BrailleIO - a Tactile Display Abstraction Framework

Jens Bornschein
Human-Computer Interaction Research Group
Technische Universität Dresden
Nöthnitzer Str. 46, 01187 Dresden, Germany
jens.bornschein@tu-dresden.de

ABSTRACT

In this paper BrailleIO, a small .NET framework for developing two-dimensional tactile applications, is presented. It offers general features for displaying tactile information and for interaction. BrailleIO includes hardware abstraction, window and visualization features as well as basic interaction functions, such as panning and zooming on different content types. Information visualization can be organized in several independent screens that can be divided into multiple areas, having a full box model. Interaction can be realized via hardware keys of the used device or by basic gestures if the device is touch-sensitive.

Categories and Subject Descriptors

D.3.3 [**Programming Languages**]: Language Constructs and Features—*classes and objects, data types and structures, frameworks*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*haptic I/O, input devices and strategies, Screen design, User interface management systems (UIMS)*; K.4.2 [**Computers and Society**]: Social Issues—*assistive technologies for persons with disabilities*

General Terms

Documentation, Design

Keywords

Framework, tactile display, two-dimensional Braille display, pin-matrix device, hardware abstraction, standardization, tactile interaction, software development, tactile user interface

1. INTRODUCTION

Information technology is omnipresent in our daily life and we accept that because of so many benefits. With the increasing possibilities of information technology and the devices available everywhere getting more powerful, presentation of data can get even more user friendly. In our visual world this means that rich applications and big data get more and more visual to expose their secrets. This way, in addition to purely text-based systems interactive visual and graphical user interfaces pose a big benefit for the most users but leave also some behind.

TacTT '14, Nov 16 2014, Dresden, Germany.

Visually impaired users rely on a substituting alternative way to receive the information that is presented in a visual way. This mostly includes spatial information, too. While the visual world is entering the third dimension in information presentation now, visually impaired people are getting access to the second digital dimension with the help of the arisen smart mobile devices with touch screens. These are well usable tools for easy and self-determined interaction. But that is only audible in most cases. The step to a real tactile and, therefore, graphical interaction is still underrepresented.

Because of so much graphical information, the growing request for two-dimensional dynamic tactile displays for blind computer users results in an increasing number of available ideas and products. Several different designs and techniques for refreshable tactile displays exist.

Many prototypes for building tactile displays were developed, using different ideas to generate a dynamic tactile display [11, 12, 15]. Different approaches were used to generate tactile stimuli and arrange them in matrixes with more or less resolution. The techniques range from mechanical, electromagnetic, piezoelectric, or pneumatics, hydraulic or shape memory alloy based actuators up to surface changing polymers [8].

There are not only prototypes already available. Completely functional systems built for solving application-specific problems are available. The *Mimizu* system, using the *DotView-Pin-Display* [5], is a stylus-pen based drawing workstation. The pin display consists of 1,536 pins in a 48 x 32 pin-matrix with inter-pin space of about 3 mm and a display size of 144 by 96 mm. The application allows for painting and erasing freehand structures or the presentation of small images.

With the *GWP* system [1], a small pin-matrix display of 16 x 24 dots is used. The pin-matrix has a 3 mm pin raster, too. The device is portable and has 15 function keys, allowing for zooming, navigation and function calling. The *GWP* device is used to give access to mathematics, especially to the graphical part.

Within the *HyperBraille* project the touch-sensitive device called *BrailleDis* was developed. Two generations of pin-matrix devices exist, varying in body dimensions, amount of function keys and touch sensor resolution [13, 6]. The *BrailleDis* devices have a ten dpi dot matrix of 60 x 120 pins, resulting in a tactile area of 150 by 300 mm with 7,200 pins. The *HyperReader* software [10], using the *BrailleDis* devices for in- and output, was built to get access to desktop applications, such as office or web browsers on Windows systems. Thereby it offers different views on the content while keeping spatial layout information [9]. Within the *Hyper-Reader* a region concept for tactile user interfaces is used and evaluated as a well usable tool for structuring information [7]. The architecture of the *HyperReader* software is built to enable extensions and adaptations for different applications. The software framework is big and hard to learn. Furthermore, it is not free to use and cannot address other tactile displays for in- or output.

