation by the software, the SVG charts must contain semantic annotations using attributes based on the Accessible Rich Internet Applications (ARIA) system. In particular, AChart Interpreter can handle the AChart roles and properties presented in Section 5.3. Currently, AChart Interpreter supports bar charts, line charts, and pie charts with an arbitrary number of data series.

The development of AChart Interpreter was inspired by Describler (see Section 3.6 and Subsection 4.5.1). AChart Interpreter was motivated by several aspects. Firstly, it represents a tool for developers and chart authors, which can be used to test the compliance of SVG charts with specific ARIA markup. AChart Interpreter's graphical user interface (GUI) provides a split-screen view of the chart and the derived textual representation. Moreover, the application is screen reader compatible and thus can supplement existing general-purpose screen readers when it comes to exploring accessible SVG charts. In case no screen reader is available, or for use by sighted users, AChart Interpreter provides optional integrated speech output.

AChart Interpreter runs in all modern web browsers, meaning those with support for HTML 5 and ECMAScript version 5. Furthermore, it can be compiled to self-contained, standalone binary executable packages for various platforms with Electron [OpenJS 2021a]. The web-based approach was chosen so the software can be used not only as a self-contained application, but in future potentially as a browser extension for reading embedded SVG charts.

The software is open-source under an MIT licence and is available at [Kopel, Andrews, Mendoza Estrada, Bodner et al. 2021a]. A live demo of AChart Interpreter is available at [Kopel, Andrews, Mendoza Estrada, Bodner et al. 2021b]. A showcase video of AChart Interpreter was created by Perko [2021]. A showcase video of an earlier version of the software, then still called AChart Reader, was made by Bodner et al. [2020b].

Finally, a spin-off command-line tool, called AChart Summariser, can output a textual summary of an accessible SVG chart as plain text. With the tool, it is possible to perform automatic sequential analysis of multiple charts using shell scripts and programmatically processing the output.

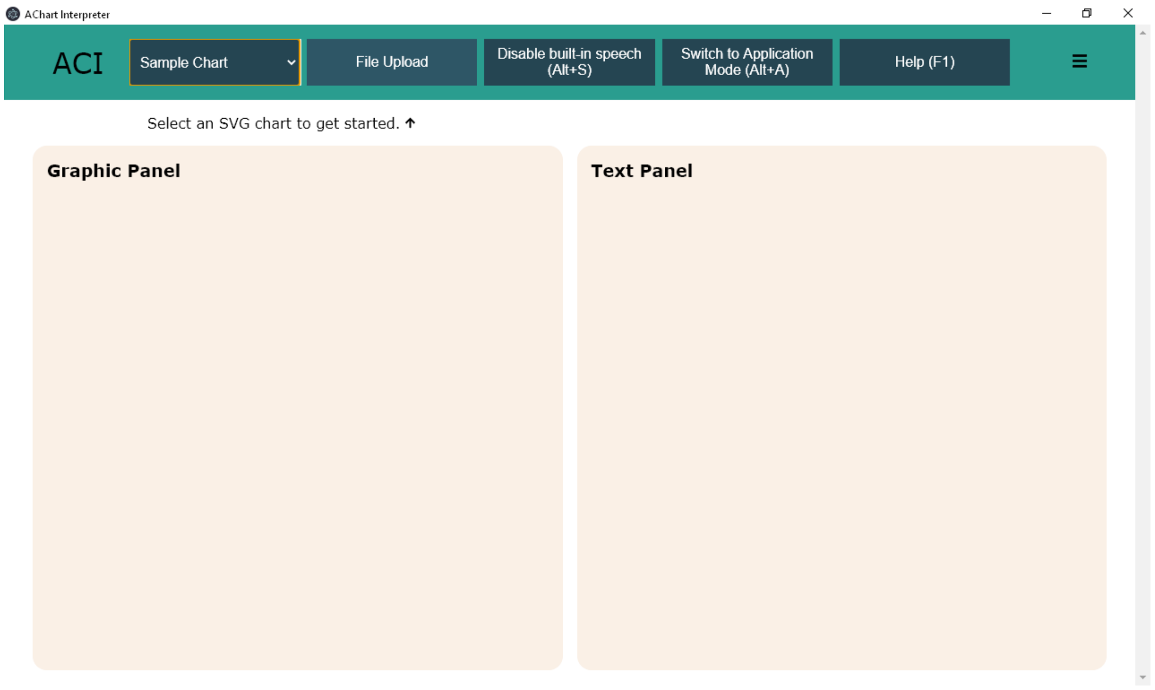

AChart Interpreter
ACI
Sample Chart
File Upload
Disable built-in speech (Alt+S)
Switch to Application Mode (Alt+A)
Help (F1)
Select an SVG chart to get started. ↑
Graphic Panel
Text Panel

Figure 7.1: AChart Interpreter's main window after launch. Both the Graphic Panel and Text Panel are initially empty. [Screenshot taken by the author of this thesis.]

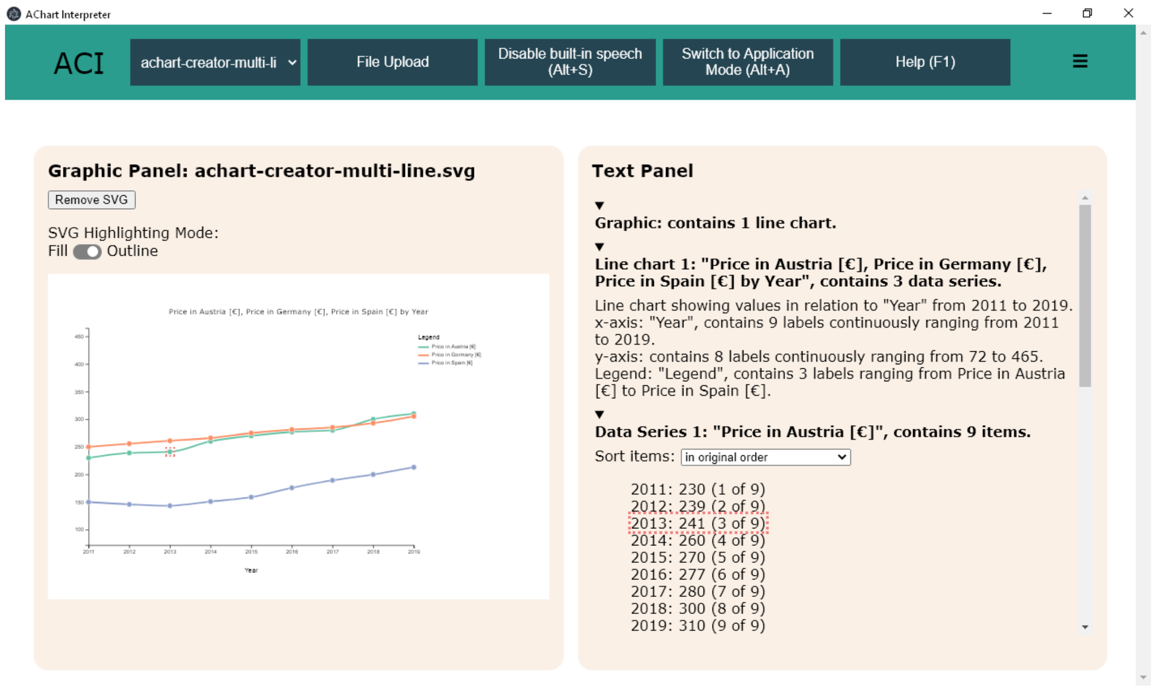
7.1 User Interface

The GUI of AChart Interpreter consists of three major components, namely the Main Menu and two synchronised panels, the Graphic Panel on the left and the Text Panel on the right. After launch, both panels are empty, as can be seen in Figure 7.1. Between the menu and the panels, a short placeholder text is displayed, prompting the user to "Select an SVG chart to get started.". For screen reader users, the GUI starts with the objects of the menu at the top, followed by the placeholder text, the Graphic Panel, and then the Text Panel.

An accessible SVG chart can be opened in AChart Interpreter using the File Upload button in the main menu. Alternatively, one of several sample SVG charts provided along with AChart Interpreter can be loaded via the Sample Chart drop-down menu. The design is responsive, so that on narrower screens, the main menu is collapsed behind a menu button ("hamburger icon").

Once an accessible SVG chart has been loaded, it is displayed in the two synchronised panels. Figure 7.2 shows the GUI of AChart Interpreter with the sample three-series line chart from Listing 6.4. The Graphic Panel displays the chart graphically. The file name of the current chart is shown to the right of the panel's heading "Graphic Panel". A button Remove SVG is available to remove the chart and a switch allows toggling between the two forms of highlighting the currently focused chart object: fill and outline (the default). Changing the highlighting mode can be helpful if the current manner of highlighting is not well-suited to a particular chart. The Text Panel displays a structured textual representation of the chart (the so-called *chart tree*), garnered from the semantic annotations and which is suitable for reading out. This includes a title and description, if included in the chart, a basic automatically-generated description of its content, the titles and items of its axes and legends, the titles of its data series, and the names and values of all its data points. This way, a developer or chart author can verify that the SVG chart's structure and ARIA markup conform to AChart's taxonomy. Clicking on an element of the chart in one panel selects, focuses, and highlights it in both panels.

From a screen reader's point of view, the Graphic Panel's heading is placed below the menu, followed by the Remove SVG button and the switch for toggling the highlighting mode, which is exposed to screen readers as a checkbox. If the checkbox is enabled, highlighting mode is set to outline. Immediately



Graphic Panel: achart-creator-multi-line.svg

SVG Highlighting Mode:
Fill Outline

Text Panel

▾
Graphic: contains 1 line chart.

▾
Line chart 1: "Price in Austria [€], Price in Germany [€], Price in Spain [€] by Year", contains 3 data series.

Line chart showing values in relation to "Year" from 2011 to 2019.
x-axis: "Year", contains 9 labels continuously ranging from 2011 to 2019.
y-axis: contains 8 labels continuously ranging from 72 to 465.
Legend: "Legend", contains 3 labels ranging from Price in Austria [€] to Price in Spain [€].

▾
Data Series 1: "Price in Austria [€]", contains 9 items.

Sort items: in original order

2011: 230 (1 of 9)
2012: 239 (2 of 9)
2013: 241 (3 of 9)
2014: 260 (4 of 9)
2015: 270 (5 of 9)
2016: 277 (6 of 9)
2017: 280 (7 of 9)
2018: 300 (8 of 9)
2019: 310 (9 of 9)

Figure 7.2: AChart Interpreter displaying the sample SVG multi-line chart from Listing 6.4. The graphical version on the left shows three data series. The structured textual output on the right lists the chart's title, descriptions of the chart, the legend, the x- and y-axes, and the three data series with their titles and all contained data points. Data point 3 of data series 1 is currently selected. [Screenshot taken by the author of this thesis.]

below, screen reader users find the heading of the Text Panel, followed by all its contents. The graphical content is hidden for screen readers, and the `Tab` navigation is modified, so that graphical objects do not receive keyboard focus. This form of presentation was chosen because all the accessible information of the graphic is available in the Text Panel and, this way, more convenient keyboard navigation between the Main Menu and the Text Panel is possible.

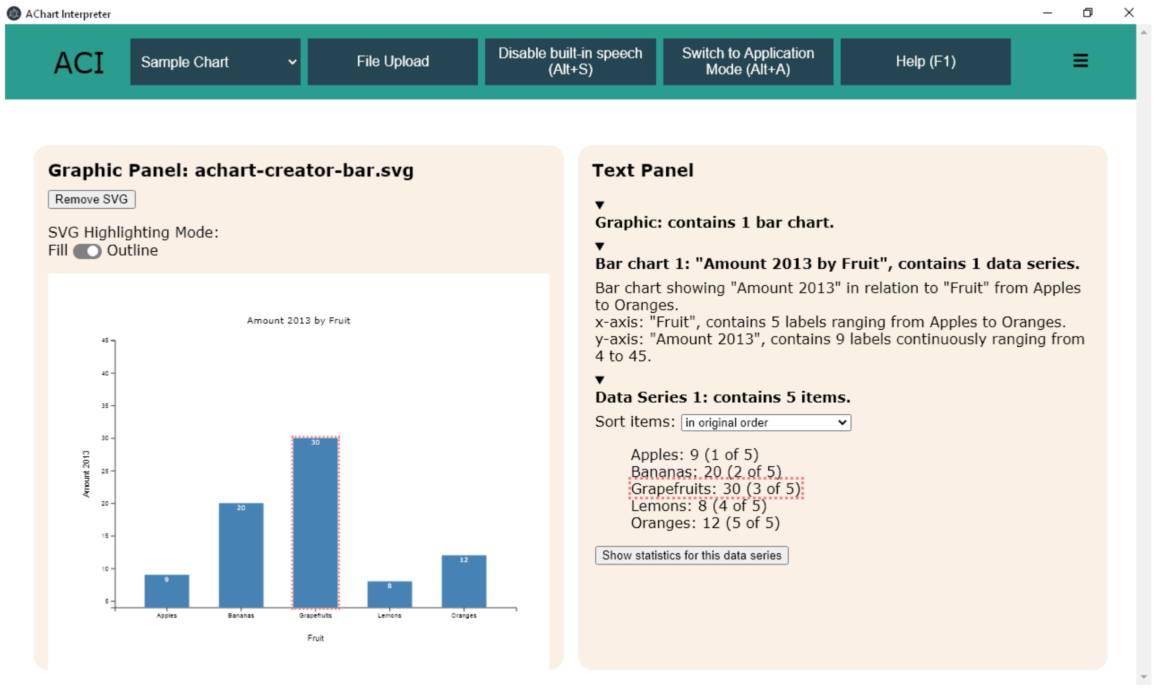
To the right of the two controls for opening an SVG file described above, the Main Menu provides three further buttons, each with a keyboard equivalent:

- Built-in speech (Alt+S): Toggles integrated speech output off or on. Depending on the current state, the button text says "Disable built-in speech (Alt+S)" or "Enable built-in speech (Alt+S)". In case that no synthesiser can be detected it reads "No built-in speech available".
- Application Mode (Alt+A): Toggles screen reader interaction between application mode and document Mode (see Subsection 7.1.3). Depending on the current state, the button text says "Switch to Application Mode (Alt+A)" or "Switch to Document Mode (Alt+A)".
- Help (F1): Opens a scrollable modal window showing the application's help text, which is included in Appendix B. The help window can be closed by clicking the Help (F1) button again, clicking the "Close (ESC)" button in the upper right corner of the dialogue window, or by pressing `Escape`.

In addition to line charts, AChart Interpreter supports bar charts and pie charts, as shown in Figures 7.3 and Figures 7.4, respectively.

7.1.1 Chart Accessibility Tree (CAT)

Since no official system for annotating chart elements has yet been standardised, current web browsers neither detect semantically enriched objects contained in accessible SVG charts, nor enter and maintain them as such in their standard accessibility tree (see Subsection 2.4.3).



Graphic Panel: achart-creator-bar.svg

SVG Highlighting Mode:
Fill Outline

Text Panel

▼
Graphic: contains 1 bar chart.

▼
Bar chart 1: "Amount 2013 by Fruit", contains 1 data series.

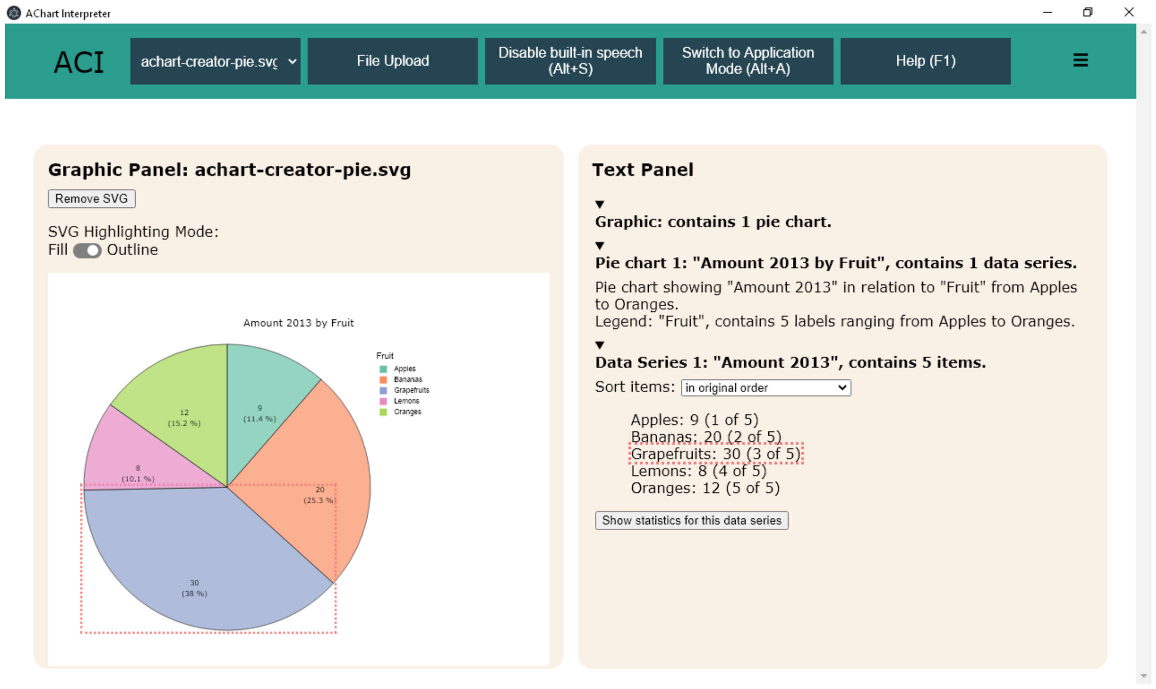
Bar chart showing "Amount 2013" in relation to "Fruit" from Apples to Oranges.
x-axis: "Fruit", contains 5 labels ranging from Apples to Oranges.
y-axis: "Amount 2013", contains 9 labels continuously ranging from 4 to 45.

▼
Data Series 1: contains 5 items.

Sort items: in original order

Apples: 9 (1 of 5)
Bananas: 20 (2 of 5)
Grapefruits: 30 (3 of 5)
Lemons: 8 (4 of 5)
Oranges: 12 (5 of 5)

Figure 7.3: AChart Interpreter displaying the sample SVG bar chart from Listing 6.6. The graphical version of the bar chart is shown on the left. The structured textual output on the right lists the chart's title, descriptions of the chart, the x- and y-axes, and the data series with its title and all contained data points. Data point 3 has been selected. [Screenshot taken by the author of this thesis.]



Graphic Panel: achart-creator-pie.svg

SVG Highlighting Mode:
Fill Outline

Text Panel

▼
Graphic: contains 1 pie chart.

▼
Pie chart 1: "Amount 2013 by Fruit", contains 1 data series.

Pie chart showing "Amount 2013" in relation to "Fruit" from Apples to Oranges.
Legend: "Fruit", contains 5 labels ranging from Apples to Oranges.

▼
Data Series 1: "Amount 2013", contains 5 items.

Sort items: in original order

Apples: 9 (1 of 5)
Bananas: 20 (2 of 5)
Grapefruits: 30 (3 of 5)
Lemons: 8 (4 of 5)
Oranges: 12 (5 of 5)

Figure 7.4: AChart Interpreter displaying the sample SVG pie chart from Listing 6.7. The graphical version of the pie chart is shown on the left. The structured textual output on the right lists the chart's title, descriptions of the chart and the legend, and the data series with its title and all contained data points. Data point 3 has been selected. [Screenshot taken by the author of this thesis.]

AChart Interpreter recognises and processes chart elements annotated according to the AChart taxonomy. While interpreting an accessible SVG chart, AChart Interpreter constructs its own internal hierarchical representation of the chart, called the Chart Accessibility Tree (CAT), which reflects the nesting of the corresponding SVG elements in the chart. Similar to the standard accessibility tree of web browsers, every graphics object with semantic significance is represented by a corresponding tree node along with its computed alternative text.

Dedicated classes are defined for all major chart objects, such as the chart root, axes, and data series. Instances of these classes constitute nodes of the CAT. Minor chart objects without any descendants, such as titles, axis labels, and data point values, are stored in ordinary member variables of type `number` or `string`. Thus, the hierarchical structure of chart objects is reflected by the respective memberships of the instances and variables. For example, an instance of class `SVGDocument` holds one or more instances of class `Chart`; each instance of class `Chart` holds one or more instances of class `Axis` and one or more instances of class `Dataset`; each instance of class `Axis` has member variables for the axis title, the axis variable, and the minimum and the maximum value, as well as its contained axis labels; and so forth. Further details of this architecture will be presented in Subsection 7.2.2.

As soon as the analysis of the SVG chart has been completed, the CAT for the chart is visualised in the `Text Panel` as an expandable tree of nodes with associated text strings. This visualisation of the CAT is known as the *chart tree*. A triangular icon next to every node allows it to be expanded or collapsed. Alternatively, every node can be expanded or collapsed by the `Enter` or the `Space` key when focused. The `Text Panel` becomes scrollable if the chart tree requires more vertical space. Any object in the chart tree can be focused by `Tab` navigation or by left-clicking it, which also selects and focuses the same chart element in the `Graphic Panel` (and vice versa). Focus in the `Text Panel` is indicated by a dotted outline around the object in focus. The order of the chart tree nodes is the same for the visual arrangement and the screen reader navigation from top to bottom as well as for forward `Tab` navigation.

Most chart tree objects have an equivalent *textual summary*, which is constructed dynamically from the nodes in the CAT, as follows:

1. The type of the object, followed by a colon (:) and a space character. For certain object types, a space character and an index number starting at 1 are placed between the type and the colon. For example, “Line chart 1: ”

2. The title of the object, if provided by the chart’s author, within quotation marks ("), followed by a comma and a space character. For example, “Line chart 1: "Price in Austria [€], Price in Germany [€], Price in Spain [€] by Year", ”

3. The string “contains ”, followed by the number and type of descendant objects, delimited by a space character. The type is expressed in its singular form if only one descendant object is present, otherwise in its plural form. The text concludes with a full stop (.) character. For example, “Line chart 1: "Price in Austria [€], Price in Germany [€], Price in Spain [€] by Year", contains 3 data series.”

Subsequent discussion for particular chart tree objects which conform to the text pattern above will omit the details about space and punctuation characters. A detailed explanation of the implementation of textual summary and description composition is provided in Subsection 7.2.3.

The root object of the chart tree corresponds to the root `<svg>` element. Its textual summary starts with the word “Graphic” and any title assigned to the entire graphics document. Afterwards, the number and type of contained charts is stated. If charts of distinct types are present, each of these chart types is enumerated with its number of occurrences, separated by the string “ and ”. For example: “Graphic: contains 1 bar chart and 2 line charts.”. If an overall description has been provided for the graphics document in a `<desc>` element, it is shown immediately below the root. Charts are always represented as descendant objects of the graphics root, even if the `<svg>` element has been assigned the role `chart` and, thus, the logical chart root is identical to the root `<svg>` element. This is one of few cases in which both the CAT and the visual chart tree may differ from the hierarchy of the SVG structure.

The textual summary of a chart root begins with the chart type, followed by the string " chart " and an index number. If a chart title has been provided by the author, it is displayed afterwards. If applicable, the number of contained data series is denoted along with the expression "data series". For example: "Line chart 1: "Price in Austria [€], Price in Germany [€], Price in Spain [€] by Year", contains 3 data series."

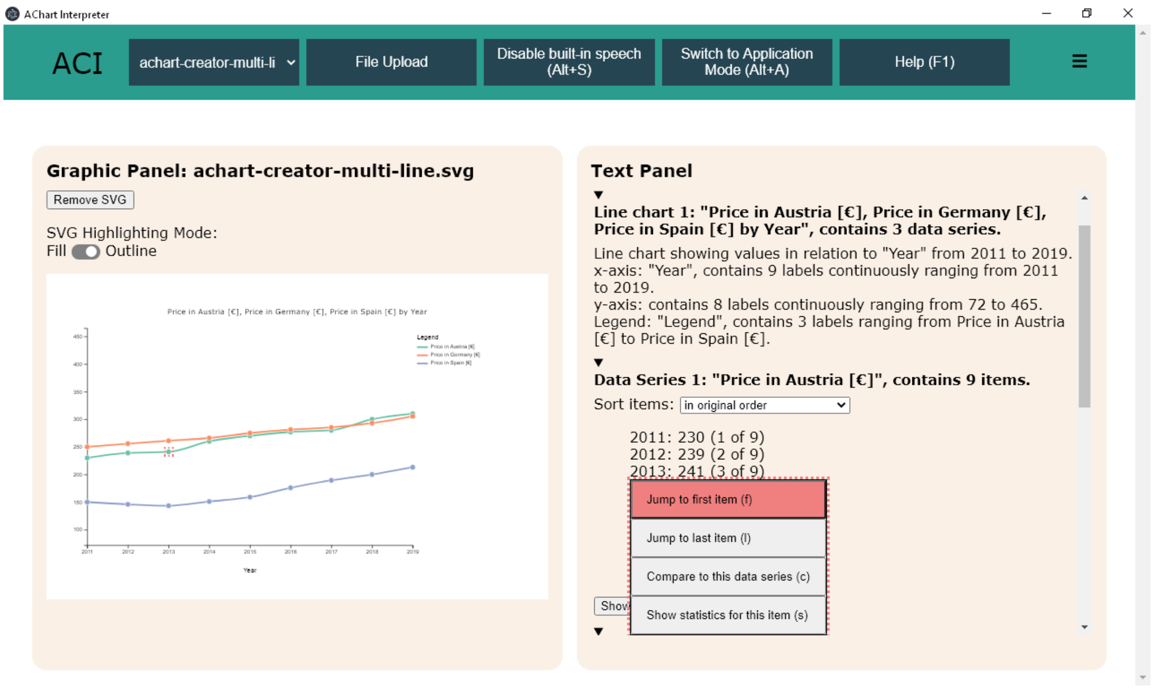
Immediately below the chart object, a short description is given. This description contains the chart type, followed by the string " chart showing ", the y-axis title in quotation marks, the string " in relation to ", the x-axis title in quotation marks, the text " from ", the first x-axis label, the string " to ", and the last x-axis label. This way, the user is able to rapidly obtain an idea of what the chart is about. In the case of a pie chart, which does not have any axes, the title of the first data series (where available) is displayed instead of the y-axis title, and the x-axis information is replaced by the title of the legend (where available), and the first and last item of the legend. Independently of the chart type, if no y-axis, data series, x-axis, or legend title has been specified, the word "values" is inserted as a replacement for the respective title. If no x-axis is available in a bar or a line chart, or if no legend is available in a pie chart for any reason, the part starting with " in relation to " is omitted. For example, a description can be: "Line chart showing values in relation to "Year" from 2011 to 2019."

Below the chart root object, the x-axis, the y-axis, and/or all legends are enumerated, where each axis or legend is represented on a separate line and the legends are listed in the order they appear in the SVG source code. Each of these objects displays a text starting with the object type. In the case of an axis, its variable is prepended to the word "axis" with a hyphen (-). The text continues with a possible axis or legend title and information of its content. Afterwards, the number of contained labels or legend items is stated along with the word "labels" or "items". In the case of a continuous axis, the word "continuously" is appended. The text ends with the string " ranging from ", the first axis label or legend item, the text " to ", and the last axis label or legend item. For example: "x-axis: "Year", contains 9 labels continuously ranging from 2011 to 2019."

Subsequently, all data series of the chart are listed. The text for each data series object begins with the expression "Data Series", followed by an index number, a possible data series title, and the number of contained data points along with the string "items". For example: "Data Series 1: "Price in Austria [€]", contains 9 items." If the data series is expanded, a drop-down menu and a list of all the data points contained in this series is shown. The drop-down menu is labelled "Sort items:". It contains the three options "in original order", "from lowest to highest value", and "from highest to lowest value". The first option is the default and lists the data points in the order they appear in the SVG source code, whereas the second and the third option cause the data points to be displayed in increasing or decreasing order of value, respectively.

The data points are enumerated as a list beneath the drop-down menu. The text of each data point starts with its name, that is, its x-coordinate or, in the case of a pie chart, the text of its associated legend item. The name is followed by a colon and a space character. Afterwards, the value of the data point is given, that is, its y-coordinate or, in the case of a pie chart, its associated label. It is followed by a space character, an opening round bracket, and an index number which denotes the position of the data point within the order of the SVG document. Only if the data points are sorted "in original order", do the index numbers always increase by one with each item in the list. The number is followed by the string " of ", the total number of data points in the series, and a closing round bracket. For example: "2013: 241 (3 of 9)"

While all other objects of the chart tree, as well as all buttons and the drop-down menu, can be focused by pressing `Tab` or `Shift+Tab`, these key commands focus the list of the data points as a whole. To navigate within the list, a particular data point can then be selected with the `Cursor-Up` and `Cursor-Down` keys. Another press of `Tab` or `Shift+Tab` sets the focus to the next or previous GUI object outside the list. This concept of navigation was chosen as it resembles the behaviour of many standard GUIs with list views. Furthermore, it provides keyboard users a way to leave the list of a data series faster than consecutively stepping through multiple data points with `Tab` or `Shift+Tab`. When focusing a list, the selection is set to


AChart Interpreter
ACI
achart-creator-multi-li
File Upload
Disable built-in speech (Alt+S)
Switch to Application Mode (Alt+A)
Help (F1)
Remove SVG
Price in Austria [€], Price in Germany [€], Price in Spain [€] by Year
Legend
Price in Austria [€]
Price in Germany [€]
Price in Spain [€]
Year
Jump to first item (f)
Jump to last item (l)
Compare to this data series (c)
Show statistics for this item (s)
Show

Graphic Panel: achart-creator-multi-line.svg

SVG Highlighting Mode:
Fill Outline

Text Panel

▼
Line chart 1: "Price in Austria [€], Price in Germany [€], Price in Spain [€] by Year", contains 3 data series.

Line chart showing values in relation to "Year" from 2011 to 2019.
x-axis: "Year", contains 9 labels continuously ranging from 2011 to 2019.
y-axis: contains 8 labels continuously ranging from 72 to 465.
Legend: "Legend", contains 3 labels ranging from Price in Austria [€] to Price in Spain [€].

▼
Data Series 1: "Price in Austria [€]", contains 9 items.

Sort items: in original order

2011: 230 (1 of 9)
2012: 239 (2 of 9)
2013: 241 (3 of 9)

Figure 7.5: Main window of AChart Interpreter's GUI after opening one of its sample SVG charts. The context menu has been activated for data point 3 of data series 1 an shows the four options described. The first option "Jump to first item (f)" is selected. [Screenshot taken by the author of this thesis.]

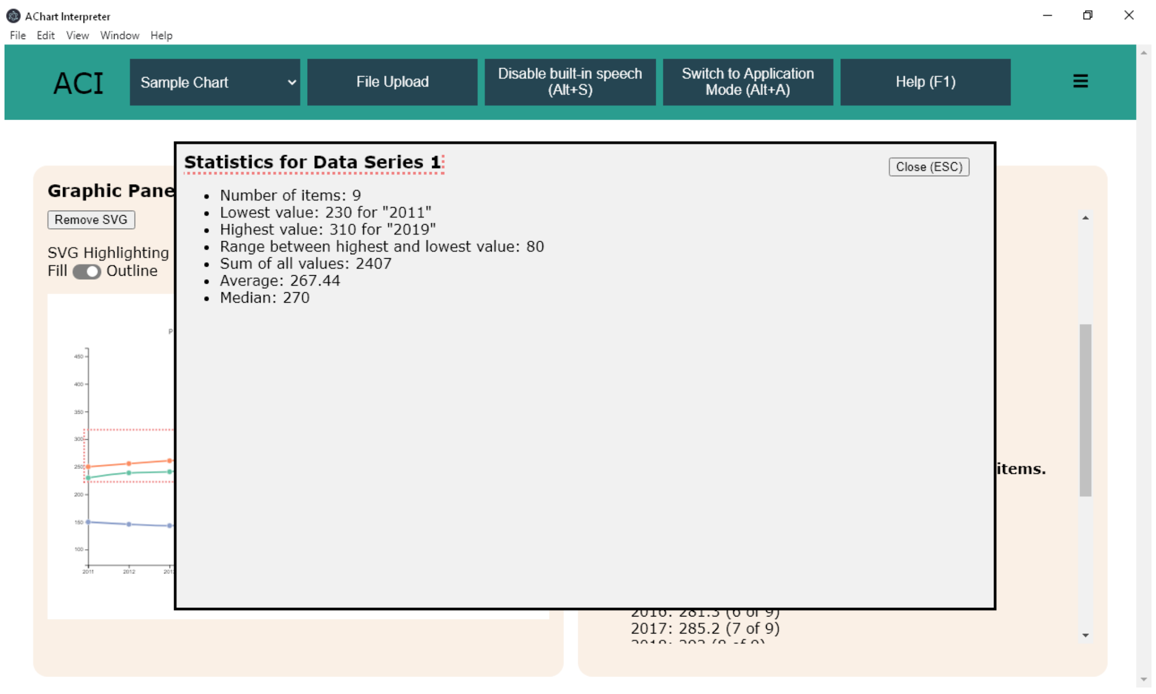
the data point which has last been focused in this data series; if this list has not yet been focused since starting the application, its first item is selected.

In addition to navigation among the data points of one series with `Cursor-Up` and `Cursor-Down`, the focus can be set to the first list item by the `Home` and to the last item by the `End` key. If the chart contains multiple data series, the user can rapidly move to the previous one pressing `Cursor-Left` and to the next one with `Cursor-Right`. If the focus is moved to a closed data series this way, the GUI object of the data series is automatically opened.

A context menu can be opened for each data point by clicking on it with the right mouse button or pressing the `Context Menu` key. A menu item is activated by a left mouse click or selecting it with the `Cursor-Up` or `Cursor-Down` key and then pressing `Enter`. In addition, each menu item provides a dedicated key command for direct activation (also known as *hot key*). This key command is displayed in round brackets at the end of the respective menu item. The menu contains the following options:

- "Jump to first item (f)": sets the focus to the first item in the current list of data points (equivalent to pressing `Home`);
- "Jump to last item (l)": sets the focus to the last item in the current list of data points (equivalent to pressing `End`);
- "Compare to this data series (c)": opens a window listing comparisons of the selected data point to all other data points in the same data series (see Subsection 7.1.2);
- "Show statistics for this item (s)": opens a window listing comparisons of the selected data point to statistical values of the same data series (see Subsection 7.1.2).

The context menu is closed without invoking any option by clicking outside its boundaries or pressing `Escape`. Figure 7.5 shows the main GUI window with the context menu opened.



Statistics for Data Series 1

- Number of items: 9
- Lowest value: 230 for "2011"
- Highest value: 310 for "2019"
- Range between highest and lowest value: 80
- Sum of all values: 2407
- Average: 267.44
- Median: 270

2016: 281.3 (6 of 9)
2017: 285.2 (7 of 9)

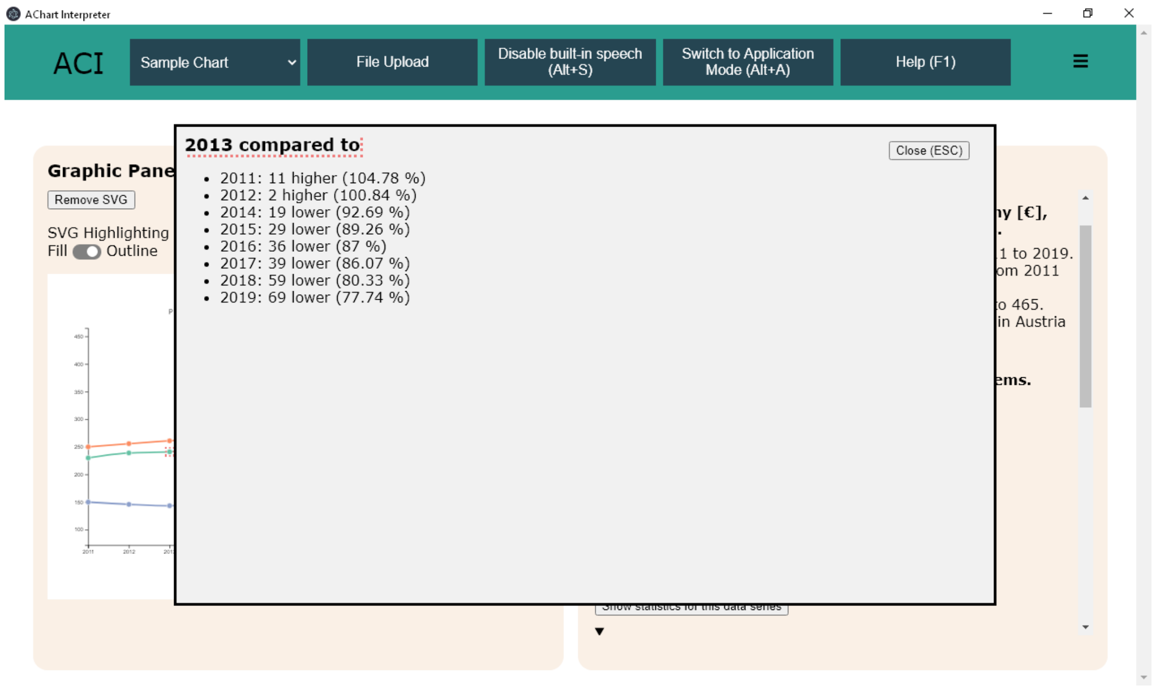
Figure 7.6: AChart Interpreter opens a modal window to display summary statistics for a data series as an unordered list. [Screenshot taken by the author of this thesis.]

7.1.2 Retrieving Additional Information

AChart Interpreter offers three analytical features for displaying further details on the underlying data of the chart. This information is displayed as unordered lists in a separate modal window. Each specific detail is represented as a list item containing the type of information, followed by a colon, a space character, and the corresponding value(s). In the current implementation, all calculated values are displayed with a precision of two decimal digits and rounded accordingly. The heading of this window as well as all list items are focusable by keyboard. After opening, the focus is set to the window heading. For more convenient keyboard navigation, both the virtual cursor of the screen reader and the keyboard focus are temporarily restricted to the window. It can be closed again by a button named Close (ESC) at the upper right corner of the window or by pressing `Escape`.

One option shows statistical values for a particular data series (see Figure 7.6). It can be invoked by activating a dedicated button named "Show statistics for this data series" underneath the corresponding list of data points. The heading of the opened window says "Statistics for Data Series ", followed by the index number of the data series. The list contains the following items:

- "Number of items": The number of data points contained by the respective data series.
- "Lowest value": The data point with the minimum value occurring in this data series; the value is concatenated with the string " for " and the name of the data point enclosed in quotation marks, for instance: "Lowest value: 230 for "2011"".
- "Highest value": The data point with the maximum value occurring in this data series; the value is concatenated with the string " for " and the name of the data point enclosed in quotation marks, for instance: "Highest value: 310 for "2019"".
- "Range between highest and lowest value": The difference between the maximum and the minimum value of this data series.
- "Sum of all values": The sum calculated from the values of all the data points in this series.