As mentioned before, those products are often delivered with their own proprietary software to visualize or retrieve information. Developing a new software product, addressing those output devices, is a time-consuming and expensive task. It could be hard for an application developer to convey information through the original software on the output device that is not related to the original purpose the system was built for.

In the following the framework *BrailleIO* is presented. Its goal is to lower the obstacles and simplify the start for developing applications for two-dimensional tactile displays by giving basic tools for tactile user interface design and hardware modeling.

2. THE FRAMEWORK

To reduce time and costs for developing a software application that uses a tactile display as output device, the framework *BrailleIO* is developed. It is a `.NET 4` based software framework written in `C#` and, therefore, only usable for Windows operating systems.

A main goal of *BrailleIO* is to give basic implementations for common needed functions and structures when building a tactile application. The barrier to enter should be as low as possible. The framework enables developers to achieve quick success without starting to solve fundamental problems in the first place. The framework should give possibilities for structuring information, presenting graphics and text - preferably as Braille - and enable interaction.

Successfully tested concepts, proven in previous projects [9, 7], are used to build tactile user interfaces and bring them to a wider field of users. The reuse of those concepts can help to improve quality of tactile applications and can start to bring consistency in structure as well as in look and feel of such applications. Consistency is important, especially for visually impaired users, because the user can rely on known structures and interaction paradigms [2]. This can improve learnability as well as efficiency and can reduce errors at the same time.

Therefore, it seems to be necessary to idealize and abstract the used hardware for input and output as well. This means, application developers can use basic features without any knowledge of the used device or get access to required information through the framework.

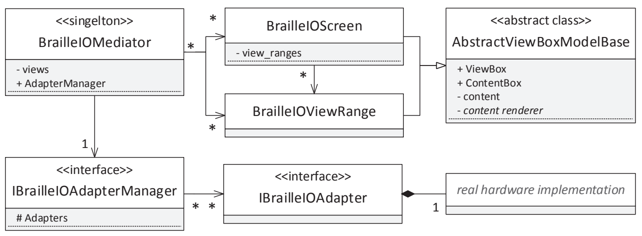

Figure 1. Basic structure of *BrailleIO*.

2.1 General Overview - Structure

The framework is divided into two main parts. The first part handles basic graphical elements and the generating of output. The second is responsible for managing, modeling and implementing the real used hardware devices. As shown in Figure 1, the two parts are linked together by a so-called mediator (`BrailleIOMediator`). The mediator handles the visibility and the rendering of the tactile user interface by managing different views which are building together the resulting tactile output. Every important part of the framework should be accessible through this component.

All modeled hardware devices - called adapters - have to be registered to the mediator. It distributes the generated tactile output to all registered and activated devices. The access to interaction events caused by user interaction with a device is not handled by the mediator. An interaction handling process has to be connected directly to every single registered adapter, which can be accessed through the mediator.

2.2 Tactile User Interface

The framework offers elements to build and structure tactile graphical user interfaces. The basic object for that is the so-called screen (`BrailleIOScreen`). As shown in Figure 1, a `BrailleIOMediator` can hold an unlimited set of screens. Normally, only one screen should be active at once and, therefore, will be rendered and sent to the displaying device. With this multiple screen metaphor it is possible to implement and provide several different views or applications simultaneously. The user can then switch easily between the different screens. A screen itself cannot have any content directly.

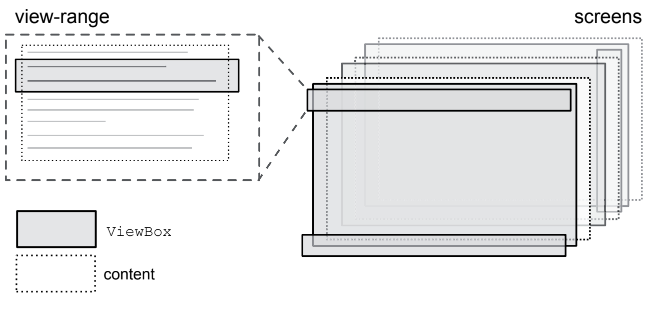

Figure 2. Screens and view-ranges.