2013 compared to

- 2011: 11 higher (104.78 %)
- 2012: 2 higher (100.84 %)
- 2014: 19 lower (92.69 %)
- 2015: 29 lower (89.26 %)
- 2016: 36 lower (87 %)
- 2017: 39 lower (86.07 %)
- 2018: 59 lower (80.33 %)
- 2019: 69 lower (77.74 %)

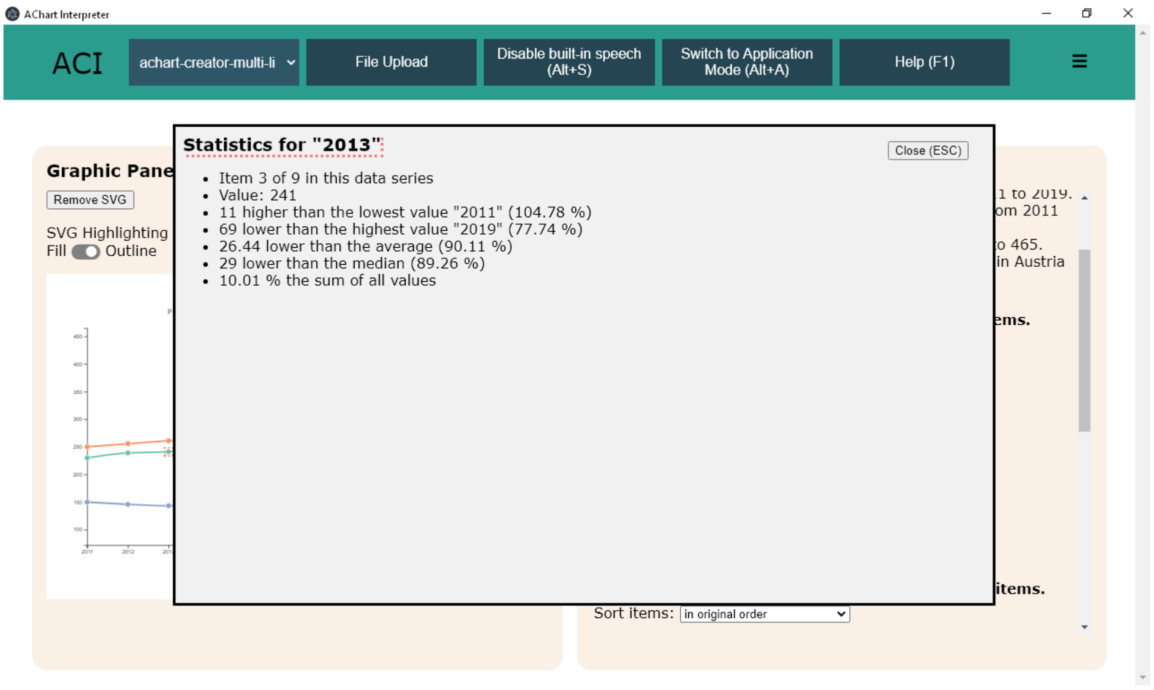
Figure 7.7: AChart Interpreter opens a modal window to show comparisons between a selected data point and other data points in the same data series. Each comparison is expressed as the absolute difference as well as the corresponding percentage and is displayed as an item of an unordered list. [Screenshot taken by the author of this thesis.]

- "Average": The arithmetic mean calculated from the values of all the data points in this series.
- "Median": The median calculated from the values of all the data points in this series.

Via the context menu, a comparison of the selected data point to all other data points in the same series can be displayed. After activating the corresponding menu item, a window opens which is entitled by the name of the selected data point, concatenated with the string "compared to"; for instance: "2013 compared to". The subsequent list items enumerate the comparisons of the selected data point to all the other ones in the data series. A list item is composed by the name of the enumerated data point, followed by a colon and a space character, the absolute difference between the value of the selected data point and that of the enumerated one, and the string "higher" or "lower", respectively. Finally, the ratio in percent between the value of the selected data point and that of the enumerated one is stated, enclosed in round brackets. For example: "2014: 19 lower (92.69 %)" In case that the values of both data points are equal, the word "equal" is displayed instead of the numerical comparisons. The comparisons window is shown in Figure 7.7.

Moreover, the user can retrieve statistical information about the selected data point, as shown in Figure 7.8. This option, too, can be chosen from the context menu. The heading of the window for this feature says "Statistics for ", concatenated with the name of the selected data point in quotation marks; for instance: "Statistics for "2013"". The first list item is composed by the string "Item ", the index number of the data point, the string " of ", the total number of data points contained by this series, and the string " in this data series". For example: "Item 3 of 9 in this data series". The next item contains the string "Value: " and the value of the selected data point.

Below, comparisons of the data point to the minimum, the maximum, the mean, and the median of the data series are displayed in this order. Each of these list items starts with the absolute difference between the value of the selected data point and that of the statistical property, concatenated with the string " higher than the " or " lower than the ", respectively. If the value of the data point is equal to that of the statistical property, the item starts with "equal to the " instead. In both cases, the type of statistical



Statistics for "2013"

- Item 3 of 9 in this data series
- Value: 241
- 11 higher than the lowest value "2011" (104.78 %)
- 69 lower than the highest value "2019" (77.74 %)
- 26.44 lower than the average (90.11 %)
- 29 lower than the median (89.26 %)
- 10.01 % the sum of all values

Figure 7.8: AChart Interpreter opens a modal window to display statistics related to a particular data point as an unordered list. [Screenshot taken by the author of this thesis.]

property and a space character are appended. The type is expressed by the string “lowest value”, “highest value”, “average”, or “median”, respectively. In the case of the minimum and the maximum, the name of the data point with the minimum or maximum value is appended, enclosed in quotation marks, and followed by a space character. All four items end with the ratio in percent between the value of the data point and the statistical property, enclosed in round brackets. For example: “11 higher than the lowest value "2011" (104.78 %)” and “26.44 lower than the average (90.11 %)” Finally, the ratio in percent between the value of the selected data point and the sum of all values is stated, followed by the string “ the sum of all values”; for instance: “10.01 % the sum of all values”

7.1.3 Modes of Screen-Reader Interaction

When the GUI of AChart Interpreter is used in combination with screen readers, they should enable their navigation facilities for web documents by default, such as the *virtual cursor* in JAWS or *browse mode* in NVDA, as explained in Subsection 2.2.3.3. While this might represent the optimal mode for reading the textual information of the chart objects, the `Cursor-Left` and `Cursor-Right` keys are captured by the screen reader in this case, which has the consequence that the user cannot benefit from the possibility to rapidly navigate between adjacent data series by means of these keys. For the same reason, the `Home` and `End` keys cannot be used for moving the focus to the start or the end of a data point list either. When activating *application mode* in JAWS or *focus mode* in NVDA, the named keys are passed through to AChart Interpreter and fulfil the described purpose. However, according to the accessibility testing performed during the implementation (see Subsection 7.4.1), this causes objects with static text, that is, the headings, the chart description, the axes, and the legends, not to be shown on a connected Braille display when using Mozilla Firefox in combination with JAWS. Moreover, some users might wish to use the virtual cursor of JAWS or the browse mode cursor of NVDA to have single characters of the text information read aloud.

To address this problem, it was decided to let the user choose between the two modes of interaction. While both JAWS and NVDA provide their own key commands to do so [Vispero 2020; NV Access 2020], AChart Interpreter offers a specific additional means of doing so. When activating the button “Switch to application mode (Alt+A)” in the Main Menu, all lists of data points available for currently loaded charts are assigned the ARIA role `application`. This means that JAWS and NVDA automatically enable their respective mode for passing keystrokes through to the application, as soon as the focus is set to one of these data point lists. When the virtual cursor or the cursor of browse mode, respectively, is moved to one of the data lists, the screen reader enables its mode for passing keys through after pressing `Enter`. Moving the focus out of the data lists causes NVDA to automatically enable browse mode again; in the case of JAWS, its virtual cursor needs to be manually activated by the user if its forms mode configuration is set to "manual". AChart Interpreter can be switched back to its default mode of interaction by pressing the same Main Menu button, now labelled "Switch to document mode (Alt+A)", again. The mode can also be toggled at any time by pressing `Alt+A`.

7.1.4 Integrated Speech Output

AChart Interpreter is able to output all its textual information using speech synthesis, independent of any screen reader. However, the application itself does not have a built-in speech synthesiser, because most speech engines are released under a proprietary licence and require a significant amount of disk storage. Instead, AChart Interpreter uses the Web Speech API, which is supported by most modern browsers [MDN 2019]. This means that the browser and/or the operating system must provide speech synthesis which can be accesses through this API. Depending on the particular configuration of browser and operating system, a speech engine may be installed locally, either as part of the operating system or of the browser, or the browser may temporarily download it from a remote server.

When launching AChart Interpreter, integrated speech output is enabled by default. It can be turned off by the menu button Disable built-in speech (Alt+S) or by the key command `Alt+S`. This causes the integrated speech to say "Speech disabled.", the text label of the button changes to "Enable built-in speech (Alt+S)", and speech output is deactivated. It can be turned on again by activating the button or pressing `Alt+S` another time. In this case, the integrated speech says "Speech enabled!", and the text label of the button changes accordingly. If no speech synthesiser is provided in the current configuration, the button is labelled No built-in speech available. In this case, neither the button nor the key command has any effect.

While speech output is active, it is triggered on every focus change and reads aloud the text content of the newly focused object. If the object is a button, it appends the word "button" for clarification. When focusing on the drop-down menu for adjusting the sorting mode, the speech first reads its label "Sort items:", followed by the mode currently selected. On each change of sorting mode, it then only speaks the newly selected mode. In the case of the checkbox for adjusting the visual highlighting of the SVG objects, the speech synthesis reads its label "Toggle SVG highlighting mode" when it is focused. If the mode is toggled, the speech says "SVG highlighting mode changed to Outline" or "SVG highlighting mode changed to Fill", respectively. The speech is interrupted by pressing an arbitrary key or clicking anywhere within AChart Interpreter's window, where any information newly available due to the user action is spoken instead. This manner of user interaction corresponds to the common behaviour of screen readers like JAWS and NVDA [Vispero 2020; NV Access 2020].

7.2 Software Architecture

AChart Interpreter was implemented according to the model-view-controller (MVC) design pattern. Data extraction and preparation is performed by dedicated classes, output composition and handling of user input by different ones, where both groups of classes do not directly communicate with each other. The main class `AChartInterpreter` acts as a bridge between model and view which interprets requests by the user, calls the appropriate methods, retrieves data, and passes them to the GUI classes.

7.2.1 Common Interfaces

In a separate module, four types are declared which are used in several classes across the model, view, and controller. In particular, these types describe special data structures which are returned by methods of certain model classes (see Subsection 7.2.2) and passed to the class `Message` (see Subsection 7.2.3). The type declarations are performed by means of TypeScript interfaces to ensure a consistent, clearly specified exchange of data structures without providing the `Message` class access to the respective model classes. The interfaces are defined as follows:

- `Scale`: A subset of the instance variables declared in the `Axis` class and the `Legend` class, where the axis-specific members `variable` and `type` are optional.
- `Statistics`: All characteristics considered for the calculation of statistical information, such as `count`, `min`, `max`, `sum`, `average`, and `median`. A data structure of this type is returned by a method of the `Dataset` class when retrieving statistics for a data series.
- `Comparison`: Characteristics for the comparison of a data point to a numerical value. A data structure of this type is returned when retrieving a comparison from the `Datapoint` class.
- `StatisticsComparisons`: All statistical characteristics for a data point. A data structure of this type is returned by a method of the `Dataset` class when retrieving data point statistics.

The same module also declares an enumeration called `Sorting` with the constant values: `NONE`, `UPWARDS`, and `DOWNWARDS`. This enumeration is used by the classes `Dataset`, `UserInterface`, and `AChartInterpreter` in order to unambiguously specify a mode for listing data points. Here, `Sorting.NONE` means no sorting, that is, the order of the data points in the SVG source code, whereas `Sorting.UPWARDS` means sorting in increasing order and `Sorting.DOWNWARDS` means sorting in decreasing order.

7.2.2 Model Classes

The model consists of six classes representing different types of logical graphical objects, namely `SVGDocument`, `Chart`, `Axis`, `Legend`, `Dataset`, and `Datapoint`. In other words, the model classes all represent nodes of the CAT. All of them have in common that their instances are initialised with a given SVG element in the current document object model (DOM), where this element is expected to be the root node of the graphics object represented by the instance. The constructor method traverses the DOM tree in order to search for one or multiple specific descendant objects. As these graphics objects are uniquely characterised by their ARIA `role` attribute in most cases, JavaScript selector methods are mainly used for this purpose.

The constructor then iterates over all elements found and processes each of them. If the type of the descendant object searched for corresponds to one of the classes named above, a new instance of the respective class is created and passed the element found. Otherwise, the data contained within the descendant object are immediately extracted. In both cases, the results are stored in an array which is accessible as a public instance variable. The techniques for searching and extracting the chart objects were inspired by those employed in Describler (see Section 3.6 and Subsection 4.5.1). The six model classes are described in the following paragraphs.

The class `SVGDocument` represents the entire loaded graphics structure and is the first class to perform the procedure of data extraction described above. When instantiated, its constructor expects to be passed the root <svg> element of the graphics document for initialisation. First, it tests whether a <title> and/or <desc> element is present as a direct descendant, in order to specify a title and/or description for the entire graphic.

Afterwards, it searches all descendants of the root for elements with the role chart. If no charts have been detected this way, the root <svg> element is itself tested for the role chart, since the Describler system also permits this element to be the chart root. For each of the charts found, a new instance of the class Chart is created and initialised with the respective chart element. All Chart instances created are stored in an array of type Chart which is accessible as a public variable of the SVGDocument instance.

A single chart is represented by an instance of the class Chart and is initialised by the passed chart element. The constructor then searches for a chart title, which is represented by a descendant element with role heading, and stores its text content in a public string instance variable. Moreover, the chart type is determined by retrieving the value of the ARIA property aria-charttype attached to the chart root element and is then stored in a public string variable as well.

Afterwards, all descendants of the chart element are searched for axes, legends, and data series. They are detected by querying for the role axis, legend, or dataset, respectively. For each axis, legend, and data series found, an instance of the respective class is created, initialised with the found element, and stored in a dedicated array as public instance member. As in Describler and the current AChart taxonomy, an axis is identified by its variable as part of its role (for example, role=xaxis in the case of an x-axis), additional queries for axes annotated with these roles are necessary: if no x-axis has been stored after the extraction procedure described above, the chart is searched for an element with role xaxis. The same strategy is applied for a y-axis.

The Chart class contains three public static methods for tasks often performed within the model. extractAll() has the purpose to find all descendant elements of a given node with a specified role. For each descendant found, a callback function is executed. The method getTitle() searches for the title of a particular chart object, applying various strategies. The algorithm is similar to that for the computation of accessible names recommended by W3C [2018d] (see Section 4.2), additionally considering descendant elements with role heading. According to the AChart ARIA system, the method first retrieves a possible aria-labelledby property of the chart object and searches for all elements referenced by this attribute. If no title has been found this way, the method tests whether a descendant with the role heading is present, which also takes into account titles annotated according to the Describler system. With regard to this role, however, it is important to take into account that it may occur several times within the graphics document, namely for the chart root, the axes, the legend, and the data series titles as well as the data point names. This means that for each of the elements annotated with the role heading, the associated chart object needs to be determined. While the AChart system prescribes that a title or name and its corresponding object be linked by the aria-labelledby property for this purpose, such an association is not defined in the original Describler system.

If, for example, all descendants of the chart root element are queried for elements with role heading, the selector method will return not only the chart title but also the axis, legend, and / or data series titles. In order to identify the chart title out of all the elements found, the role of the ancestor needs to be detected. Since the root element of the respective chart object does not need to be an immediate ancestor, the applied strategy is to recursively obtain the ancestor element until a node with a role attribute has been found. If this attribute corresponds to the object for which the title is being sought (that is, chart in the current example), the found element is considered as title for the object. The recursive test just described is performed by the method hasParent(), returning the result as a boolean value.

In the case that no suitable element with role heading could be found either, the method getTitle() tests the given object for a possible aria-label property and returns it instead. This case is not expected to occur for charts annotated by the AChart or the Describler system; nevertheless, it is considered in order to achieve a certain degree of compatibility to other chart annotation formats. For the same reason, if the property aria-label has not been specified either, a possible descendant <title> or <text> element is considered independently of any ARIA attribute. For unambiguous detection of the chart object associated with this element, the method hasParent() just described is applied here, too. An excerpt of the declarations and the constructor for instances of the Chart class is given in Listing 7.1, the described static methods can be seen in Listings 7.2 and 7.3.

Similar to the case of the Chart class, instances of classes Axis and Legend are initialised with the respective root element. Afterwards, the constructors search for an axis or legend title using the method Chart.getTitle() and for descendants representing the contained items. In the case of an axis, the items are identified by the role axislabel, for legends by the role legenditem. The text content of all axis labels is stored in a public instance variable of type string array. The same for all legend items. Furthermore, the title, minimum, and maximum values of an axis or legend are stored in separate public variables of the instance. To detect the minimum and maximum value, the presence of the ARIA properties aria-valuemin and aria-valuemax is tested. If one or both properties are not available, the first and/or the last axis label or legend item according to the order in the SVG source code are considered instead.

Since data series are denoted by the role dataset in the applied ARIA system, the class for data series is accordingly named Dataset. The constructor first tests for a possible title of the data series. Subsequently, it searches for all data points contained, querying all descendant elements with the role datapoint. Each data point element found is passed to a new instance of the class Datapoint, and all Datapoint instances are stored in two public arrays of the Dataset instance. The first array is left without modifications, whereas the second one is sorted in increasing order by the first values of the data points. Afterwards, the constructor performs the computation of the statistical values for the data series. The results are stored in public variables of the Dataset instance. Both the sorting and the statistical calculations are achieved by accessing the public numerical variables of all stored Datapoint instances.

Apart from the constructor, Dataset instances offer the public methods getStatistics() for obtaining all the statistical values as one data structure of type Statistics (see Subsection 7.2.1), getComparisonToStatistics() for obtaining the statistics of a given data point, and getComparisonToAll() for comparing a given data point to all other values of this series. The comparison methods use the public comparison method of the given data point instance.

Data points are represented by the class Datapoint. Each of its instances expects the root SVG element of the respective data point for initialisation. In order to determine the value of the data point, the constructor queries for a descendant element with the role datavalue. Its text content is extracted and stored as a public instance variable of type string. In order to support sorting and statistical computations, the string is also converted to a numerical value and stored in an associated instance variable of type number.

In addition, the Datapoint constructor tries to determine the name of the data point. In both the AChart and the Describler systems, the relation between a data point and its name is expressed by the ARIA property aria-labelledby; however, in the former case, it is assigned to the data point element, whereas, in the latter case, it is attached to the object for the data point value. For this reason, the constructor first tests if the property is set for the data point and uses this one to determine the name. Only if no name can be obtained in this way, is an aria-labelledby property possibly assigned to the data point value is considered instead. In both cases, for pie charts, the property points to the id of the associated legend item, otherwise to that of the corresponding x-axis label or, in the AChart system, to a descendant element with role heading. Since the x-axis or legend does not represent a descendant object of the data point, the entire SVG structure needs to be traversed in this case.

Beyond the constructor, this class contains the public method getComparisonTo() for calculating a comparison of the data point to a given numerical value. The method returns a data structure of type Comparison (see Subsection 7.2.1) containing the label of the data point, the absolute difference between the value of the data point and the given value, as well as the corresponding percentage.

```
export class Chart
{
  // Root element of the chart:
  root : SVGElement

  // Chart type (e. g., bar, line, pie):
  type : string

  // Chart title:
  title : string
  ...
  // Data series:
  datasets : Dataset[] = []

  axes =
  {
    x: <Axis>undefined,
    y: <Axis>undefined,
    others: new Array<Axis>(0)
  }
  ...

  constructor(root : SVGElement)
  {
    this.root = root;

    this.type = root.getAttribute("aria-charttype");
    ...
    this.title = Chart.getTitle(this.root, "chart", this.root);
    ...

    Chart.extractAll(root, "dataset", (item : SVGElement) =>
    {
      this.datasets.push( new Dataset(item, this.root) );
    });

    ...

  }

  ...
}
```

Listing 7.1: Excerpt from the source code for the model class Chart. First, the public instance variables are declared. The constructor then retrieves the chart type by reading the ARIA property aria-charttype, the title by means of the static method getTitle(), and uses the static method extractAll() to find and store all contained data series.

```
static extractAll(root : SVGElement, type : string,
    callback : Function, required_parent_role? : string) : void
{
  let elements = root.querySelectorAll("[role='" + type + "']");
  for (let index = 0; index < elements.length; index++)
  {
    if ( (!required_parent_role) ||
        (Chart.hasParent(elements[index], required_parent_role)) )
    {
      callback(elements[index]);
    }
  }

}

...

static hasParent(element : Element, role : string) : boolean
{
  let parent_role = "", parent_element = element.parentElement;

  // Find the next parent of the given element with any ARIA role:
  while ( (parent_element) &&
      !(parent_role = parent_element.getAttribute("role")) )
  {
    parent_element = parent_element.parentElement;
  }

  // Check if this parent has the required role:
  return (parent_role === role);
}
```

Listing 7.2: Two public static methods of the model class Chart. extractAll() searches all descendants of the given SVG element for objects with the specified ARIA role and executes the provided callback function with each of the found elements as argument. Optionally, only those of the found objects are considered which have an ancestor with the specified role. The method uses hasParent(), which searches the next ancestor with an ARIA role assigned and then tests whether this role is the desired one.

7.2.3 View Modules

The basic initial GUI of AChart Interpreter is defined in an HTML file. It contains the logo text of the application, the main menu including the two controls for loading an SVG file, the button to hide and unveil the menu in the case of displaying the GUI in a narrow window, and a container for the placeholder text along with the arrow-up symbol. Moreover, the basic structure of the graphic and the text panel is defined, and the main JavaScript file and Cascading Style Sheets (CSS) definitions are referenced.

The interactive view functionality mainly consists of the class UserInterface. It is instantiated by the constructor of the AChartInterpreter class instance immediately when launching the programme. UserInterface is in charge of creating GUI objects, writing messages and chart data to them, appending them to the appropriate node in the DOM tree, attaching event listeners to them, handling user input, and managing keyboard focus. The GUI objects are created by inserting HTML templates defined as string constants in the static class HTMLTemplate; some of them also contain additional inline CSS. For each GUI object which might need to be modified at a later time, its reference is stored in an instance variable of the class

```
static getTitle(element : SVGElement, role : string,
    root : SVGElement) : string
{

  // First, consider a possible aria-labelledby property

  let label_ids_str = element.getAttribute("aria-labelledby") || "";
  let label_ids = label_ids_str.match(/\S+/g) || [];

  let label = "";
  for (let index = 0; index < label_ids.length; index++)
  {
    let label_element = root.querySelector("#" + label_ids[index]);
    if ( (label_element) && (label_element !== <Element>element) )
    {
      if (label)
      {
        label += ", ";
      }
      label += label_element.textContent.trim();
    }
  }

  if (label) { return label; }

  // If no title has been found this way,
  // search for a child element with ARIA role "heading"
  let title_element = element.querySelector("[role='heading']");
  if ( (title_element) && (Chart.hasParent(title_element, role)) )
  {
    return title_element.textContent.trim();
  }

  // If still no title has been found, consider the property "aria-label":
  if (label = element.getAttribute("aria-label"))
  {
    return label.trim();
  }

  // If still no title has been found, search for a child <title> element
  // without ARIA role:
  title_element = element.querySelector("title");
  if ( (title_element) && (!title_element.getAttribute("role"))
      && (Chart.hasParent(title_element, role)) )
  {
    return title_element.textContent.trim();
  }

  // As a last attempt, look fora child <text> element without ARIA role:
  title_element = element.querySelector("text");
  if ( (title_element) && (!title_element.getAttribute("role"))
      && (Chart.hasParent(title_element, role)) )
  {
    return title_element.textContent.trim();
  }

  // It seems there is no suitable text at all, so return an empty string:
  return "";
}
```

Listing 7.3: Public static method getTitle() of the model class Chart. It applies the described algorithm to determine the title of the given element and uses hasParent() to test if the role of the found candidate is equal to that of the element.

UserInterface in order to minimise the number of DOM queries. Furthermore, every chart tree object is attached a listener for focus events in order to facilitate synchronised focus highlighting, and every SVG element representing a chart object within the graphic panel is added a listener for mouse events which sets the focus to its counterpart in the text panel.

The constructor of the UserInterface instance composes the initial GUI and enables the general functionality. It adds text labels to the existing controls and creates the buttons for activating and deactivating the integrated speech output, for toggling between document and application mode, as well as for showing the help window. Moreover, it adds handlers for click events to these controls and a global event listener for processing key presses and mouse events to the main window. Afterwards, it initialises the components of the graphic panel and attaches key event listeners so that Tab navigation skips the displayed SVG document. Apart from the GUI creation, the constructor also instantiates the Speech class for the initialisation of the speech synthesis and attaches an event listener to the main window which causes the text of every newly focused object to be read aloud if the user has enabled the speech output. Finally, the context menu for data points is initialised.

When a file has been opened, its SVG content is passed to the method insertSvg() of the UserInterface instance. This method hides the placeholder text, appends the SVG to a container element of the graphic panel, and unveils the controls for removing the graphic as well as for toggling the highlighting mode. After the completion of the analysis, the method initTextPanel() is called, which takes information on the root <svg> element of the loaded graphic as arguments and creates a corresponding root node of the chart tree. Subsequently, all other chart tree objects are inserted into the text panel by dedicated methods of the UserInterface instance. They are called and passed the corresponding chart data by an instance of the main class AChartInterpreter. Those chart tree objects which can be expanded and collapsed in order to display and to hide descendant chart objects are implemented using the HTML element pairing of <summary> and <details>.

The method initDataList() has a special role for initialising a list of data points. It is invoked one time at the creation of each data series and appends the drop-down list for choosing the sorting mode, a container element for the list view, and the button for displaying the statistics to the data series object. Afterwards, it adds an event listener to the drop-down list which triggers the sorting according to the user's choice. The list view container, too, has an event listener attached, which handles key presses for the navigation among the data points and series. The event listeners for opening the context menu are attached to each single data point upon creation. Other methods of the UserInterface instance include those for the visual highlighting of SVG objects, for displaying details in a separate window, as well as for the removal of the current list view and its replacement by a new one according to the given sorting mode.

The context menu displayed for each data point on request is represented by instances of the ContextMenu class. An instance stores each menu item as a special type of data structure according to an interface defined in the same module. The set of menu items is implemented as a linked list of these variables. First, the constructor creates the menu container and then assigns it the necessary listeners for mouse and key events. The method addItem() creates a menu item and adds it to the menu. This method needs to be called for each new menu item, passing it the text of the item, its associated key command, and the function which shall be called on activation. The order in which the different menu items are registered determines the order they are listed in the menu. To open and close the menu, other dedicated methods of the ContextMenu instance need to be called. The open() method expects to be passed the DOM node the menu shall be appended to. More details on the implementation of this class are given in Subsection 7.4.2.

Message is a static class with various methods for composing human-readable text messages, such as chart, axis, and legend summaries and descriptions. Each method expects the necessary data to be shown in the respective message and concatenates them with text blocks defined in the Text class. The latter is a static class of string constants for all kinds of text output intended. This organisation of message composition shall provide the basis for localisations to other languages with low technical effort. The methods of the Message class are mainly called by an instance of the main class AChartInterpreter, where their return value is immediately passed to a method of the UserInterface instance. As an example, the

```
static getChartDescription(type : string, values_title : string,
    names_scale : Scale) : string
{
  // Start the description with the chart type:
  let description = (type in Text.CHART_TYPE) ? Text.CHART_TYPE[type][2] : Text.
      CHART_TYPE.other[2];

  // Append "showing" and the y-axis or data series title;
  // if none is given, insert "values" as placeholder text instead:
  description += ` ${Text.SHOWING} ` + ( (values_title) ?
      `"${values_title}"\n` : `${Text.SCALE_TITLE_REPLACEMENT} ` );

  // Only append this part if any x-axis or legend data are given:
  if (names_scale)
  {

    // Append "in relation to" and the x-axis or legend title ;
    // if none is given, insert "values" as placeholder text instead:
    description += `${Text.RELATED_TO} ` + ( (names_scale.title) ?
        `"${names_scale.title}"\n` : `${Text.SCALE_TITLE_REPLACEMENT} ` );
    // End with the range of x-axis or legend values:
    description += `${Text.FROM} ${names_scale.min} ` +
        `${Text.TO} ${names_scale.max}`;
  }

  return description + ".";
}
```

Listing 7.4: Source code of the static method getChartDescription() in the Message class. It expects the chart type, the title of the y-axis, as well as the title, the first, and the last label of the x-axis as input. In the case of a pie chart, the data series title can be passed instead of a y-axis title and the legend data instead of x-axis data. The method creates a short description of the chart. If no x- or y-axis title is given, it inserts the placeholder text "values". In case no x-axis or legend data are present, it omits the part starting with " in relation to ".

method Message.getChartDescription() is shown in Listing 7.4.

The class Speech is used to facilitate the integrated speech feature of AChart Interpreter. It encapsulates the various classes of the Web Speech API and performs all necessary steps for their initialisation. When creating a Speech instance, its constructor tests if any speech synthesis is available on the current configuration and initialises the text label of the GUI button for toggling speech output accordingly. In case no speech is supported, a corresponding static message is written to the button. Otherwise, the button is labelled Disable built-in speech (Alt+S), and initialisation of speech synthesis is started. Afterwards, the public toggle() method is called to disable or enable the synthesis and to change the text label of the menu button accordingly. The public method speak() expects an arbitrary text string and causes it to be read aloud if speech has been enabled. Moreover, the public method readElement() composes a message out of a given GUI object, depending on the object type, and invokes speak() on this message. Further details on the implementation of the Speech class will be given in Subsection 7.4.3.

7.2.4 Controller Modules

The module `achart-interpreter` contains the main class AChartInterpreter and represents the entry point of the software. In order to facilitate choosing and opening an SVG file, it communicates with the static class FileLoader. The `achart-interpreter` module first adds an event listener waiting for the HTML content to completely be loaded. Its callback function creates an instance of the class AChartInterpreter, which launches the application.

The constructor of the AChartInterpreter class first creates a UserInterface instance and, this way, triggers the composition of the initial GUI. Afterwards, it invokes the method FileLoader.prepareFileChoosing(), which attaches event listeners for user input to both controls for loading a file and inserts the names of all sample SVG charts into the corresponding drop-down list. The set of sample charts is determined by the entries of the plain-text file `samples.txt` which is generated at build time and placed into the directory of AChart Interpreter's main JavaScript file.

When the user has chosen a file from one of the two controls in the main menu, its SVG content is read by a private method of the FileLoader class and passed to the method interpret() of the AChartInterpreter instance. interpret() immediately calls the method insertSvg() of the UserInterface instance, which appends the SVG content to a container of the graphic panel and returns the DOM node of the root <svg> element. Afterwards, an instance of the SVGDocument model class is created and initialised with this root element, which starts the analysis of the entire SVG structure and the composition of the CAT as described in Subsection 7.2.2. As soon as this process has completed, the method initTextPanel() of the UserInterface instance is called and passed a summary of the loaded graphic, the number of charts contained by the SVG, and a possible description provided by the author. The summary is obtained by calling the getSvgSummary() method of the Message class.

Subsequently, the interpret() method iterates over all charts and chart objects found within the SVG structure. For each graphics object, in a similar manner as just described for the root <svg> element, it invokes the appropriate Message methods in order to generate human-readable text strings and a UserInterface method to create a corresponding GUI object in the chart tree representation. As an example, the source code for inserting the GUI objects of the graphics root and all contained charts is shown in Listing 7.5.

For each data series, the method initDataList() of the UserInterface instance (see Subsection 7.2.3) is called. The iteration over all the data points of a series for inserting them into the chart tree was placed into a separate AChartInterpreter method named listDataPoints(), because it needs to be performed again every time the sorting mode is changed. Other methods of an AChartInterpreter instance include showDatasetStatistics() for displaying the statistical characteristics of a given data series, compareToRestOfDataset() for displaying the comparisons of a particular data point to all other data points in the same series, and compareToStatistics() to display the statistical characteristics of a specified data point. All three methods are invoked by event listeners within a UserInterface instance. They retrieve the requested data or comparisons from the specified Dataset instance, pass them to an appropriate method of the Message class, and hand the resulting text string to the showDetails() method of the UserInterface instance.

```
interpret(svg_content : string, filename : string) : void
{
  // Display the SVG in the graphic panel:
  let svg_root = this.user_interface.insertSvg(svg_content, filename);

  // Parse the SVG, extract chart data etc.:
  this.svg_document = new SVGDocument(svg_root);

  // Initialise text panel with SVG title and summary:
  this.user_interface.initTextPanel(Message.getSvgSummary(
      this.svg_document.titles[0], this.svg_document.chart_type_counts),
      this.svg_document.charts_count, this.svg_document.descriptions[0]);


  // Iterate over all known chart types incl. "unknown"

  // Index of chart independent of its type (for user interface):
  let charts_index = 0;

  for (let chart_type in this.svg_document.charts)
  {


    // Add all charts of this type to the web interface

    for (let charts_index_of_type = 0;
        charts_index_of_type < this.svg_document.charts[chart_type].length;
        charts_index_of_type++, charts_index++)
    {
      let chart = this.svg_document.charts[chart_type][charts_index_of_type];
      this.svg_document.all_charts[charts_index] = chart;

      // If x-axis is present, consider it as names scale,
      // otherwise consider a possible legend:
      let names_scale = chart.axes.x ||
          ( (chart.legends) ? chart.legends[0] : undefined );
      let names_scale_data : Scale = (names_scale) ?
      {
        min: names_scale.min,
        max: names_scale.max,
        title: names_scale.title
      } : undefined;

      // If y-axis is present, consider it as values scale,
      // otherwise consider the first data series:
      let values_scale_title = (chart.axes.y) ? chart.axes.y.title :
          ( (chart.datasets) ? chart.datasets[0].title : "");

      this.user_interface.addChart(charts_index,
          chart.root, true, chart.datasets.length,
          Message.getChartSummary(chart.type, chart.title,
          chart.datasets.length, true, charts_index_of_type),
          Message.getChartDescription(chart.type, values_scale_title,
          names_scale_data));


      // Add all axes of each chart to the web interface
      ...
```

Listing 7.5: Excerpt from the source code of the method interpret() defined in the AChartInterpreter class. It first displays the visualisation in the graphic panel, starts the analysis of the SVG content, creates the GUI object for the graphics root, and then inserts a GUI object for each of the charts found within the graphics document, enumerating them by their types.

7.3 AChart Summariser

AChart Summariser is a command-line programme which outputs a textual summary of an accessible ARIA-annotated SVG chart. The text output is encoded as 8-bit Unicode Transformation Format (UTF-8) and is structured using markdown syntax [GitHub 2020]. The programme displays basic summary information about each chart in the SVG file, such as the chart title, its type, the titles of the data series and the number of contained data points, as well as information about the axes and legends. Command-line options can be used to output statistics on all data series and to list their data points.

7.3.1 User Interaction

AChart Summariser is started by calling its executable binary file from the text terminal. If the executable file is located in the current working directory or in a directory included within the search path of the platform, usually the command `asummarise` or `./asummarise` will launch the programme. The command-line syntax is as follows:

```
asummarise [--output OUTPUT-FILENAME] [--statistics]
           [--datapoints] [--version]
           [--help] [--input] SVG-FILENAME
```

All options are treated as case-insensitive and can be used in arbitrary order. The argument `SVG-FILENAME` is mandatory and can be given either as the last command-line parameter or, alternatively, at any position, prepended by `--input`. In case an argument other than those above is given, the programme will exit with an error message stating that an invalid option has been used. Similarly, if no input SVG file is specified or if one of the above options requiring an associated argument is used without providing the argument, the programme will exit with an appropriate error message. In all these cases, the standard help text (see Appendix C) will be displayed along with the respective error message.

Once running, the version information "AChart Summariser version 1.0.0" and the message "File... loaded" are written to stdout. The chart summary is written to stdout, unless an output file was specified. It consists of the following data:

- The name of the loaded SVG file.
- The title and description of the SVG and the number of contained charts.
- Each chart found in the SVG, expressed by its chart type, an index number, its title, and the number of its contained data series.
- For each chart:
  - A description including the chart type, the titles of the contained axes, and the first and last value of the x-axis. In the case of pie charts, which do not have any axis, the title of the first data series found, the title of the first legend found, as well as the first and last item of this legend will be displayed instead.
  - Each axis found, expressed by its variable, its title, the number of its labels, and its first and last label.
  - Each legend found, expressed by its title, the number of its labels, and its first and last label.
  - Each data series found, expressed by an index number, its title, and the number of contained data points.

If the option `--statistics` is given, AChart Summariser will additionally display statistical information for each data series, including the number of contained data points, the minimum and maximum value, the range between the latter two values, the sum of all values, the average, and the median. Using the option `--datapoints` will additionally output all data points of every data series. All graphics and chart information is composed in exactly the same manner as the textual content of the chart tree objects specified in Subsection 7.1.1, with the exception that each title is followed by a quotation mark, a comma and a line break instead of a quotation mark, a comma, and a space character. The line breaks are inserted to avoid writing beyond the end of a screen line in case of long titles since most text terminals do not provide automatic line breaking between words. The text fragments are structured by means of markdown elements for headings and unordered lists as described in Subsection 7.3.2.

If the option `--output` is specified along with a valid filename, the above output, starting with the name of the SVG file, is directed to the specified file as plain text. In this case, the user will be informed by a console message that the summary is being written to this file. Both the input and the output filename may include a path. If no path or a relative path is specified, the programme will take the current working directory as a reference. The option `--help` causes AChart Summariser to display its standard help text, as listed in Appendix C, and then exit. If the option `--version` is given, AChart Summariser prints version information and exits.

7.3.2 Software Architecture

AChart Summariser can be regarded as a command-line version of AChart Interpreter. Indeed, both programmes share nine common modules: in particular, all six model classes for chart analysis, the view classes `Message` and `Text` for message composition, and the module declaring the common interfaces. Node.js is used as the JavaScript runtime environment. Since no browser DOM is available in this case, the DOM is emulated using the `jsdom` package [jsdom 2021], in order to facilitate the DOM traversal of SVG elements performed by the model classes. Self-contained binary executable files for various platforms are generated using `nexe` [Boyd et al. 2020].

The handling of command-line arguments and text output is achieved by the main class `AChartSummariser`. For reading SVG and writing to text files, the programme contains the singleton class `FileHelper`. The general procedure of AChart Summariser can be described as follows. After launching the `asummarise` command, an instance of the main class is created, containing instance variables which correspond to the possible execution parameters. The constructor calls the method `parseCommandLine()`, which sets these variables according to the arguments specified by the user.

Afterwards, the `AChartSummariser` instance is passed to the static method `FileHelper.loadFile()`. This method opens the specified input file, searches it for a starting `<svg>` tag, reads all the file content beginning at the position of this string, and appends the SVG structure as DOM node to the `<body>` element of the virtual document. As the function for reading the file is an asynchronous process, the root `<svg>` element cannot be returned to the constructor method. Instead, it is passed to the method `processSVG()` of the main class instance by means of an anonymous callback function, which is executed as soon as loading has been completed.

The method `processSVG()` first calls the static method `FileHelper.openOutput()` which sets the instance variable `output` of the `AChartSummariser` instance according to the user's preference. If an output file name has been specified from the command line, a file with the desired name is opened with write access, and its associated write stream is stored in `output`. Otherwise, the instance variable is set to stdout. This way, all subsequent information can be written to the preferred output target without the need for multiple conditional statements. Afterwards, an instance of the class `SVGDocument` is created from the passed SVG node, which starts the analysis process.

Finally, the method `processSVG()` iterates over all chart objects which have been detected during the analysis and whose information has been requested by the user. For each of these objects, the extracted information is composed into human-readable text strings using the methods of the singleton `Message` class and written to the stream stored in the `output` instance variable. In order to create a textual layout and markdown syntax reflecting the nested structure of the chart objects, line breaks and indentations are inserted according to the level of the respective object within the hierarchy. All graphical objects are represented as list items by a leading hyphen and a space character (-). The main heading is marked by prepending the number character and a space "# ", subheadings are denoted by prepending two number characters and a space "## ".

7.4 Selected Details of the Implementation

This section covers aspects of AChart Interpreter's GUI implementation which were particularly interesting or tricky.

7.4.1 Accessibility Considerations

As mentioned in the introductory paragraphs of this chapter, a major goal of the project was to make AChart Interpreter as screen-reader-friendly as possible. Throughout the entire implementation phase, the accessibility of the GUI was extensively tested on the following configurations:

- *Operating system*: Windows 10.
- *Browsers*: Mozilla Firefox, Google Chrome, and Electron.
- *Screen readers*: JAWS 2019, JAWS 2020, and NVDA 2020.

A fundamental question was how to present the chart objects and their data to screen reader users. In most of the software solutions currently available, a blind user navigates directly through the SVG elements, where information on the focused element is conveyed to the screen reader. For instance, Describler writes the data extracted from the focused object to a read-only text field. If this text field was assigned the ARIA property `aria-live`, screen readers might announce any text written to it by speech, but, testing for this project found that this information was not shown on a connected Braille display. Furthermore, this strategy does not provide any means for screen reader users to have the current information spelt out or to repeat it without leaving the current element and returning to it.

The solution of embedding data into the SVG elements by means of the `aria-label` or `aria-labelledby` property, as applied in several charting libraries, overcomes the two described issues. Nevertheless, this manner of implementation implies the modification of the SVG source code at runtime. The sorting of data points, especially, would require substantial changes to the chart itself in this case. In general, presenting the data in a standard HTML user interface produced more stable results with lower latencies during keyboard navigation. For this reason, the hybrid representation by means of a graphic and a text panel was chosen for AChart Interpreter. This solution has the additional advantage that, for a sighted user, several pieces of the extracted data are visible at the same time.