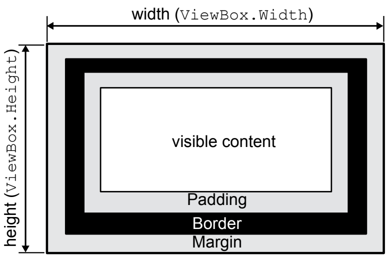

Figure 3. Box model of view-range and relation to *ViewBox*.

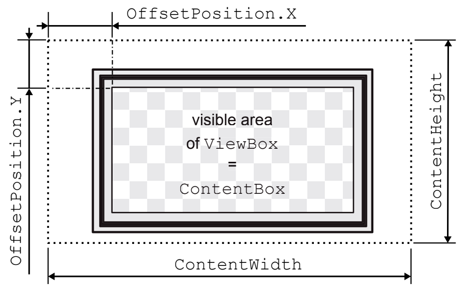

Figure 4. Relation between ViewBox, ContentBox and content of a view-range.

As the arrangement of a two-dimensional tactile output into several regions can support the bi-manual exploration of information [9], the framework should allow this feature, too. Therefore, screens can be divided into several content areas (see Figure 2). Those areas are called view-ranges (`BrailleIOViewRange`). A screen consists of an unlimited number of view-ranges (see Figure 1). Every view-range can be independently filled with content, placed on the screen, sized, activated or deactivated.

The view-ranges are stored at the corresponding screen in an internal list in the order they were added. This list represents a hierarchical, linear structure that has direct influence on the rendering process. The position inside the list corresponds to a common z-index mechanism. This means an overlapping view-range in a further position inside the list overwrites an overlapped view of an earlier position. In addition, the definition of a layer order, not corresponding to the list order, is also possible by setting the z-index property for the view-ranges. View-ranges are always opaque. That means the underlying content will be erased by an overlapping container, even if it is empty.

View-ranges have a full box model corresponding to the CSS box model for web sites [14] (see Figure 3). The box model consists of padding (space between content and border), a border and a margin (free space around the border to other elements). Free space is defined as lowered pins which results in a recognizable gap between elements. A part of a border is rendered as a continuous straight vertical or horizontal line of raised pins. Other border styles are not available until now. A border can be used to separate elements with a clear tactile stimuli. If a border is used, it is recommended to set a margin and a padding as well to mark the separating line as not related to the content. A border of one pin width seems to be sufficient in most cases. This also saves rarely available display space. All properties of the box model are independently definable in all four directions.

With these tools a screen can be organized in regions with different content which can be separated by significant tactile features, such as dividing space, separating lines or marking frames. The division of the available display space seems not always to be reasonable, especially on small displays such as the *GWP* [1]. However, for displaying special, temporary or non-persistent information the method of showing brief overlapping view-ranges could be useful. To ensure that contents of different areas are not mixed together by recognizing them as one whole content area it seems to be necessary to mark them as divided. Therefore, the box model can be used even if only one of its parameters is applied to create a border or a gap.

A view-range is mainly characterized by the size and the position on a screen. These parameters are stored in the `ViewBox` member variable. In the `ViewBox`, the outer dimension of the view-range is defined as well as the x and y position in relation to the top left corner of the corresponding screen. The remaining visible area of a view-range is called `ContentBox`, which has the dimension of the `ViewBox` subtracting the several box model sections (see Figure 4). All dimensions and positions used as parameters in this framework are defined in pins. Therefore, the definition of sizes and points is depending on the resolution of the used output device.

For handling oversized content and enabling the access to content that doesn't fit in a view-range, a panning concept is realized. With the parameter `OffsetPosition` the content can be moved under the view range frame. The offset position defines the start position of the content in relation to the left corner of the `ContentBox`. If necessary, simple tactile scrollbars are added and rendered to give feedback to the user about the position inside the content. The scrollbars consist of a continuous line with an adjacent indicator which is three points long and one point thick. To keep the scrollbar recognizable a one pixel space in the direction of the content is set up. Therefore, the scrollbars only reduce the available content space by three pins.