The objects of AChart Interpreter's GUI were implemented by semantically appropriate HTML elements wherever possible. In particular, the following elements were used:

- `<h1>` to `<h3>` for headings.
- `<p>` for paragraphs of static text.
- `<button>` for all buttons which toggle between different modes, or open or close a window.
- `<select>` with descendant `<option>` elements for drop-down lists.
- `<label>` for the associated text of a drop-down list with the `<select>` element as descendant.
- `<details>` for chart tree objects containing descendant GUI objects, i.e. the graphics root object, charts, and data series. The `<details>` element supports opening and closing by both mouse click and keyboard and reports its current status to the accessibility tree of the browser.
- `<summary>` for the text label of a `<details>` element.
- `<ol>` and `<ul>` for lists.
- `<li>` for data points and all other list items.

The semantically neutral `<div>` element was only used for containers and for short fragments of static text, in particular, the objects for descriptions by the graphic's author, chart descriptions, axes, and legends.

With this implementation, no ARIA roles needed to be set for most of the GUI elements. For all those which have their associated text included, ARIA labels were not regarded as necessary either. However, every container for a list of data points was given the property `aria-label` with its value set to the title of the respective data series, causing this title to be read aloud by screen readers when moving the focus into this list. For data series without any title, the default label was set to “Data list”.

Both the structure of the basic HTML source code and the insertion of all the dynamically created GUI objects were implemented so that the resulting order within the DOM tree corresponds to a logical reading order for blind users. This way, screen reader users can start the navigation from the main menu at the top of the window and then move downwards level-by-level through the chart tree until reaching a list of data points. Furthermore, all GUI objects were assigned the `tabindex` attribute so they can receive keyboard focus. While this would not have been necessary for static text fragments due to the special navigation features of screen readers for web documents, these objects were included into the `Tab` order as well, so that they can be read by the integrated speech output.

For most of the elements, the `tabindex` attribute was set to `0`, enabling navigation by `Tab` and `Shift+Tab`. In the case of the lists of data points, by contrast, the value `0` was only assigned to the list container, whereas for the single data point objects the `tabindex` attribute was set to `-1`. This way, the data points can programmatically receive focus without being included in the `Tab` order. An event listener was attached to the container of a data list, which reacts as soon as the container is focused by `Tab` or `Shift+Tab`. Its callback function immediately moves the focus to the descendant data point object which has been stored as previous selection of the respective list. This, however, had the consequence that the propagation of focus events needed to be stopped for all data point objects because, otherwise, focusing a data list would result in an infinite loop. Once the focus has been set to any data point object, the `Cursor`, the `Home`, and the `End` keys are evaluated in a `switch` block for calculating which data point to focus next. The corresponding GUI object is then selected by the `focus()` method specified for markup elements.

Certain interactive elements, such as `<button>` and `<details>`, dispatch click events also on presses of the `Enter` or the `Space` key, others do not, such as drop-down menus. For this reason, an additional event listener for handling presses of the `Enter` key was attached to the element. A solution for the desired functionality could be to use a listener for change events, which is always triggered when selecting a list item by mouse or keyboard. However, since stepping through the list items by keyboard is a common way for blind users to read the items within a drop-down list, this manner of implementation would have caused the associated action to be triggered at each of these navigation steps.

AChart Interpreter was implemented in an iterative design process which involved several informal usability evaluations of the GUI performed by a blind user. In addition, the user interface was informally tested one time by three other blind persons. In general, the GUI received highly positive feedback from all participants. Many of the suggestions for improvement given by the users could be implemented in subsequent iteration steps.

7.4.2 The Context Menu

Context menus are a common way to access specific options related to the selected GUI object. Context menus are particularly popular among screen reader users, since they can usually be opened by a single key press without further searching, support keyboard navigation, and additionally offer hot keys to directly activate a particular option. Another advantage of context menus for visually impaired users is the fact that, after quitting a menu without any action, usually by pressing `Escape`, the focus is reset to the GUI object from which the menu was opened. For this reason, it was decided to provide some of AChart Interpreter’s optional functions in a context menu which can be launched for each data point. The menu is opened by clicking on any data point with the right mouse key or selecting a data point by keyboard and then pressing the `Context Menu` key. While it currently contains only four entries, the `ContextMenu` class was implemented so that any item can be added by calling the `addItem()` method of the respective instance. This method expects three arguments: the string to be displayed for the entry, the associated hot key, and the function to be executed on activation.

HTML 5.1 specifies the `<menu>` element along with `<menuitem>` for exactly the purpose just described. However, these elements are marked as experimental and are currently not supported by most browsers [MDN 2020], meaning that the implementation of the desired functionality belongs to the web author's responsibilities for now. The design goals for the data point context menu were the following:

- It should appear as a pop-up menu, visually located near the chart tree object of the selected data point.
- The activation of any option should be possible by left mouse click.
- Alternatively, the activation of any option should be possible with a hot key.
- Selection of a menu item should be possible using the `Cursor-Up` and `Cursor-Down` keys with cyclic navigation, i.e. moving upwards from the first or downwards from the last item set the selection to the last or first item, respectively.
- The selected item should be visually highlighted.
- When opening the menu, screen readers should be aware of it; both screen readers and the integrated speech, if enabled, should report to the user that the focus has moved to a menu and output the menu item initially selected.
- When selecting a different menu item, it should be reported by screen readers and the integrated speech.
- The option of the selected menu item should be activated by pressing the `Enter` key.
- The menu should always close after activating an option or, without any other menu action, when clicking outside the boundaries of the menu or pressing `Escape`.
- When closing the menu by pressing `Escape`, the focus should return to the data point selected when the menu was opened.

To achieve this functionality, the context menu was implemented as follows. The `<div>` element was chosen as container for the menu, combined with CSS specifying the visual appearance. In order to provide appropriate semantic information for screen readers, the element was assigned the ARIA role `menu`. Furthermore, it was given the `tabindex` attribute set to `-1`, so that it can be focused programmatically.

Since the class `ContextMenu` is meant to be instantiated only once at GUI creation, the constructor was designed to perform as much of the initialisation as possible. The `<div>` container defined in the `HTMLTemplate` class is made a DOM node and an event listener is attached for all key presses. In order to avoid any possible conflicts with key commands of the web application or the browser, the first step within the callback function is invoking the `stopPropagation()` and `preventDefault()` methods of the key event. Afterwards, the key event is tested for any modifier key pressed, returning without any action if the result is `true`. Finally, the pressed key is evaluated in a `switch` block which considers `Cursor-Up` for selecting the previous menu item, `Cursor-Down` for selecting the next one, `Enter` for activating the option associated with the selected item, and `Escape` for closing the menu. Moreover, in the `default` case, it is tested if the pressed key represents the hot key defined for any of the menu items and, if so, the associated option is executed. In a second event handler, mouse clicks on a menu item are processed, so that the option of the clicked item is executed.

The basic element of a single menu item was defined in `HTMLTemplate` as `<button>` with the ARIA role `menuitem`. All items currently supported by the application were declared in an array of the `Text` class, as shown in Listing 7.6. Each array item represents one menu item in the form of a structure which holds the displayed text as well as its hot key. In the instances of the class `ContextMenu`, a menu item is represented as a structure of a special type `Item` defined by means of a TypeScript interface in the same module (see Listing 7.7). This type contains four variables, namely the reference of the GUI object for the item,

```
static readonly MENU_ITEM =
{
  JUMP_TO_BEGINNING:
  {
    text: ‘Jump to first item‘,
    hot key: ‘f‘
  },
  JUMP_TO_END:
  {
    text: ‘Jump to last item‘,
    hotkey: ‘l‘
  },
  COMPARE_TO_SAME:
  {
    text: ‘Compare to this data series‘,
    hotkey: ‘c‘
  },
  COMPARE_TO_OTHER:
  {
    text: ‘Compare to other data series‘,
    hotkey: ‘d‘
  },
  SHOW_DATAPOINT_STATISTICS:
  {
    text: ‘Show statistics for this item‘,
    hotkey: ‘s‘
  }
}
```

Listing 7.6: Excerpt from the source code of the static class Text. The four items of the data point context menu are defined as an array, each of whose entries contains both the text label for the menu item and the associated hot key. In addition, a fifth text and hot key for comparing a data point to another data series is defined for future use.

the reference to the function which shall be executed on activation, as well as two variables referencing the previous and the next item within the order of the menu. This way, the items can form a circular doubly linked list, which was considered as preferred data structure because its sequential manner of access facilitates the cyclic sequential navigation by means of the Cursor keys without the need of any conditional execution on each step.

Each item is inserted into the menu by appending its <button> element to the <div> container and initialising a new Item structure. If the new item is the last one within the sequence, its reference to the next item is set to the first one, whereas the reference of the first item to the previous one is assigned the new item, facilitating cyclic navigation. Moreover, the item structure is stored in a map, where its index represents the associated hot key. This way, the press of a hot key can be processed by solely accessing the corresponding map entry. In order to enable the insertion of items after instantiation has taken place, the steps just described were implemented in a separate public method, which expects to be passed the displayed text, along with the corresponding hot key of the item as the two-string structure defined in the Text class, and the function to be executed on activation. An excerpt of the method is shown in Listing 7.8.

When opening the menu, its container element is inserted into the DOM tree as immediate descendant of the GUI object passed by the caller, which is expected to be the selected data point object. Afterwards, the first item is selected, and the focus is set to the menu container. Navigating forward and backward within the menu is possible by just removing the selection from the current item and selecting the one

```
interface Item
{
  run : Function
  element : HTMLElement
  previous_item : Item
  next_item : Item
}
```

Listing 7.7: Excerpt from the source code of the class ContextMenu. The type for storing a menu item is declared as a structure with the member variables for the GUI object reference, the function to be executed when activating the menu item, the previous item, and the next item within the order of the menu.

referenced by the `next_item` or `previous_item` pointer, respectively. The source code for selecting the next item and for processing hot keys can be seen in Listing 7.9. The selection of a particular item is performed by setting the instance variable `selected_item` to the target Item structure and adding a class for visual highlighting to the associated HTML element. Furthermore, the id attribute of the element is set to a predefined string, which is used in conjunction with the ARIA property aria-activedescendant assigned to the menu container. With this attribute, screen readers can detect which descendant element of an interactive control is currently selected.

The context menu implemented as described above should usually cause a screen reader to announce the menu and its items and to switch into a state comparable to application mode (JAWS) or focus mode (NVDA) in order to pass the keys for the menu navigation to the application. While this worked as expected with NVDA on both Mozilla Firefox and Google Chrome, JAWS only recognised the menu reliably in conjunction with Firefox. On Chrome, by contrast, JAWS kept displaying the selected data point when using the virtual cursor in combination with certain user settings (see Appendix B). Several attempts were made to overcome this problem, such as appending the menu container element to different DOM nodes and assigning the ARIA property aria-haspopup to the data point objects. Another option was to include the menu element into the DOM tree already at startup with the hidden attribute set and opening the menu by removing this attribute. Moreover, modifying the indication of the selected menu item by assigning each item a different id attribute and changing the value of the property aria-activedescendant accordingly was also tried. Despite these intensive efforts, no satisfactory solution to this problem could be found.

7.4.3 Speech Synthesis

The Speech class of AChart Interpreter is in charge of all communication with any speech synthesiser available through the Web Speech API [MDN 2019], which provides an integrated speech service to the instances of the UserInterface and the ContextMenu class. At the beginning of the class declaration, the Speech class defines a boolean constant `ON_BY_DEFAULT`,which determines whether speech output should be enabled at startup. For the version described in this thesis, this constant is set to `true`.

Since some browsers and operating systems provide for the temporary download of speech engines on request, a first step is to start this download as soon as possible [Bodner et al. 2020a, page 10]. This is performed at the beginning of the Speech constructor. As the Speech instance is created within the UserInterface constructor, the download is initiated immediately after launching the application and can take place while the user is choosing a chart.

After this initialisation, the Speech constructor tests the availability of any speech engine by calling the interface window.speechSynthesis. If speech is supported by the current configuration, this interface returns

```
addItem(text_key : string, run : Function,
    position? : Item, append = false) : HTMLElement
{
  let item : Item =
  {
    run: run,
    element: Helper.appendHTML(this.menu_element,
        HTMLTemplate.MENU_ITEM),
    previous_item : undefined,
    next_item: undefined
  };

  // If no item has been added yet, set this.first_item to this item
  // and select it as default:
  if (!this.first_item)
  {
    this.first_item = item;
    this.selectItem(item);
  }
  // Otherwise, link the item with the one previously added:
  else
  {
    item.previous_item = this.last_item;
    this.last_item.next_item = item;
  }

  // Every new item is the last item of the menu so far:
  item.next_item = this.first_item;
  this.last_item = item;
  this.first_item.previous_item = item;

  this.item_count++;

  // Text (incl. hotkey) displayed for the item:
  item.element.textContent = Text.MENU_ITEM[text_key].text
      + " (" + Text.MENU_ITEM[text_key].hotkey + ")";
  item.element.setAttribute("aria-keyshortcuts", Text.MENU_ITEM[text_key].hotkey);

  // Assign corresponding hotkey to the item:
  this.items[Text.MENU_ITEM[text_key].hotkey] = item;

  return item.element;
}
```

Listing 7.8: Excerpt from the source code of the method addItem() implemented in the class ContextMenu. The method creates a context menu item by appending a new <button> element to the menu container and writing the appropriate text label to it. The data of the menu item are stored in a structure of type Item, where the member variables for the previous and the next item are set according to the items already present within the menu. This structure is then stored in the instance variable `items` of type Map<string,Item> under the associated hot key.

```
    this.menu_element.addEventListener("keydown", (event : KeyboardEvent) =>
    {
      event.preventDefault();
      event.stopPropagation();

      if ( (event.shiftKey) || (event.altKey) || (event.ctrlKey) || (event.metaKey)
          )
      {
        return;
      }
      let key = "";
      if (event.key)
      {
        key = event.key.toLowerCase();
      }

      switch (key)
      {

        case "arrowdown":
          this.removeSelection();
          this.selectItem(this.selected_item.next_item);
          this.user_interface.speech.speak(
              this.selected_item.element.textContent);
          break;

        ...
        default:
          // If the key is among the defined hot keys, call the function
          // assigned to the corresponding menu item and close the menu:
          if (this.items[key])
          {
            this.close();
            this.items[key].run();
          }

      }

    });
```

Listing 7.9: Excerpt from the source code of the constructor for ContextMenu instances. In the `switch` block of the callback function for the event listener attached to the menu container, the key `Cursor-Down` is processed in order to select the subsequent menu item. The latter does not need to be calculated, but is given by the member variable `next_item` of the structure for the current item. A hot key press is evaluated in the `default` case by querying the map `items` for it and executing the function stored in the `run` member of the corresponding item structure.

an instance of the class SpeechSynthesis, otherwise null. In the case that speech is available and shall be enabled by default, the constructor calls the method setSynthesiser(), which stores the SpeechSynthesis instance in the variable synthesizer of the Speech instance and calls the method setVoice(). In addition, it attaches a listener to the SpeechSynthesis instance which is triggered in the case that voices have been downloaded from a remote server and the loading process has been completed. Its callback function then invokes setVoice() another time if no voice has yet been initialised.

The method setVoice() of the Speech instance detects the voices currently available and chooses an appropriate one according to the preferred language. It therefore calls the method getVoices() of the SpeechSynthesis instance. This method returns an array of all the voices provided, where each voice is represented by an instance of the class SpeechSynthesisVoice. In order to find a voice for a particular language, the array of voices is searched for a SpeechSynthesisVoice instance whose member variable lang starts with a certain string. This variable is specified to contain a language tag as defined in [IETF 2009], for example en-US for American English.

At the time of writing, the user interface of AChart Interpreter exists in English only, so only a voice for English is searched for. However, the facility to search for voices of arbitrary languages is already implemented. If no suitable voice has been found by the lang instance variable, the SpeechSynthesisVoice array is searched for an instance whose name variable, converted to a lower-case representation, contains the string english. If this query does not yield any result either, the first entry of the SpeechSynthesisVoice array returned by the getVoices() method is considered. In any case, the voice is stored in the variable voice of the Speech instance.

The public method stop() of the Speech instance interrupts any ongoing speech by calling the method cancel() of the SpeechSynthesis instance. It is invoked when pressing any key as well as by clicking anywhere within the GUI in order to provide the possibility to immediately output a new message. Without interrupting the speech, the new information would be stored in a message queue and spoken only after completing the current output, which would hinder rapid keyboard navigation by blind users.

The method speak() of the Speech instance takes a text string as input and sends it to the speech engine. For this purpose, it first tests if a voice has been successfully initialised. If this is not the case, it calls the method setVoice() another time. Afterwards, an instance of the class SpeechSynthesisUtterance is created and initialised with the text to be read aloud. Moreover, the SpeechSynthesisUtterance instance needs to be passed the voice and the tag for the language to be used, which is performed by setting its member variables voice and lang accordingly. Finally, the SpeechSynthesisUtterance instance is passed to the method speak() of the SpeechSynthesis instance.

The public method readElement() is provided to read the text content of a focused GUI object. It expects the reference of this object as an argument and reads its text content as well as its element name, and, in some cases, its ARIA role. Depending on the element name and the role, the content of the object is concatenated with certain text indicators, such as “button” for button objects, “expanded”, or “collapsed” for an open or a closed node of the chart tree. The composed string is then read aloud by the speak() method.

Speech synthesis is disabled or enabled by the public toggle() method. If speech is active at the moment, it destroys the current SpeechSynthesis instance, otherwise it creates a new one. This method also takes care of speaking the appropriate message when turning speech on or off and changing the text label of the corresponding menu button.

Chapter 8

Outlook and Future Work

Although the AChart suite described in the practical part of this thesis is equipped with powerful features for the creation and exploration of accessible charts, there is still much potential for improvements and enhancements. All the components of the AChart software were implemented in a modular design in order to provide the basis for future extensions. The source code of AChart Creator is publicly available at [Kopel, Andrews, Mendoza Estrada, Grass et al. 2021], that of AChart Interpreter and AChart Summariser at [Kopel, Andrews, Mendoza Estrada, Bodner et al. 2021a], and any helpful contribution will be highly appreciated. Some ideas for desirable future development will be described in Section 8.2, embedding them into general trends observed in the field. Among others, an essential goal is the support for more of the numerous well-known chart variants. This, however, requires a system of ARIA roles and properties covering the wide range of chart objects occurring in such charts to be developed. For this reason, several potential extensions to the AChart taxonomy will be discussed first.

8.1 Evolving an Extended AChart Taxonomy

In Section 5.3, an ARIA-based system of roles and properties for charts was proposed. This taxonomy builds upon the system used by Describler (see Subsection 4.5.1) and modifies it in order to overcome certain insufficiencies of the original approach. Nevertheless, the resulting system still meets only the requirements of bar, line, and pie charts with one or more data series. In pursuit of support for a larger set of common chart types, it seems reasonable first to explore other types of charts which use *tabular data*. As a first step, new values for the property aria-charttype should be defined, including but not restricted to scatter, parallelcoordinates, star, donut, and gantt. As scatter plots can be classified as cartesian charts with characteristics similar to those of bar and line charts, merely setting aria-charttype to scatter might suffice to express the semantics of this chart type by means of the AChart taxonomy, provided that variation in colour, size, or shape of the data point symbols can be represented by distinct data series. Similarly, donut charts could be annotated in the same manner as pie charts applying the corresponding property value for the chart type.

Parallel coordinates charts and star plots, by contrast, involve different axis types. For this reason, it is proposed to use the role axis to define an axis in a chart with more than two axes. The property aria-variable could then be defined to express its variable or index number. In addition, the standard ARIA property aria-orientation could be applied to indicate the visual orientation of an axis, where possible values could include horizontal, vertical, radial, and other. The data points for such charts are truly multidimensional tuples, which cannot in general be represented as name-value pairs, but could still contain one descendant element per component with role heading or datavalue, depending on the context. For unambiguous annotation of these elements, the standard ARIA property aria-labelledby could be attached to each of them in order to reference its associated axis. The strategy of allowing multiple data point descendants with role datavalue might represent a general solution for all those cases where data points have more than one value; for instance, in Gantt charts with a starting and an ending value on a time axis for each data point.

Here, the property aria-labelledby could be used to point to the label associated with the value (in the Gantt chart example, “start” or “end”).

The current AChart taxonomy adopts the Describler role dataset for compatibility reasons and uses it to denote data series. Describler does not recognise data series as such, but can handle multiple datasets. Since the Describler project has not been continued since 2015, however, it seems reasonable to introduce the dedicated role dataseries into future versions of the AChart taxonomy.

8.2 Software Enhancements

As one of the first steps for the further development of AChart Interpreter, it would be desirable to solve the minor problems with regard to screen reader interaction described in Subsection 7.1.3. It is assumed, however, that this will require cooperation with the developers of the respective screen readers.

In both AChart Creator and AChart Interpreter, it is intended to implement support for other chart types. This could first include scatter plots and donut charts, as well as charts of tabular data with higher dimensions, such as parallel coordinates charts, star plots, and similarity maps, which do not significantly differ from the already supported types with regard to their data structures. For AChart Creator, the visualisation of multiple data series should be added to more chart types as well.

In a second step, other kinds of visualisations could be considered, for instance, flow diagrams and hierarchy visualisations. Similarly, both AChart Creator and AChart Interpreter could be extended to a larger set of ARIA-based systems for chart annotation, such as the pseudo-table (see Subsection 4.5.4) and the pseudo-list approach (see Subsection 4.5.5), as well as a system based on the WAI-ARIA Graphics module (see Subsection 4.5.2), all possibly enhanced as proposed in Section 5.2.

The functionality of AChart Interpreter could be extended by other analytical features. For example, it would be possible to compare a data point not only to the data series it belongs to, but also to different ones. As an additional option for obtaining an overview of the data, a sonification feature could be implemented, starting with sequential playback of a data series, as provided in most of the solutions presented in Chapter 3. Going even further, AChart Interpreter’s navigation commands could be applied to sonification functionality, too, where the currently selected data point is played, so that the user is in full control of the playback. This option could be combined with the existing textual output, which might further help some users to understand the chart.

It would be possible to add interaction with touch-sensitive tactile graphics displays like the Graphiti by Orbit [2021] (see Subsection 3.1.2). Like with the visual and textual representation in the Graphic Panel and Text Panel, synchronised navigation should be possible, so that the chart object currently focused in AChart Interpreter is clearly highlighted on the tactile display and, conversely, touching an object within the haptic representation sets the focus of AChart Interpreter to its counterpart in the Graphic Panel and Text Panel. This way, AChart Interpreter could be extended to a system offering multimodal exploration, where the accessible information of the currently touched chart object is read aloud by speech synthesis, displayed in Braille, and/or sonified, all flexibly combined according to the user’s preferences. The open application programming interface (API) provided by Graphiti for both input and output as well as AChart Interpreter’s selection of graphical objects by mouse are a promising foundation for the implementation of this feature.

Furthermore, the chart description displayed in the text panel for each chart root object could be enhanced with further characteristics of the data, such as a brief description of the trends shown by the data series. As an alternative, AChart Interpreter could integrate a sophisticated algorithm which generates overall descriptions of the chart, like the one applied used the SIGHT system by Carberry et al. [2012] and Moraes et al. [2014] introduced in Section 3.4. Such a description may represent another helpful means to rapidly obtain an idea of a chart’s content immediately when encountering it, so as to better decide whether to further explore it. It can be assumed that research in the field of artificial intelligence will lead to even more powerful solutions over the coming years, which could then be combined with AChart Interpreter.

As mentioned in Subsection 7.4.1, the iterative development process of AChart Interpreter was accompanied by informal accessibility and usability tests. Nevertheless, there is scope for further formative usability evaluation, such as a heuristic evaluation or thinking aloud test. As shown in Chapter 3, presenting charts to visually impaired recipients is the subject of extensive research. For this reason, a comparative study considering some of the presented solutions and AChart Interpreter would certainly yield highly interesting results.

Finally, as a longer-term goal, it might be possible to publish a version of AChart Interpreter as a plug-in which can be integrated into web browsers or screen readers, giving a visually impaired user the possibility to explore charts on arbitrary web pages. With the implementation of the software, certain preparations for this purpose were already taken into account, such as the search for SVG documents embedded in any loaded web page and the assignment of event listeners to them which start the analysis process.

Chapter 9

Concluding Remarks

This thesis examined several aspects related to the problem of presenting charts to visually impaired recipients and described a new solution. After an introduction to the field, Chapter 2 explained the concepts of information accessibility and described some of the assistive technologies available for blind and partially sighted users. Based on this knowledge, essential requirements and techniques for producing and handling accessible web documents were summarised. In particular, the WAI-ARIA system was introduced, which represents a basis for the practical part of this thesis.

The next Chapter 3 gave an overview of different approaches for enabling visually impaired persons to efficiently consume charts, concentrating on user interaction and various possible output modalities. In Chapter 4, current approaches for the inclusive design of graphics documents were discussed. The SVG format was introduced as an appropriate means for defining accessible graphics, and several options for embedding accessible information in SVG were explained. Of special relevance is Section 4.5, as it provided an extensive survey of systems for annotating SVG charts with semantic information and the underlying chart data in a machine-readable format. ARIA-based proposals were described, and current practice as applied in the charting libraries Highcharts, FusionCharts, amCharts, Semiotic, and AnyChart was documented.

Chapter 5 acted as a connection between the theoretical and the practical part of this thesis. It first gave an introduction to the AChart project including a list of all its participants with their contributions. It then discussed some of the practical and research approaches and argued that none of the approaches presented in Chapter 3 represents an open, ARIA-based solution to the problem at issue, which also fulfils the criteria of extensibility, public availability, and screen reader compatibility. Hence, the motivation for the AChart project. In Section 5.2, the advantages and drawbacks of the SVG-based taxonomies presented in Section 4.5 were analysed, and possible extensions to some of the ARIA-based systems were suggested. Finally, AChart's ARIA taxonomy, based on that of Describler, was introduced.

AChart Creator, a command-line programme for generating charts in SVG format from CSV data and annotating them with accessible markup was described in Chapter 6. The software was first developed by Grass et al. [2019] and enhanced in the context of this thesis. The current version 2.0.0 supports the creation of bar, pie, and line charts, where for the latter chart type, multiple data series can be visualised. It has been shown that it is possible to create highly accessible charts by means of D3 and to automatically save them to SVG files.

AChart Interpreter and AChart Summariser are presented in Chapter 7. They are two related, powerful applications developed as practical part of this thesis which analyse accessible SVG charts and present them in a textual form. AChart Interpreter provides a graphical user interface with an intuitive visual representation, screen-reader-friendly navigation, and optional integrated speech output. On the one hand, both programmes are intended to provide developers and chart authors with a tool for testing the accessibility of SVG charts, in particular, their compliance with the AChart taxonomy. Simultaneously, they enable blind users to rapidly and efficiently explore and understand accessible charts, by providing an overview of the relevant chart objects, a summary of the chart, and the possibility to view and compare all data points of the chart as well as to obtain statistical information on its data. In informal accessibility and usability tests with four blind participants, the application received highly positive feedback.

All the components of the AChart software were implemented in a class-based, modular design in order to facilitate future modifications and extensions. Some ideas for possible enhancements were summarised in Section 8.2. These include the support for other common chart types and publishing AChart Interpreter as a browser extension which can be used by blind persons to read charts on arbitrary web pages. Nevertheless, an important prerequisite for the universal usage of AChart Interpreter or any similar assistive solution is the annotation of charts according to a system which is widespread and includes a large number of possible chart objects. The current AChart taxonomy and most of the other approaches presented in Section 4.5, however, are not compatible among each other and only cover line, bar, and pie charts. For this reason, several extensions to the AChart taxonomy were proposed in Section 8.1. A longer-term goal of the AChart project is to establish a standard for semantically enriching digital charts and to provide a software module supplementing the capabilities of browsers and screen readers. It is hoped that this thesis will make a valuable contribution to these developments and that a general standard for annotating charts will soon be developed and agreed on, which could then be supported by a significant number of chart authoring tools, charting libraries, browsers, and assistive software.

Appendix A

AChart Creator Help

AChart Creator – Creates a chart in SVG format with WAI-ARIA markup from a CSV file.

Command-line syntax:

```
acreate [--chart] CHART-TYPE [--dataset CSV-FILENAME]
        [--output SVG-FILENAME] [--chart-title TITLE]
        [--chart-desc DESCRIPTION] [--x-axis-title TITLE]
        [--y-axis-title TITLE] [--legend-title TITLE]
        [--target SOFTWARE] [--column DATA-COLUMN] [--no-sort]
        [--no-legend] [--no-tooltips] [--no-bar-values]
        [--no-segment-values] [--no-segment-percentages]
        [--segment-percentage-precision PLACES] [--svg-precision
           PLACES]
        [--version] [--help]
```

Mandatory arguments:

- `CHART-TYPE`: Specifies the type of the chart to be created. Currently supported chart types are bar, line, and pie (case-insensitive). This argument can be given either as the first command-line parameter or, alternatively, at any position, prepended by `--chart`.

Optional arguments:

- `--dataset CSV-FILENAME`: Specifies the CSV file containing the data to be visualised in the chart. If not specified, a default CSV file will be chosen.
- `--output SVG-FILENAME`: Specifies the name of the resulting SVG file. If not specified, the output file will be named according to the input filename with the extension .svg and placed into its directory.
- `--chart-title TITLE`: Specifies a title for the chart which is visible and accessible by screen readers. If not specified, the title will be derived from the headers of the CSV columns.
- `--chart-desc DESCRIPTION`: Assigns the chart a more detailed overall description in addition to the title. The description is not visible but can be obtained by screen readers.
- `--x-axis-title TITLE`: Specifies a title for the x-axis (if applicable). The title is visible and accessible by screen readers. If not specified, it will be derived from the header of the first CSV column. For pie charts, the option has no effect. Alias: `--x-title TITLE`
- `--y-axis-title TITLE`: Specifies a title for the y-axis (if applicable). The title is visible and accessible by screen readers. If not specified, in the case of a single-series bar or line chart, it will be derived from the header of the data column. For pie charts, the option has no effect. Alias: `--y-title TITLE`
- `--legend-title TITLE`: Specifies a title for the legend (if applicable). The title is visible and

accessible by screen readers. If not specified, in the case of a pie chart, it will be derived from the header of the first CSV column. For multi-line charts, the legend title defaults to "Legend". If no legend is printed, the option has no effect.

- `--target SOFTWARE`: States which assistive software the accessibility markup shall be optimised for. Valid arguments for `SOFTWARE` are (case-insensitive):
  - `achart` for AChart Interpreter (default).
  - `describler` for Describler.
  - `screen-reader` for common screen readers (JAWS, NVDA, etc.) interacting with browsers without any special chart reading software.

  This is meant to optimise the user experience for the respective target; in general, all the three targets should work with all the named types of assistive software.
- `--column DATA-COLUMN`: Specifies the CSV column containing the data series to be visualised.`DATA -COLUMN` is an integer > 0, where the columns are assumed to be counted with increasing numbers from left to right. If the option is not given, in the case of a line chart, all columns of the CSV file, starting by number 1, will be visualised, where one line (data series) corresponds to one column. For bar and pie charts, `DATA-COLUMN` defaults to 1.
- `--no-sort`: By default, all data points are sorted in increasing order by name, i.e. the content of the corresponding cells of the first CSV column. If this option is given, the data points will instead be visualised in the order their corresponding lines are listed in the CSV file from top to bottom.
- `--no-tooltips`: Suppresses all tooltips on mouse-over (<title> elements). This may slightly impair the optimisation for Describler, even if the option `--target describler` is used.
- `--no-legend`: Suppresses the creation of a legend. Unless `--no-tooltips` is given as well, in the case of a multi-line chart, each line will still be given a <title> element with the corresponding data series title, which is accessible by screen readers and visible as a tooltip on mouse-over. For pie charts, labels are instead displayed within their corresponding pie segments.
- `--no-bar-values`: Suppresses the visual labelling of bars in bar charts with data point values. The bar values are then available as tooltips, unless `--no-tooltips` is given as well. For other chart types, the option has no effect.
- `--no-segment-values`: Suppresses the visual labelling of pie segments in pie charts with data point values. The segment values are then available as tooltips, unless `--no-tooltips` is given as well. For other chart types, the option has no effect.
- `--no-segment-percentages`: Suppresses the visual labelling of pie chart segments with data point percentages. The percentages are then available as tooltips, unless `--no-tooltips` is given as well. For other chart types, the option has no effect.
- `--segment-percentage-precision PLACES`: Specifies the number of decimal places for rounding percentages in labels or tooltips of pie chart segments. `PLACES` must be an integer ≥ 0. The default is 1. For other chart types, the option has no effect.
- `--svg-precision PLACES`: Specifies the number of decimal places for rounding SVG coordinates and lengths. `PLACES` must be an integer ≥ 0. The default is 3.
- `--version`: Prints version information and exits.
- `--help`: Prints this help message and exits.

All parameters and options are case-insensitive. If an argument contains spaces, enclose it in double quotation marks (""). File names may contain relative or absolute paths. If no path is given, the current working directory is assumed.

Appendix B

AChart Interpreter Help

Help

Welcome to AChart Interpreter!

AChart Interpreter (standing for "Accessible Chart Interpreter") is a screen reader for charts. It gives you a description of charts created in SVG format. The chart needs to contain appropriate WAI-ARIA markup as generated, e.g., by the tool AChart Creator. If you are interested in more details, AChart Interpreter lets you further explore a chart by keyboard navigation in combination with either a built-in speech synthesis or the screen reader of your choice.

Usage

To start AChart Interpreter, open the graphic you want to explore. You can either choose a sample chart from the drop-down list or load an SVG file from your computer using the "File Upload" button. After opening an SVG file, it will be displayed in the part of the window named Graphic Panel. In the Text Panel, the textual information will be shown.

To move around within AChart Interpreter's windows, use the `Tab` key or the appropriate cursor of your screen reader (for example, the virtual cursor in JAWS or the cursor of the browse mode in NVDA, both controlled by the `Arrow` keys). To close any window, you can use the `Esc` key or the "Close" button at the top of the window.

In the main window of AChart Interpreter, next to the "File Upload" button, you find a button to switch the built-in speech synthesis on and off. You can also do this by pressing `Alt+S`. If the speech is enabled, it reads aloud every element when focusing it. The speech can be interrupted pressing any key or clicking anywhere within AChart Interpreter's window. Note that the integrated speech feature depends on your browser and the voices installed on your local machine and that it is not available in Internet Explorer.

The next button toggles the list view for the data items to Application Mode and back to Document Mode (see below).

This help text can be shown again using the "Help" button or the `F1` key.

Underneath these buttons, you will find the Graphic Panel and the Text Panel. In the Graphic Panel, the opened visualisation will be displayed. The button "Remove SVG" can be used to close the graphic again. The Text Panel shows the title and description of the graphic as well as a list of all charts and chart components contained by it. All chart components can be selected using the `Tab` key. Mouse users can also select any component by a left click either on the graphical object or its textual counterpart. The selected component will be visually highlighted in both the graphical and the textual representation. The style of highlighting for the graphical objects can be chosen from two options by the switch "Toggle SVG highlighting mode".

To hide or unveil the information on a particular chart, select its title and close or open its details, respectively, by pressing `Enter` or `Space` or left-click on the triangle symbol. When the details for a chart have been opened, they are displayed right underneath its title. Below, you will find a list of all data series within the chart. To hide or unveil the information on a particular data series, again, select its title and close/open its details by pressing `Enter` or `Space` or left-click on the triangle symbol.

When the details for a data series have been opened, you will find a list of all items (that is, all values, all data points) contained in this data series. Using the combo box above this list, you can choose if the items are displayed either in increasing order ("from lowest to highest value") or in decreasing order ("from highest to lowest value"). With the default choice "in original order", the items are listed in the same order they appear in the SVG source code, that is, in most cases, along the x-axis from left to right. To choose a certain sorting option, open the combo box, select the option by using the `Arrow-Up` and `Arrow-Down` keys and then press `Enter`.

Once you have reached the list for a data series, use the `Arrow-Up` and `Arrow-Down` keys to navigate through it. If no screen reader is running or Application Mode is activated (see below), you can use the following additional navigation keys: the `Home` key will move the selection to the first item in the list; the `End` key will take you to the last item. If there are multiple data series contained in the chart, you can quickly move between their data lists using the `Arrow-Left` and `Arrow-Right` keys.

Moving to an item and pressing the `Context-Menu` key or right-clicking on it, a context menu will appear offering several options: you can view statistics about the particular item or a list of comparisons of this item to all others in the same data series. Moreover, you can move to the first or the last item within the data list, which might be helpful in case of longer data series.

Pressing `Tab` within the list for a certain data series will take you to the button underneath this data list. Using this button, you can obtain statistics about the whole data series.

Application Mode

Most screen readers like JAWS and NVDA have their own special keyboard setting for navigating Web pages, which might block the functionality of some of AChart Interpreter's navigation keys (like the `Home` and `End` keys). If you are using a screen reader and would like to benefit from all of AChart Interpreter's key commands, switch to Application Mode using the corresponding button at the top of the window or pressing `Alt+A`. When you move to a data list afterwards, your screen reader should enable a special mode in which all keystrokes are passed through to AChart Interpreter (in JAWS this is also known as Application Mode or Forms Mode, in NVDA it is called Focus Mode). If your screen reader does not activate this mode automatically, try pressing `Enter` after moving to the data list. Note that after switching to Application Mode, the data lists might look different and that NVDA may announce them as "Application".

Using the same button as before or pressing `Alt+A` again, you can switch back to AChart Interpreter's standard mode, called Document Mode.

Known Issues

AChart Interpreter does not yet support Internet Explorer at the moment. It is recommended to use the current version on Mozilla Firefox.

On Google Chrome, JAWS might not recognise a context menu correctly after opening it. If so, try using AChart Interpreter's Application Mode or enabling automatic forms mode of JAWS.

When AChart Interpreter is switched to Application Mode, JAWS might not announce all elements outside the data lists appropriately. If so, try leaving forms mode and enabling the virtual cursor of JAWS manually by pressing `Numpad +` (desktop keyboard) or `JAWS+;` (laptop keyboard).

Appendix C

AChart Summariser Help

AChart Summariser – Outputs a textual summary of an SVG chart.

Command-line syntax:

```
asummarise [--output OUTPUT-FILENAME] [--statistics]
           [--datapoints] [--version] [--help]
           [--input] SVG-FILENAME
```

Mandatory arguments:

- `SVG-FILENAME`: Specifies the SVG file to analyse. This argument can be given either as the last command-line parameter or, alternatively, at any position prepended by `--input`.

Optional arguments:

- `--output OUTPUT-FILENAME`: Writes the output to the specified plain text file. If not specified, output is written to stdout.
- `--statistics`: Additionally outputs statistical information on each data series.
- `--datapoints`: Additionally outputs all the data points in the chart.
- `--version`: Prints version information and exits.
- `--help`: Prints this help message and exits.

All options are case-insensitive. If an argument contains space characters, it should be enclosed in quotation marks (""). File names may contain relative or absolute paths. If no path or relative path is given, the current working directory is assumed.

Bibliography

Altmanninger, Kerstin and Wolfram Wöß [2008]. *Accessible Graphics in Web Applications: Dynamic Generation, Analysis, and Verification*. Computers Helping People with Special Needs: 11th International Conference (ICCHP 2008) (Linz, Austria). Edited by Klaus Miesenberger, Joachim Klaus, Wolfgang Zagler, and Arthur Karshmer. Volume 5105. Lecture Notes in Computer Science. Berlin / Heidelberg, Germany: Springer-Verlag, 09 Jul 2008, pages 378–385. doi:10.1007/978-3-540-70540-6_55 (cited on pages 6, 21, 31, 90).

amCharts [2020a]. *Accessibility in amCharts 4*. Documentation. 2020. https://amcharts.com/accessibility/accessible-charts (cited on pages 39, 79).

amCharts [2020b]. *amCharts 4 Demos*. 2020. https://amcharts.com/demos (cited on pages 39, 79).

American Thermoform [2019]. *Tactile Graphics Machines*. 2019. http://americanthermoform.com/product-category/tactile-graphics-machine (cited on page 23).

Andrews, Keith [2018]. *Writing a Thesis: Guidelines for Writing a Master's Thesis in Computer Science*. Graz University of Technology, Austria. 30 Dec 2018. https://ftp.isds.tugraz.at/pub/keith/thesis/ (cited on page xiii).