A view-range can get several types of content to display. As shown in Figure 5 the basic data structure, to which every other content type will be transformed, is a two-dimensional Boolean matrix (`bool[,]`). A true-value represents a raised pin on the tactile display, a false-value a lowered one. Other content types are text, which is rendered in Braille or as an image, as well as pictures, which are rendered as binary images. With a free definable threshold for the lightness of a pixel, darker pixels will be set to raised pins and lighter ones to lowered pins.

In the end the framework allows for setting any other type of content. However, this requires the definition of a specialized renderer for the given content type, implementing the `IBrailleIOContentRenderer` interface, converting the content to a usable Boolean matrix.

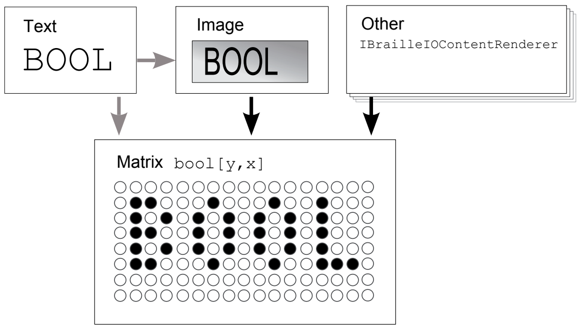

Figure 5. Content types for view-ranges.

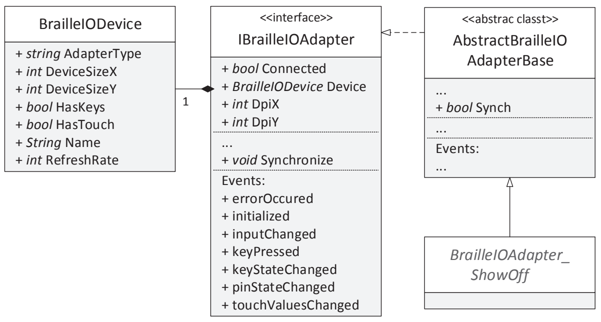

Figure 6. Adapter and device class structure.

Nearly all renderer are hookable. This means, an extension or an user of the framework can get access to the renderer functionality by registering a hook. This hook will be called before the renderer starts his work and after the renderer has rendered the result. A hook has the opportunity to manipulate all function parameters in the beginning of the rendering. Furthermore, the redering result can be modified before it will be returned and sent to the output devices. This gives programmers the power to use already implemented standard renderer and adapt or extent them to their needs, without implementing an own renderer.

2.3 Device Abstraction

As mentioned before, the abstraction and modeling of hardware is an essential part of the framework, too. Therefore, the first idea was to identify basic properties and features. A construct for modeling and mapping a real hardware interface was developed. In this, special properties are defined, for instance, a proposed image refresh rate, the number of pin-rows and columns or the availability of buttons or touch-sensitivity of the hardware device.

A specific hardware device still needs to be mapped to the proposed framework. For this purpose, an adapter has to be provided that implements the `IBrailleIOAdapter` interface and generates a unique device representation as an object of type `BrailleIODevice` (see Figure 6). This has to be done once for every new hardware device that should be used with the *BrailleIO* framework.

The implemented adapter is responsible to achieve proper access to the hardware device. This means the adapter has the task to create, open and hold a channel to the hardware device as long as needed and to bring the standardized output-matrix on the device. At the same time, the adapter implementation has to map the proprietary and device-specific interactions and events to the idealized data models which are expected by the framework. If a device has hardware keys, they have to be modeled, too. Nine basic function keys are defined as a minimal set for a sufficient interaction on a touch-sensitive tactile graphic device (see Figure 7). The key set consists of four navigation keys and two zoom buttons to interact with oversized, graphical or zoomable content, such as images. Furthermore, two interaction buttons for approval and refusal are defined. Finally, a special key for touch-sensitive devices seems to be necessary to start and stop a gesture input to avoid midas touch effects [10].

Some general device events are proposed as well (see Figure 6). These events should give feedback about availability and changes in key-, touch- or pin-states as well as the occurred errors. In these events the sending device and the original raw device data of the event are enclosed and sent to all registered listeners. The listener can decide if he wants to handle the generalized data, such as the general buttons, or the original device data, e.g. additional keys that are not mapped to one of the general keys.