AnyChart [2020a]. *Accessibility Settings*. Documentation. 2020. https://docs.anychart.com/Common_Settings/Accessibility/Settings (cited on pages 5, 39, 86).

AnyChart [2020b]. *AnyChart Accessibility Demos. Accessible SVG Charts with WAI-ARIA Support*. 2020. https://anychart.com/accessibility/a11y-demos (cited on pages 39, 86).

Apple [2020a]. *Vision Accessibility – iPad*. 2020. https://apple.com/accessibility/ipad/vision (cited on pages 10, 12).

Apple [2020b]. *Vision Accessibility – iPhone*. 2020. https://apple.com/accessibility/iphone/vision (cited on pages 7, 10, 12).

Apple [2020c]. *Vision Accessibility – Mac*. 2020. https://apple.com/accessibility/mac/vision (cited on pages 7, 10).

Banf, Michael and Volker Blanz [2013]. *Sonification of Images for the Visually Impaired Using a Multi-Level Approach*. Proc. 4th Augmented Human International Conference (AH '13) (Stuttgart, Germany). 07 Mar 2013, pages 162–169. doi:10.1145/2459236.2459264. http://staff.www.ltu.se/~kalevi/References/Sonification%20of%20images%20for%20the%20visually%20impaired%20using%20a%20multilevel%20approach.pdf (cited on page 29).

Bliss, James C. [1969]. *A Relatively High-Resolution Reading Aid for the Blind*. IEEE Transactions on Man-Machine Systems 10.1 (Mar 1969), pages 1–9 (cited on pages 26–27).

Bodner, Lukas, Daniel Geiger, and Lorenz Leitner [2020a]. *Accessible Charts with AChart Reader*. Graz University of Technology, 29 Jun 2020. https://courses.isds.tugraz.at/ivis/projects/ss2020/ivis-ss2020-g1-project-achart-reader.pdf (cited on pages 90, 148).

Bodner, Lukas, Daniel Geiger, and Lorenz Leitner [2020b]. *AChart Reader Showcase Video*. Graz University of Technology, 17 Jun 2020. `https://youtu.be/7unJ2aU9ghc` (cited on page 121).

Bostock, Mike [2021]. *D3 – Data-Driven Documents*. 29 Mar 2021. `https://d3js.org/` (cited on pages 3, 97, 115).

Boyd, Caleb, James M. Greene, and Jared Allard [2020]. *nexe*. 05 Apr 2020. `https://github.com/nexe/nexe` (cited on pages 3, 115, 143).

Braier, Jonas, Katharina Lattenkamp, Benjamin Räthel, Sandra Schering, Michael Wojatzki, and Benjamin Weyers [2014]. *Haptic 3D Surface Representation of Table-Based Data for People with Visual Impairments*. ACM Transactions on Accessible Computing 6.1 (Dec 2014), 1:1–1:35. ISSN 1936-7228. doi:10.1145/2700433 (cited on pages 23, 25).

Bublitz, Blaine and Eric Schoffstall [2021]. *gulp. A Toolkit to Automate & Enhance Your Workflow*. 08 Feb 2021. `https://gulpjs.com/` (cited on pages 89, 115).

Bulatov, Vladimir, John A. Gardner, and Holly Stowell [2005]. *The ViewPlus IVEO Scalable Vector Graphics Technology for Universally Usable Complex Information*. Proc. 11th International Conference on Human-Computer Interaction: Universal Access in HCI (HCI International 2005) (Las Vegas, NV, USA). Volume 5 – Emergent Application Domains in HCI. 22 Jul 2005. `https://viewplus.com/about/abstracts/05hcibulatov.html` (cited on pages 30, 90).

Carberry, Sandra, Stephanie Elzer Schwartz, Kathleen McCoy, Seniz Demir, Peng Wu, Charles Greenbacker, Daniel Chester, Edward Schwartz, David Oliver, and Priscilla Moraes [2012]. *Access to Multimodal Articles for Individuals with Sight Impairments*. ACM Transactions on Interactive Intelligent Systems 2.4 (Dec 2012), 21:1–21:49. ISSN 2160-6455. doi:10.1145/2395123.2395126. `http://assets.ctfassets.net/kdr3qnns3kvk/6HYQd4VEtOqsKIeMEGEOOO/6dc01d7477738dd2ad8ac4efb8c6eb01/tiis-a21-carberry.pdf` (cited on pages 21, 23, 32, 154).

Choi, Stephen H. and Bruce N. Walker [2010]. *Digitizer Auditory Graph: Making Graphs Accessible to the Visually Impaired*. CHI '10 Extended Abstracts on Human Factors in Computing Systems (CHI EA '10) (Atlanta, GA, USA). New York, NY, USA: ACM, 10 Apr 2010, pages 3445–3450. doi:10.1145/1753846.1753999. `http://sonify.psych.gatech.edu/publications/pdfs/2010CHI-ChoiWalker.pdf` (cited on pages 23, 29).

D3plus [2019]. *Accessibility*. Documentation. 06 Dec 2019. `https://d3plus.org/accessibility` (cited on page 64).

DIAGRAM Center [2017]. *Poet Image Description Tool*. Benetech, 2017. `https://poet.diagramcenter.org/` (cited on page 21).

Dürnegger, Bernhard, Christina Feilmayr, and Wolfram Wöß [2010]. *Guided Generation and Evaluation of Accessible Scalable Vector Graphics*. Computers Helping People with Special Needs: 12th International Conference (ICCHP 2010) (Vienna, Austria). Edited by Klaus Miesenberger, Joachim Klaus, Wolfgang Zagler, and Arthur Karshmer. Volume 6179.I. Lecture Notes in Computer Science. Berlin / Heidelberg, Germany: Springer-Verlag, 14 Jul 2010, pages 27–34. doi:10.1007/978-3-642-14097-6_5 (cited on pages 31, 45–46, 90).

Duxbury [2020]. *Product Info: QuickTac*. Duxbury Systems, 25 Apr 2020. `https://duxburysystems.com/product2.asp?product=QuickTac` (cited on page 25).

Enabling Technologies [2015]. *Cyclone, Trident and Phoenix (gold) Braille Embosser User's Manual*. Enabling Technologies. Apr 2015. `https://web.archive.org/web/20200113224602/http://www.brailler.com/images/downloads/manuals/CyclonePhoenixgoldTridentManual.pdf` (cited on page 24).

Enabling Technologies [2017]. *Enabling Technologies – the Supreme Braille Dot Quality!* 2017. `https://web.archive.org/web/20201020035404/http://www.brailler.com/` (cited on pages 24–25).

Engel, Christin, Emma Franziska Müller, and Gerhard Weber [2019]. *SVGPlott: an Accessible Tool to Generate Highly Adaptable, Accessible Audio-Tactile Charts for and from Blind and Visually Impaired People*. Proc. 12th ACM International Conference on PErvasive Technologies Related to Assistive Environments (PETRA '19) (Rhodes, Greece). New York, NY, USA: ACM, 05 Jun 2019, pages 186–195. doi:10.1145/3316782.3316793 (cited on pages 31, 49, 90).

EPRS [2018a]. *Assistive Technologies for People with Disabilities. In-Depth Analysis*. Science and Technology Options Assessment. European Parliamentary Research Service, 22 Feb 2018. 15 pages. doi:10.2861/422217. `https://op.europa.eu/s/oUy3` (cited on page 6).

EPRS [2018b]. *Assistive Technologies for People with Disabilities. Part II: Current and Emerging Technologies*. Science and Technology Options Assessment. European Parliamentary Research Service, 22 Feb 2018. 73 pages. doi:10.2861/567013. `https://op.europa.eu/s/oUy4` (cited on pages 6, 11).

Falk, Eike-Peter [1999]. *Blindengerechte Umsetzung grafischer Bildinformationen in eine taktile Darstellungsform*. Diploma Thesis. Universität Karlsruhe, Jul 1999 (cited on page 26).

Ferres, Leo [2015]. *iGraph: the iGraph-Lite Source Code*. 27 Feb 2015. `https://github.com/leoferres/igraph` (cited on pages 29, 49, 90).

Ferres, Leo, Gitte Lindgaard, Livia Sumegi, and Bruce Tsuji [2013]. *Evaluating a Tool for Improving Accessibility to Charts and Graphs*. ACM Transactions on Computer-Human Interaction 20.5 (Nov 2013), 28:1–28:32. ISSN 1073-0516. doi:10.1145/2533682.2533683 (cited on pages 29, 49, 90).

Ferres, Leo, Petro Verkhogliad, Gitte Lindgaard, Louis Boucher, Antoine Chretien, and Martin Lachance [2007]. *Improving Accessibility to Statistical Graphs: the iGraph-Lite System*. Proc. 9th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS 2007) (Tempe, AZ, USA). New York, NY, USA: ACM, 15 Oct 2007, pages 67–74. doi:10.1145/1296843.1296857 (cited on pages 28, 49, 90).

Ferro, Tyler J. and Dianne T. V. Pawluk [2013]. *Automatic Image Conversion to Tactile Graphic*. Proc. 15th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS 2013) (Bellevue, WA, USA). New York, NY, USA: ACM, 21 Oct 2013, 39:1–39:2. doi:10.1145/2513383.2513406 (cited on page 25).

Fisher, Carie [2019]. *Creating Accessible SVGs*. Blog posting. 10 Jan 2019. `https://deque.com/blog/creating-accessible-svgs` (cited on page 49).

Fitzpatrick, Donal, A. Jonathan R. Godfrey, and Volker Sorge [2017]. *Producing Accessible Statistics Diagrams in R*. Proc. 14th International Web for All Conference on The Future of Accessible Work (W4A 2017) (Perth, Western Australia, Australia). 22. New York, NY, USA: ACM, 02 Apr 2017, 22:1–22:4. doi:10.1145/3058555.3058564. `https://research.birmingham.ac.uk/portal/files/43284212/a22_Fitzpatrick.pdf` (cited on pages 32, 49, 91).

Fredj, Z. Ben and David A. Duce [2006]. *GraSSML: Accessible Smart Schematic Diagrams for All*. Proc. 2006 International Cross-Disciplinary Workshop on Web Accessibility: Building the Mobile Web: Rediscovering Accessibility? (W4A 2006) (Edinburgh, UK). New York, NY, USA: ACM, 23 May 2006, pages 57–60. doi:10.1145/1133219.1133229. `http://ra.ethz.ch/CDstore/www2006-wwa/p57-benfredj.pdf` (cited on pages 23, 49).

FusionCharts [2020]. *Accessibility Extension for FusionCharts Beta*. Documentation. 07 May 2020. `https://fusioncharts.com/extensions/accessibility` (cited on pages 34, 82).

Gardner, John A. [2016]. *Universally Accessible Figures*. Computers Helping People with Special Needs: 15th International Conference (ICCHP 2016) (Linz, Austria, 13 Jul 2016). Edited by Klaus Miesenberger, Christian Bühler, and Petra Penaz. Volume 9758.I. Lecture Notes in Computer Science. Springer International Publishing Switzerland, 06 Jul 2016, pages 417–420. doi:10.1007/978-3-319-41264-1_57 (cited on pages 24, 30, 90).

Gardner, John A. and Vladimir Bulatov [2006]. *Scientific Diagrams Made Easy with IVEO*. Computers Helping People with Special Needs: 10th International Conference (ICCHP 2006) (Linz, Austria). Edited by Klaus Miesenberger, Joachim Klaus, Wolfgang Zagler, and Arthur Karshmer. Volume 4061. Lecture Notes in Computer Science. Springer-Verlag, 11 Jul 2006, pages 1243–1250. doi:10.1007/11788713_179 (cited on pages 1, 21–24, 30, 90).

Gardner, John A. and Vladimir Bulatov [2010]. *Highly Accessible Scientific Graphical Information through DAISY SVG. Improving SVG for Perfect Accessibility*. Proc. 8th International Conference on Scalable Vector Graphics (SVG Open 2010) (Paris, France). 30 Aug 2010. http://graphicalweb.net/2010/papers/56-Highly_Accessible_Scientific_Graphical_Information_through_DAISY_SVG (cited on pages 30, 47, 90).

GitHub [2020]. *Basic Writing and Formatting Syntax – GitHub Docs*. 2020. https://docs.github.com/en/github/writing-on-github/basic-writing-and-formatting-syntax (cited on page 142).

Godfrey, A. Jonathan R. [2013]. *Statistical Software from a Blind Person's Perspective. R is the best, but we can make it better*. The R Journal 5.1 (03 Jun 2013), pages 73–79. ISSN 2073-4859. doi:10.32614/RJ-2013-007. https://journal.r-project.org/archive/2013/RJ-2013-007/RJ-2013-007.pdf (cited on page 32).

Godfrey, A. Jonathan R. [2019]. *The BrailleR Project*. 12 Dec 2019. https://r-resources.massey.ac.nz/BrailleR (cited on page 32).

Godfrey, A. Jonathan R., Paul Murrell, and Volker Sorge [2018]. *An Accessible Interaction Model for Data Visualisation in Statistics*. Computers Helping People with Special Needs: 17th International Conference (ICCHP 2018) (Linz, Austria). Edited by Klaus Miesenberger and G. Kouroupetroglou. Volume 10896. Lecture Notes in Computer Science. Springer International Publishing AG, 11 Jul 2018, pages 590–597. doi:10.1007/978-3-319-94277-3_92 (cited on pages 32, 49, 91).

Goncu, Cagatay and Kim Marriott [2015]. *GraCALC: an Accessible Graphing Calculator*. Proc. 17th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS 2015) (Lisbon, Portugal). New York, NY, USA: ACM, 26 Oct 2015, pages 311–312. doi:10.1145/2700648.2811353 (cited on page 30).

Goncu, Cagatay and Kimbal George Marriott [2008]. *Tactile Chart Generation Tool*. Proc. 10th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS 2008) (Halifax, Nova Scotia, Canada). New York, NY, USA: ACM, 13 Oct 2008, pages 255–256. doi:10.1145/1414471.1414525 (cited on pages 24–25).

Google [2020a]. *Get Started on Android with TalkBack – Android Accessibility Help*. 2020. https://support.google.com/accessibility/android/answer/6283677 (cited on pages 10–12).

Google [2020b]. *Introducing ChromeVox*. 2020. https://chromevox.com/ (cited on page 10).

Google [2020c]. *Use YouTube with a Screen Reader – YouTube Help*. 2020. https://support.google.com/youtube/answer/189278 (cited on page 12).

Grass, Alexander, Lea Novak, and Danica Radulovic [2019]. *Accessible D3 and SVG*. Graz University of Technology, 30 Jun 2019. https://courses.isds.tugraz.at/ivis/projects/ss2019/ivis-ss2019-g2-project-d3-a11y.pdf (cited on pages 90, 97, 115, 157).

Harlan, Gregor, Jens Bornschein, Denise Prescher, and the Tangram team [2019]. *SVG-Plott*. Technische Universität Dresden, 24 May 2019. https://github.com/TUD-INF-IAI-MCI/SVG-Plott (cited on pages 31, 49).

Harpo [2021]. *PIAF (Pictures in a Flash)*. Harpo Assistive Technology, 08 Apr 2021. http://piaf-tactile.com/ (cited on page 23).

Headley, Patrick C. and Dianne T. V. Pawluk [2010]. *A Low-Cost, Variable-Amplitude Haptic Display for Persons Who Are Blind and Visually Impaired*. Proc. 12th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS 2010) (Orlando, Florida, USA). New York, NY, USA: ACM, 25 Oct 2010, pages 227–228. doi:10.1145/1878803.1878844 (cited on page 27).

Help Tech [2021]. *Aids for the Blind and Visually Impaired*. 08 Apr 2021. https://helptech.de/ (cited on pages xiii, 7–9, 26).

Highcharts [2021a]. *Accessibility Module*. 08 May 2021. https://highcharts.com/docs/accessibility/accessibility-module (cited on pages 33, 70).

Highcharts [2021b]. *Accessibility Module Feature Overview*. 08 May 2021. https://highcharts.com/docs/accessibility/accessibility-module-feature-overview (cited on pages 33, 70).

Highcharts [2021c]. *Highcharts for Accessibility*. 08 May 2021. https://highcharts.com/blog/accessibility/ (cited on pages 34, 70).

Hörzu [2020]. *EPGDATA Textprogramm*. 25 Aug 2020. https://hoerzu.de/text (cited on page 14).

Hribar, Victoria E., Laura G. Deal, and Dianne T. V. Pawluk [2012]. *Displaying Braille and Graphics with a "Tactile Mouse"*. Proc. 14th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS 2012) (Boulder, Colorado, USA). New York, NY, USA: ACM, 22 Oct 2012, pages 251–252. doi:10.1145/2384916.2384978. https://researchgate.net/profile/D_Pawluk/publication/262207469_Displaying_braille_and_graphics_with_a_tactile_mouse/links/5400ac280cf2c48563ae5e61/Displaying-braille-and-graphics-with-a-tactile-mouse.pdf (cited on page 27).

HyperBraille [2020]. *HyperBraille*. Mar 2020. http://hyperbraille.de/ (cited on page 26).

IETF [2009]. *Tags for Identifying Language*. Request for Comments 5646. Internet Engineering Task Force, 30 Sep 2009. doi:10.17487/RFC5646. https://rfc-editor.org/rfc/pdfrfc/rfc5646.txt.pdf (cited on page 151).

Index Braille [2020]. *Braille Printers / Braille Embossers by Index Braille*. 2020. https://indexbraille.com/ (cited on page 24).

Irie-AT [2019]. *Braille and Tactile Graphics Embossers for the Visually Impaired*. 2019. https://irie-at.com/braille/braille-embossers (cited on page 25).

ISO [2013]. *International Standard ISO 17049 First Edition: Accessible Design – Application of Braille on Signage, Equipment, and Appliances*. Technical report. 15 Oct 2013 (cited on page 6).

Iwarsson, Susanne and Agneta Ståhl [2003]. *Accessibility, Usability, and Universal Design—Positioning and Definition of Concepts Describing Person-Environment Relationships*. Disability and Rehabilitation 25.2 (2003), pages 57–66. ISSN 1464-5165. doi:10.1080/0963828021000007969. http://olemygind.pbworks.com/f/Accessibility_Iwarsson.pdf (cited on page 5).

jsdom [2021]. *jsdom*. 04 Apr 2021. https://github.com/jsdom/jsdom (cited on pages 3, 115, 143).

Kipke, Siegfried [2006]. *Taktile Kontrollvorrichtung*. DPMA DE 10 2004 046 526. Help Tech. 06 Apr 2006. https://register.dpma.de/DPMAregister/pat/PatSchrifteneinsicht?docId=DE102004046526B4 (cited on pages 6–7, 9).

Klatzky, Roberta L. and Susan J. Lederman [2004]. *Handbook of Psychology: Experimental Psychology*. In: volume 4. John Wiley & Sons, 16 Apr 2004. Chapter 6, pages 147–176. ISBN 047166667X. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.128.3816&rep=rep1&type=pdf (cited on page 24).

Kleynhans, Stefanus Andreas and Ina Fourie [2014]. *Ensuring Accessibility of Electronic Information Resources for Visually Impaired People: the Need to Clarify Concepts such as Visually Impaired*. Library Hi Tech 32.2 (10 Jun 2014), pages 368–379. ISSN 0737-8831. doi:10.1108/LHT-11-2013-0148. https://pdfs.semanticscholar.org/06c3/a9a4baef9dc56c4d138b2b353b8cfb7018d1.pdf (cited on page 6).

Kopacz, Lindsey [2019]. *Accessibility in D3 Bar Charts*. Blog posting. 06 May 2019. https://a11ywithlindsey.com/blog/accessibility-d3-bar-charts (cited on pages 67–69, 92, 115).

Kopecek, Ivan and Radek Oslejsek [2008]. *GATE to Accessibility of Computer Graphics*. Computers Helping People with Special Needs: 11th International Conference (ICCHP 2008) (Linz, Austria). Edited by Klaus Miesenberger, Joachim Klaus, Wolfgang Zagler, and Arthur Karshmer. Volume 5105. Lecture Notes in Computer Science. Berlin / Heidelberg, Germany: Springer Verlag, 09 Jul 2008, pages 295–302. doi:10.1007/978-3-540-70540-6_44 (cited on pages 29, 47, 90).

Kopel, Christopher A., Keith Andrews, Inti Gabriel Mendoza Estrada, Lukas Bodner, Daniel Geiger, and Lorenz Leitner [2021a]. *AChart Interpreter*. 14 May 2021. https://github.com/tugraz-isds/achart-interpreter (cited on pages 3, 121, 153).