Two real hardware-specific adapter implementations were built for different types of the *BrailleDis* series, named the *BrailleDis 9000* [13] and the *BrailleDis 7200* [6] (see Figure 8). In addition to the real adapter implementations, a software adapter was developed. The so-called ShowOff adapter can be used for debugging if no real hardware device is available or for monitoring a connected *BrailleDis* device. It implements the `IBrailleIOAdapter` interface and can be used as a standalone input or output device for applications based on the framework. It is inspired by the *BrailleDis 7200* device and can be used as emulator. With the simulator it is also possible to enter single touch inputs by mouse.

2.4 Interaction

A set of basic functions for interaction are implemented. As mentioned before, a first assignment of keys with functions is proposed by the adapter implementation (see section 2.3), but not realized by the framework. The linking between buttons and functions has to be done by the application developer. Nevertheless, all necessary functions are available in a basic implementation.

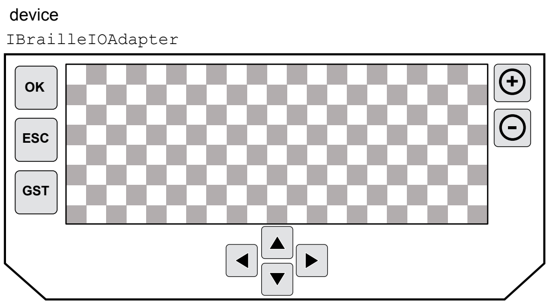

Figure 7. General device model with nine keys.

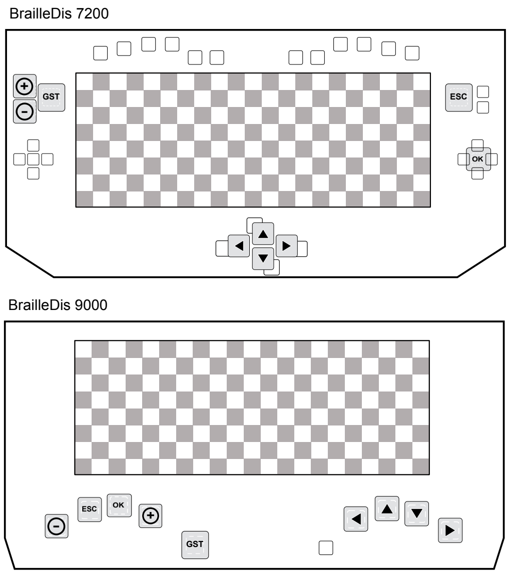

Figure 8. General key mapping for BrailleDis 7200 and BrailleDis 9000.

Zooming can be realized by setting the zoom property of the corresponding view-range. The framework will handle the zoomed rendering if possible. Panning operations can be connected to the `OffsetPosition` property of a view-range (see section 2.2). By setting the offsets to negative values the content can be moved below the visible area of the view-range's `ViewBox` (see Figure 4). Changing the y-offset realizes a vertical and changing the x-value a horizontal scrolling. The offset can be changed freely. Several functions of the abstract base class implementation of the view-ranges (see Figure 1) offer offset manipulation in an easy manner.

A basic gesture recognizer is included. It recognizes a number of basic gestures, such as pointing gestures (tab), swipes (line), pinch, circle (half and full) and three finger drag operations (compare [7]). All gestures are interpreted and returned with further information, such as start and end point, direction or orientation. In addition, a timestamp fingerprint allows for the inference on the duration, speed or temporary order of interactions.

Inversion and threshold adaption are also part of the framework for image handling. For instance, these features are useful for exploring images. Small sinks - regions of lowered pins inside an area of raised pins - can be transformed to a raised pin area for a better detection by inverting the presentation. The adaptation of the threshold level for the binary image conversion allows for adjusting the presentation to the given context. Especially if very light or very dark images are presented, the adjustment of the threshold is necessary to make structures visible at all.

2.5 Usage

In the following a small guidance about how to set up a project with the *BrailleIO* framework in a few steps is given (compare Figure 9).