Kopel, Christopher A., Keith Andrews, Inti Gabriel Mendoza Estrada, Lukas Bodner, Daniel Geiger, and Lorenz Leitner [2021b]. *AChart Interpreter*. Online demo. 15 May 2021. https://tugraz-isds.github.io/achart-interpreter/ (cited on page 121).

Kopel, Christopher A., Keith Andrews, Inti Gabriel Mendoza Estrada, Alexander Grass, Lea Novak, and Danica Radulovic [2021]. *AChart Creator*. 14 May 2021. https://github.com/tugraz-isds/achart-creator (cited on pages 3, 97, 153).

Lang, Markus, Ursula Hofer, and Friederike Beyer [2010]. *Didaktik des Unterrichts mit blinden und hochgradig sehbehinderten Schülerinnen und Schülern*. Volume 2: Fachdidaktiken. Kohlhammer, 2010. 276 pages. ISBN 3170201514 (cited on pages 5–7, 9–11, 23–24).

MDN [2019]. *Web Speech API*. 18 Mar 2019. https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API (cited on pages 42, 131, 148).

MDN [2020]. *<menu> – HTML: Hypertext Markup Language*. 11 Apr 2020. https://developer.mozilla.org/en-US/docs/Web/HTML/Element/menu (cited on page 146).

Meeks, Elijah and Susie Lu [2020]. *Semiotic – Accessibility*. 25 Apr 2020. https://semiotic.nteract.io/guides/accessibility (cited on pages 34, 39, 75).

metec [2020]. *Der „Laptop für Blinde“*. 04 Aug 2020. https://metec-ag.de/graphik%20display.html (cited on page 27).

Microsoft [2020a]. *Complete Guide to Narrator – Windows Help*. 2020. https://support.microsoft.com/help/22798 (cited on pages 7, 10).

Microsoft [2020b]. *TypeScript: Typed JavaScript at Any Scale*. 2020. http://typescriptlang.org/ (cited on pages 3, 89).

Microsoft [2021]. *Seeing AI App*. 25 Apr 2021. https://microsoft.com/en-us/ai/seeing-ai (cited on page 21).

Migliorisi, Heather [2016]. *Accessible SVGs*. Blog posting. 28 Aug 2016. https://css-tricks.com/accessible-svgs (cited on pages 46, 49, 67–69, 92).

Migliorisi, Heather [2019]. *Accessible Complex Image – Bar Graph*. Source code. https://codepen.io/hmig/pen/MeJKee (cited on pages 67–69).

Mirri, Silvia, Silvio Peroni, Paola Salomoni, Fabio Vitali, and Vincenzo Rubano [2017]. *Towards Accessible Graphs in HTML-based Scientific Articles*. 2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC) (Las Vegas, NV, USA). IEEE, 08 Jan 2017, pages 1067–1072. doi:10.1109/CCNC.2017.7983287. https://essepuntato.it/papers/rash-ads2017.html (cited on page 42).

Moraes, Priscilla, Gabriel Sina, Kathleen McCoy, and Sandra Carberry [2014]. *Evaluating the Accessibility of Line Graphs through Textual Summaries for Visually Impaired Users*. Proc. 16th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS 2014) (Rochester, NY, USA). New York, NY, USA: ACM, 20 Oct 2014, pages 83–90. doi:10.1145/2661334.2661368 (cited on pages 21, 32, 154).

Morash, Valerie S., Yue-Ting Siu, Joshua A. Miele, Lucia Hasty, and Steven Landau [2015]. *Guiding Novice Web Workers in Making Image Descriptions Using Templates*. ACM Transactions on Accessible Computing 7.4 (Nov 2015), 12:1–12:21. ISSN 1936-7228. doi:10.1145/2764916. http://valeriemorash.com/publications/2015%20Morash%20-%20Guiding%20novice%20web%20workers%20in%20making%20image%20descriptions%20using%20templates.pdf (cited on page 21).

NCAM and DIAGRAM Center [2015]. *Image Description Guidelines*. Technical report. Jun 2015. http://diagramcenter.org/table-of-contents-2.html (cited on page 21).

NV Access [2020]. *NVDA 2020.2 User Guide*. 2020. https://nvaccess.org/files/nvda/documentation/userGuide.html (cited on pages 10–12, 91, 130–131).

OpenJS [2021a]. *Electron*. OpenJS Foundation, 08 Apr 2021. https://electronjs.org/ (cited on pages 3, 121).

OpenJS [2021b]. *Node.js*. OpenJS Foundation, 08 Apr 2021. https://nodejs.org/ (cited on pages 3, 115).

Orbit [2021]. *Graphiti – A Breakthrough in Non-Visual Access to All Forms of Graphical Information*. Orbit Research, 08 Apr 2021. https://orbitresearch.com/product/graphiti (cited on pages xiii, 27–28, 154).

Owen, Justin M., Julie A. Petro, Steve M. D'Souza Ravi Rastogi, and Dianne T. V. Pawluk [2009]. *An Improved, Low-Cost Tactile "Mouse" for Use by Individuals Who Are Blind and Visually Impaired*. Proc. 11th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS 2009) (Pittsburgh, Pennsylvania, USA). New York, NY, USA: ACM, 25 Oct 2009, pages 223–224. doi:10.1145/1639642.1639686. http://sonify.psych.gatech.edu/~walkerb/classes/assisttech/pdf/Owen(2009).pdf (cited on page 27).

Patil, Sandeep R. [2007]. *Position Paper: Accessible Image File Formats - The Need and the Way*. Proc. 2007 International Cross-Disciplinary Conference on Web Accessibility (W4A 2007) (Banff, Canada). New York, NY, USA: ACM, 07 May 2007, pages 40–43. doi:10.1145/1243441.1243455. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.118.3582&rep=rep1&type=pdf (cited on page 47).

Perko, Alexander [2021]. *AChart Interpreter Showcase Video*. Graz University of Technology, 29 Apr 2021. https://youtu.be/NLKqTTnKLII (cited on page 121).

Pickering, Heydon [2016]. *Inclusive Design Patterns*. Smashing, Sep 2016. ISBN 3945749433 (cited on pages 14–17).

Prescher, Denise, Jens Bornschein, Wiebke Köhlmann, and Gerhard Weber [2017]. *Touching Graphical Applications: Bimanual Tactile Interaction on the HyperBraille Pin-Matrix Display*. Universal Access in the Information Society 17 (05 Apr 2017), pages 391–409. doi:10.1007/s10209-017-0538-8 (cited on pages 24, 26).

Revnitski, Dmitri [2005]. *Konzeption und Realisierung eines taktilen Zugangs zu grafischen Veranschaulichungen mathematischer Sachverhalte. Entwurf und Implementierung von Darstellungshilfen für den Mathematikunterricht von blinden Schülern und Studenten*. Diploma Thesis. Universität Karlsruhe, Jan 2005 (cited on pages 23, 26, 31, 90).

Rotard, Martin, Kerstin Otte, and Thomas Ertl [2004]. *Exploring Scalable Vector Graphics for Visually Impaired Users*. Computers Helping People with Special Needs: 9th International Conference (ICCHP 2004) (Paris, France). Edited by Joachim Klaus, Klaus Miesenberger, Dominique Burger, and Wolfgang Zagler. Volume 3118. Lecture Notes in Computer Science. Berlin / Heidelberg, Germany: Springer-Verlag, 07 Jul 2004, pages 725–730. doi:10.1007/978-3-540-27817-7_108. `http://vis-web.informatik.uni-stuttgart.de/~rotard/publications/ICCHP04-SVG4VisuallyImpaired.pdf` (cited on pages 31, 90).

Salameh, Khouloud, Joe Tekli, and Richard Chbeir [2014]. *SVG-to-RDF Image Semantization*. Proc. 7th International Conference on Similarity Search and Applications (SISAP 2014) (Los Cabos, Mexico). Edited by Agma Juci Machado Traina, Caetano Traina Jr., and Robson Leonardo Ferreira Cordeiro. Volume 8821. Lecture Notes in Computer Science. Springer International Publishing Switzerland, 29 Oct 2014, pages 214–228. doi:10.1007/978-3-319-11988-5_20. `https://hal.archives-ouvertes.fr/hal-01082168/document` (cited on pages 46–47).

SAS [2019]. *SAS Graphics Accelerator*. 2019. `https://support.sas.com/software/products/graphics-accelerator` (cited on page 33).

Schepers, Doug [2015a]. *Describler*. 2015. `http://describler.com/` (cited on pages 3, 42, 54, 89, 91).

Schepers, Doug [2017]. *describler: SVG Dataviz Accessibility Tool*. 31 Mar 2017. `https://github.com/shepazu/describler` (cited on pages 42–43, 54, 91).

Schepers, Doug [2019]. *Accessible Charts with ARIA*. Blog posting. 25 Nov 2019. `https://blog.tenon.io/accessible-charts-with-aria` (cited on pages 46, 57, 92).

Schepers, Doug [2020]. *Why Accessibility Is at the Heart of Data Visualization*. Nightinggale. Article. 21 May 2020. `https://medium.com/nightingale/accessibility-is-at-the-heart-of-data-visualization-64a38d6c505b` (cited on page 23).

Schepers, Douglas [2015b]. *Accessible SVG Data Visualization*. 17 Feb 2015. `https://youtu.be/W1VUr544i84` (cited on page 42).

Sharif, Ather [2015a]. *evoGraphs*. Documentation. 2015. `http://evoxlabs.org/projects/evographs` (cited on pages 21, 39).

Sharif, Ather [2015b]. *evoGraphs – a jQuery Plugin to Create Web Accessible Graphs*. EvoXLabs, 23 Jan 2015. `https://youtu.be/bnACsbcbUxY` (cited on page 42).

Sharif, Ather and Babak Forouraghi [2018]. *evoGraphs – a jQuery Plugin to Create Web Accessible Graphs*. 2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC) (Las Vegas, NV, USA). IEEE, 12 Jan 2018, pages 1–4. doi:10.1109/CCNC.2018.8319239 (cited on page 39).

Shneiderman, Ben [1996]. *The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations*. Proc. 1996 IEEE Symposium on Visual Languages (VL '96) (Boulder, CO, USA). IEEE Computer Society, 03 Sep 1996, pages 336–343. doi:10.1109/VL.1996.545307. `https://cs.umd.edu/~ben/papers/Shneiderman1996eyes.pdf` (cited on page 22).

Sorge, Volker [2016]. *Polyfilling Accessible Chemistry Diagrams*. Computers Helping People with Special Needs: 15th International Conference (ICCHP 2016) (Linz, Austria, 13 Jul 2016). Edited by Klaus Miesenberger, Christian Bühler, and Petra Penaz. Volume 9758.I. Lecture Notes in Computer Science.

Springer International Publishing Switzerland, 06 Jul 2016, pages 43–50. doi:10.1007/978-3-319-41264-1_6. https://research.birmingham.ac.uk/portal/files/33124334/icchp16AA.pdf (cited on pages 32, 47, 49, 91).

Summers, Ed, Julianna Langston, and Dan Heath [2018]. *Accessibility and ODS Graphics: Seven Simple Steps to Section 508 Compliance Using SAS 9.4M5*. SAS Institute Inc. Jan 2018 (cited on page 33).

ViewPlus [2016a]. *VP Elite and Premier Embosser User Manual*. ViewPlus. 13 Jun 2016. https://viewplus.com/downloads/docs/manuals/VPElitePremier_UM_EN_20160613.pdf (cited on page 25).

ViewPlus [2016b]. *VP EmBraille User Manual*. ViewPlus. 13 Jun 2016. https://viewplus.com/downloads/docs/manuals/VPEmBraille_UM_EN_20160613.pdf (cited on page 25).

ViewPlus [2020]. *ViewPlus Software Suite 7.0.7 User Manual*. ViewPlus. 27 Apr 2020. https://viewplus.com/downloads/docs/manuals/VPTSS7_UM_EN_20200427.pdf (cited on page 25).

ViewPlus [2021]. *Braille Embossers & Tactile Graphics*. ViewPlus Technologies, 08 Apr 2021. https://viewplus.com/ (cited on pages 25, 30).

Vispero [2020]. *JAWS® – Freedom Scientific*. 2020. https://freedomscientific.com/products/software/jaws (cited on pages 10–12, 91, 130–131).

Völkel, Thorsten, Gerhard Weber, and Ulrich Baumann [2008]. *Tactile Graphics Revised: The Novel BrailleDis 9000 Pin-Matrix Device with Multitouch Input*. Computers Helping People with Special Needs: 11th International Conference (ICCHP 2008) (Linz, Austria). Edited by Klaus Miesenberger, Joachim Klaus, Wolfgang Zagler, and Arthur Karshmer. Volume 5105. Lecture Notes in Computer Science. Berlin / Heidelberg, Germany: Springer-Verlag, 09 Jul 2008, pages 835–842. doi:10.1007/978-3-540-70540-6_124 (cited on page 26).

W3C [2011]. *Scalable Vector Graphics (SVG) 1.1 (Second Edition)*. W3C Recommendation 1.1 (Second Edition). World Wide Web Consortium (W3C), 16 Aug 2011. https://w3.org/TR/2011/REC-SVG11-20110816 (cited on pages 1, 45–46).

W3C [2015a]. *HTML5 Image Description Extension (longdesc)*. W3C Recommendation. World Wide Web Consortium (W3C), 26 Feb 2015. https://w3.org/TR/2015/REC-html-longdesc-20150226 (cited on page 21).

W3C [2015b]. *SVG Accessibility/ARIA Roles for Charts. A Proposed System of Roles and Properties for Using ARIA to Annotate Charts, Graphs, and Maps*. W3C wiki. 11 Sep 2015. https://w3.org/wiki/SVG_Accessibility/ARIA_roles_for_charts (cited on pages 60–61, 92).

W3C [2017a]. *Accessible Rich Internet Applications (WAI-ARIA) 1.1*. W3C Recommendation 1.1. World Wide Web Consortium (W3C), 14 Dec 2017. https://w3.org/TR/2017/REC-wai-aria-1.1-20171214 (cited on pages 1, 16–17, 33, 48, 50, 64, 86, 92).

W3C [2017b]. *Core Accessibility API Mappings 1.1*. W3C Recommendation 1.1. World Wide Web Consortium (W3C), 14 Dec 2017. https://w3.org/TR/2017/REC-core-aam-1.1-20171214 (cited on pages 17, 48).

W3C [2017c]. *HTML 5.2*. W3C Recommendation 5.2. World Wide Web Consortium (W3C), 14 Dec 2017. https://w3.org/TR/2017/REC-html52-20171214 (cited on pages 14, 21).

W3C [2018a]. *Accessible Name and Description Computation 1.1*. W3C Recommendation 1.1. World Wide Web Consortium (W3C), 18 Dec 2018. https://w3.org/TR/2018/REC-accname-1.1-20181218 (cited on page 18).

W3C [2018b]. *Graphics Accessibility API Mappings*. W3C Recommendation 1.0. World Wide Web Consortium (W3C), 02 Oct 2018. https://w3.org/TR/2018/REC-graphics-aam-1.0-20181002 (cited on page 48).

W3C [2018c]. *Scalable Vector Graphics (SVG) 2*. W3C Candidate Recommendation 2. World Wide Web Consortium (W3C), 04 Oct 2018. `https://w3.org/TR/2018/CR-SVG2-20181004` (cited on pages 45, 48).

W3C [2018d]. *SVG Accessibility API Mappings*. W3C Working Draft 1.0. World Wide Web Consortium (W3C), 10 May 2018. `https://w3.org/TR/2018/WD-svg-aam-1.0-20180510` (cited on pages 48, 133).

W3C [2018e]. *WAI-ARIA Graphics Module*. W3C Recommendation 1.0. World Wide Web Consortium (W3C), 02 Oct 2018. `https://w3.org/TR/2018/REC-graphics-aria-1.0-20181002` (cited on pages 48, 57, 92).

W3C [2018f]. *Web Content Accessibility Guidelines (WCAG) 2.1*. W3C Recommendation 2.1. World Wide Web Consortium (W3C), 05 Jun 2018. `https://w3.org/TR/2018/REC-WCAG21-20180605` (cited on pages 6, 12, 21).

W3C [2020]. *Techniques for WCAG 2.1*. WCAG supporting document. World Wide Web Consortium (W3C), 10 Jul 2020. `https://w3.org/WAI/WCAG21/techniques` (cited on page 12).

W3C [2021a]. *W3C Accessibility Guidelines (WCAG) 3.0*. W3C First Public Working Draft 3.0. World Wide Web Consortium (W3C), 21 Jan 2021. `https://w3.org/TR/2021/WD-wcag-3.0-20210121` (cited on page 14).

W3C [2021b]. *Web Accessibility Initiative (WAI)*. 24 Apr 2021. `https://w3.org/WAI` (cited on pages 5, 16).

W3C [2021c]. *World Wide Web Consortium (W3C)*. 08 Apr 2021. `https://w3.org/` (cited on pages 1, 12).

Walker, Bruce and Joshua Cothran [2003]. *Sonification Sandbox: a Graphical Toolkit for Auditory Graphs*. Proc. 9th International Conference on Auditory Display (ICAD 2003) (Boston, MA, USA). 06 Jul 2003, pages 161–163. `http://cs.cmu.edu/~kkitani/pdf/YKKBS-AH11.pdf` (cited on page 29).

Wall, Steven A. and Stephen A. Brewster [2006]. *Tac-tiles: multimodal Pie Charts for Visually Impaired Users*. Proc. 4th Nordic Conference on Human- Computer Interaction: Changing Roles (NordiCHI 2006) (Oslo, Norway). 14 Oct 2006, pages 9–18. doi:10.1145/1182475.1182477. `http://eprints.gla.ac.uk/3236/1/tac-tiles1.pdf` (cited on page 30).

Watson, Léonie [2014]. *Tips for Creating Accessible SVG*. Blog posting. 06 May 2014. `https://sitepoint.com/tips-accessible-svg` (cited on page 49).

Watson, Léonie [2015]. *Accessibility APIs: A Key To Web Accessibility*. 16 Mar 2015. `https://smashingmagazine.com/2015/03/web-accessibility-with-accessibility-api/` (cited on page 17).

Watson, Léonie [2017]. *Accessible SVG Line Graphs*. Blog posting. 09 Sep 2017. `https://tink.uk/accessible-svg-line-graphs` (cited on pages 64–66, 92).

Watson, Léonie [2018]. *Test Case: SVG Accessible Names*. Blog posting. 16 Jan 2018. `https://test-cases.tink.uk/svg-accessible-names/index.html` (cited on page 49).

WebAIM [2019]. *Screen Reader User Survey #8 Results*. 27 Sep 2019. `https://webaim.org/projects/screenreadersurvey8` (cited on page 10).

Wefold, Harold E. [1976]. *Computer Graphics for the Blind*. ACM SIGCAPH Computers and the Physically Handicapped 18 (Jan 1976), pages 10–12. doi:10.1145/951781.951782 (cited on page 24).

Yoshida, Tsubasa, Kris M. Kitani, Hideki Koike, Serge Belongie, and Kevin Schlei [2011]. *EdgeSonic: Image Feature Sonification for the Visually Impaired*. Proc. 2nd Augmented Human International Conference (AH ’11) (Tokio, Japan). New York, NY, USA: ACM, 13 Mar 2011, 11:1–11:4. doi:10.1145/1959826.1959837. `http://sonify.psych.gatech.edu/publications/pdfs/2003ICAD-WalkerCothran-Sandbox.pdf` (cited on pages 23, 29).

Zhao, Haixia, Catherine Plaisant, Ben Shneiderman, and Jonathan Lazar [2008]. *Data Sonification for Users with Visual Impairment: a Case Study with Georeferenced Data*. ACM Transactions on Computer-Human Interaction 15.1 (May 2008), 4:1–4:28. ISSN 1073-0616. doi:10.1145/1352782.1352786. https://cs.umd.edu/~ben/papers/Zhao2008Data.pdf (cited on pages 23, 29).

Zou, Hong and Jutta Treviranus [2015]. *ChartMaster: a Tool for Interacting with Stock Market Charts Using a Screen Reader*. Proc. 17th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS 2015) (Lisbon, Portugal). New York, NY, USA: ACM, 26 Oct 2015, pages 107–116. doi:10.1145/2700648.2809862 (cited on pages 22, 42).