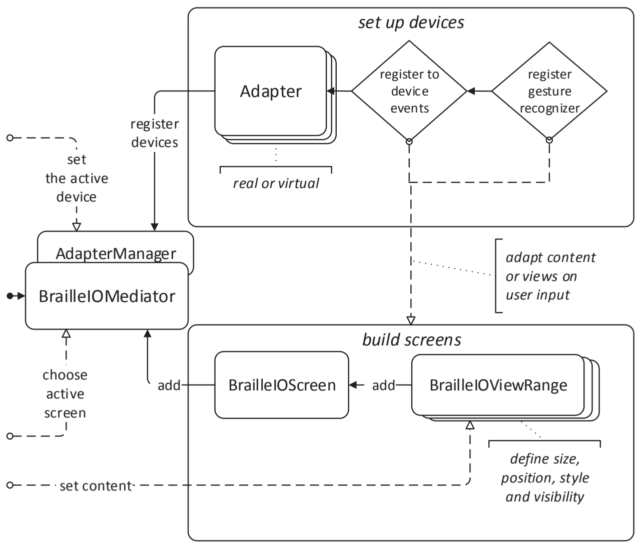

Figure 9. Steps to set up a project with *BrailleIO*.

After setting up the device that should be used for in- and output, it has to be registered to the `AdapterManager` related to the `BrailleIOMediator`. A registration to the events thrown by the device has to be done. If the device is touch sensitive, the optional basic gesture recognizer can be connected to interpret touch inputs. An unlimited number of devices can be registered but only one device can be marked as active. This device is used as main output. Other devices can be used as output by setting the `Synch` property of the `AbstractBrailleIOAdapterBase`. This leads the mediator to mirror the tactile result to these devices. With this mechanism the ShowOff adapter can be used, for example, as debug monitor beside a real tactile matrix device.

At least one view-range has to be defined for displaying content. It has to be configured with a position, size and the optional box model. The view-range can be added to the mediator directly or it can be combined with other view-ranges in a screen which has to be added to the mediator. An unlimited amount of screens are allowed. After setting a screen as active or visible it will be displayed on the output device. Every view-range has to get its own content which will be rendered and presented. The views or the contents can be changed, for example, on user interactions reported from the device events.

3. CONCLUSION AND OUTLOOK

The framework *BrailleIO* was presented. It enables a fast and easy entry into building applications on two-dimensional tactile displays for visually impaired users. With the screen and view-range constructs a proper information organization and simultaneous reception is possible. The framework also proposes a hardware abstraction for pin-matrix devices including general hardware keys and function binding.

The framework is used, for example, as groundwork for a tactile graphic production workstation called *Tangram workstation* [3]. This project enables collaborative work of a sighted and a blind user on one graphic.

Several open issues have to be solved and innumerable improvements are conceivable. At this point, there is no way to map infinite further hardware keys to the generic key construct. A continuous numbering of further keys could be a solution to overcome the fall back to the proprietary naming of these buttons. In this context, the implementation of more concrete hardware adapters would proof the concepts and the portability beyond the *BrailleDis* devices.

It is also unclear how to handle output devices with a significantly higher resolution than 10 dpi, which will lead to problems on rendering Braille. The resolution of an used device is available in his specifications and therefore has to be checked and used while rendering resolution dependent content.

The next big and challenging step is the implementation of a powerful Braille renderer. Currently, strings are rendered with an equidistant Braille font as an image and sent to the output devices. There is no way back from the rendered image to the original given text. This is necessary, for example, for a controlled audio output of touched Braille elements. For the near future, it is planned to build a renderer based on the free transcoding project *liblouis* [4]. This should allow to take HTML strings as input which can be adapted with cascading style sheets.

BrailleIO is realized as open source project. One big advantage is, if some functions or concepts are missing or are not optimal, the open source approach enables anybody to take part and improve, change or complete the project with further and better ideas. The framework is freely available through https://github.com/TUD-INF-IAI-MCI/BrailleIO.

4. ACKNOWLEDGMENTS

I thank Heinrich Kuhlmann for his work on *BrailleIO*, Dr. rer. nat. Michael Schmidt for his implementation of a basic gesture recognizer and Denise Prescher for her work on the interaction design. The *Tangram* project, in which *BrailleIO* is developed, is sponsored by the Federal Ministry of Labour and Social Affairs (BMAS) under the grant number R/FO125423.

5. REFERENCES

[1] P. Albert. Math class: An application for dynamic tactile graphics. In K. Miesenberger, J. Klaus, W. Zagler, and A. Karshmer, editors, *Computers Helping People with Special Needs*, volume 4061 of *Lecture Notes in Computer Science*, pages 1118–1121. Springer Berlin Heidelberg, 2006.

[2] R. Babu, R. Singh, and J. Ganesh. Understanding blind users' web accessibility and usability problems. *AIS Transactions on Human-Computer Interaction*, 2(3):73–94, 2010.

[3] J. Bornschein and D. Prescher. Collaborative tactile graphic workstation for touch-sensitive pin-matrix devices. In *This Proceedings*, pages 42 – 47, 2014.

[4] C. Egli. *Liblouis-a universal solution for Braille transcription services.* Deutsche Zentralbibliothek für Blinde Leipzig (DZB), 2011.

[5] M. Kobayashi and T. Watanabe. Communication system for the blind using tactile displays and ultrasonic pens - mimizu. In K. Miesenberger, J. Klaus, W. Zagler, and D. Burger, editors, *Computers Helping People with Special Needs*, volume 3118 of *Lecture Notes in Computer Science*, pages 731–738. Springer Berlin Heidelberg, 2004.

[6] D. Prescher. Redesigning input controls of a touch-sensitive pin-matrix device. In *This Proceedings*, pages 19 – 24, 2014.

[7] D. Prescher, G. Weber, and M. Spindler. A tactile windowing system for blind users. In *Proceedings of the 12th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '10, pages 91–98, New York, NY, USA, 2010. ACM.

[8] A. Richter and G. Paschew. Optoelectrothermic control of highly integrated polymer-based mems applied in an artificial skin. *Advanced Materials*, 21(9):979–983, 2009.

[9] M. Schiewe, W. Köhlmann, O. Nadig, and G. Weber. What you feel is what you get: Mapping guis on planar tactile displays. In C. Stephanidis, editor, *Universal Access in Human-Computer Interaction. Intelligent and Ubiquitous Interaction Environments*, volume 5615 of *Lecture Notes in Computer Science*, pages 564–573. Springer Berlin Heidelberg, 2009.

[10] M. Spindler, M. Kraus, and G. Weber. A graphical tactile screen-explorer. In K. Miesenberger, J. Klaus, W. Zagler, and A. Karshmer, editors, *Computers Helping People with Special Needs*, volume 6180 of *Lecture Notes in Computer Science*, pages 474–481. Springer Berlin Heidelberg, 2010.

[11] R. Velazquez, E. Pissaloux, M. Hafez, and J. Szewczyk. A low-cost highly-portable tactile display based on shape memory alloy micro-actuators. In *Virtual Environments, Human-Computer Interfaces and Measurement Systems, 2005. VECIMS 2005. Proceedings of the 2005 IEEE International Conference on*, pages 6 pp.–, July 2005.

[12] F. Vidal-Verdu and M. Hafez. Graphical tactile displays for visually-impaired people. *Neural Systems and Rehabilitation Engineering, IEEE Transactions on*, 15(1):119–130, March 2007.

[13] T. Völkel, G. Weber, and U. Baumann. Tactile graphics revised: The novel brailledis 9000 pin-matrix device with multitouch input. In K. Miesenberger, J. Klaus, W. Zagler, and A. Karshmer, editors, *Computers Helping People with Special Needs*, volume 5105 of *Lecture Notes in Computer Science*, pages 835–842. Springer Berlin Heidelberg, 2008.

[14] W3C. World wide web consortium - cascading style sheets level 2 revision 1 (css 2.1) specification - 8 box model. http://www.w3.org/TR/CSS2/box.html, 2011.

[15] E. Wilhelm, T. Schwarz, G. Jaworek, A. Voigt, and B. Rapp. Towards displaying graphics on a cheap, large-scale braille display. In K. Miesenberger, D. Fels, D. Archambault, P. Penaz, and W. Zagler, editors, *Computers Helping People with Special Needs*, volume 8547 of *Lecture Notes in Computer Science*, pages 662–669. Springer International Publishing, 2014.