IMAGE: An Open-Source, Extensible Framework for Deploying Accessible Audio and Haptic Renderings of Web Graphics

JULIETTE REGIMBAL, JEFFREY R. BLUM, CYAN KUO, and JEREMY R. COOPERSTOCK, McGill University, Canada

For accessibility practitioners, creating and deploying novel multimedia interactions for people with disabilities is a nontrivial task. As a result, many projects aiming to support such accessibility needs come and go, or never make it to a public release. To reduce the overhead involved in deploying and maintaining a system that transforms web content into multimodal renderings, we created an open-source, modular microservices architecture as part of the IMAGE project. This project aims to design richer means of interacting with web graphics than is afforded by a screen reader and text descriptions alone. To benefit the community of accessibility software developers, we discuss this architecture and explain how it provides support for several multimodal processing pipelines. Beyond illustrating the initial use case that motivated this effort, we further describe two use cases outside the scope of our project in order to explain how a team could use the architecture to develop and deploy accessible solutions for their own work. We then discuss our team's experience working with the IMAGE architecture, informed by discussions with six project members, and provide recommendations to other practitioners considering applying the framework to their own accessibility projects.

CCS Concepts: • **Human-centered computing** → **Accessibility systems and tools**; *Interactive systems and tools*; • **Software and its engineering** → *Software libraries and repositories.*

Additional Key Words and Phrases: Systems and architecture, Web accessibility, Multimodal interaction, Blind, Low vision

ACM Reference Format:
Juliette Regimbal, Jeffrey R. Blum, Cyan Kuo, and Jeremy R. Cooperstock. 2024. IMAGE: An Open-Source, Extensible Framework for Deploying Accessible Audio and Haptic Renderings of Web Graphics. *ACM Trans. Access. Comput.* 1, 1 (May 2024), 17 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Numerous research efforts have focused on automatically generating representations of visual media for blind and low vision (BLV) individuals. These were in part motivated by widespread accessibility issues, including the absence of image descriptions on much of the web, and the poor quality of descriptions written by social media users who often do not know why or how they should describe their own content [4, 13, 24, 28]. Due to the variety of purposes served by graphics and the differences across BLV groups, e.g., between early- and late-blind people [26], effective accessible representations can vary widely between people, and therefore must be tailored appropriately.

Authors' address: Juliette Regimbal, juliette.regimbal@mail.mcgill.ca; Jeffrey R. Blum, jeffrey.blum@mail.mcgill.ca; Cyan Kuo, cyk@cim.mcgill.ca; Jeremy R. Cooperstock, jer@cim.mcgill.ca, Department of Electrical and Computer Engineering, McGill University, 3480 rue University, Montréal, Québec, H3A 0E9, Canada.

1936-7228/2024/5-ART $15.00
https://doi.org/XXXXXXX.XXXXXXX

While previous work has emphasized improving the quality of image descriptions [16, 24, 30, 31], other approaches leverage non-textual and non-visual modalities [14, 15, 18, 19, 29, 32].

New challenges emerge in cases when the goals of a project include deployment to the public. Since users of these technologies will ultimately need multiple methods of representing graphics—one would expect a different experience for a map than for a photograph of friends—small single-feature projects may frustrate users if each one needs to be installed and managed separately. For the researchers, designers, and developers, projects developed from the ground-up would need to reimplement common functionality rather than reuse the same modules. This would also increase the difficulty of integrating improvements made in related fields, such as those in computer vision and large language models.

Unfortunately, since such accessibility projects are typically developed as independent, self-contained systems, the aforementioned challenges may be difficult to avoid, resulting in increased technical effort required for practitioners to deploy their work to the public or even to build upon the efforts of their peers. Even assuming that any given projects are not reduced in scope as a result of these costs, resources that could be dedicated to improving the experience of BLV users must be spent ensuring that the underlying technical infrastructure functions correctly. Some research groups may not pursue deployment if these added costs are considered likely to stretch personnel too thin to meet project deadlines on time. Instead, using a common software framework across projects to have components interact in a predictable and modular manner would allow for practitioners to work more efficiently by avoiding redesign and reimplementation efforts [21]. For some accessibility practitioners, enabling such reuse of infrastructure could make deployment a feasible goal.

We describe here the microservices-based architecture of our Internet Multimodal Access to Graphical Exploration (IMAGE) project, which produces accessible audio and haptic representations of web graphics. Since IMAGE focuses on several types of graphics, we faced these issues of reuse early in our project. Finding no framework focused on creating accessible representations of web graphics, we designed our own software architecture to be extensible so that other accessibility practitioners can build upon it, and have made it freely available to the wider community.

This article is an expanded version of our original communication in the 19th International Web for All Conference, which introduced the IMAGE architecture and provided an early look at its development [27]. Since then, the architecture has been refined through our own deployment experience. We evaluate whether it meets the base criteria to serve as a framework that promotes flexibility and deployability in accessibility projects such as IMAGE. First, we present a technical overview of the architecture to establish a basic understanding of how it functions in its current form. Second, influenced by evaluation methods of toolkits in HCI research [22], we describe scenarios encountered during the IMAGE project and hypothetical uses of the architecture in other contexts in order to demonstrate how it meets these requirements for a framework. Finally, we describe the results of interviews conducted with members of the IMAGE team who worked as developers, designers, and researchers, but were not directly involved in the design of the architecture. These interviews are intended to identify problems that may emerge over long term use and to understand how the framework may be adopted by practitioners in real, evolving projects. The factors found through these conversations are likely to be relevant to other groups using the architecture or developing their own equivalent. The main contribution of this paper is the updated description of the software framework and the illustrative scenarios and team member evaluations, which are intended to facilitate use of the framework.

Our intent is that the modularity of this framework, when adopted by other accessibility researchers, will support them in focusing more on the actual tasks that improve quality of representations of graphics, and less on the underlying effort of integrating features together. This

benefit should also be experienced as projects change in focus and scope, reducing the amount of reimplementation necessary. We hope that these features will enable practitioners to pursue goals that would otherwise be left to future work, in particular the goal of deploying their projects to members of the public for use in their daily lives.

2 BACKGROUND

2.1 Graphic Accessibility

Past projects have been concerned with the problem of producing accessible representations of digital graphics. Several from recent years are summarized here as an overview for the reader, starting first with those focused on the web. Wu et al. modified the Facebook application for iOS to produce automatic alt text for the Facebook News Feed. This system uses a machine learning model to assign tags to a photo, focusing on the most frequently used tags used by human annotators. Certain concepts were filtered out either due to the difficulty of implementation, such as landmark recognition, or due to a strong dependence on context, such as gender recognition and adjective assignment. The remaining tags were then used to construct complete sentences beginning with "Image may contain:" to provide a better user experience and emphasize the uncertainty in the system's outputs [31].

Similarly, Low et al. designed the browser extension Twitter A11y to generate alt text for Twitter. As a user scrolls through the site, the URLs of images in tweets are sent to Twitter A11y's server. If the image is part of a link preview, the linked page is searched for alt text corresponding to the preview image. If it is not part of a preview and contains text, optical character recognition (OCR) is used. For all remaining images where URL following and OCR approaches are inapplicable or fail to produce a result, a remote worker hired through Amazon Mechanical Turk writes an image description. The alt text is then inserted into the user's web page using the browser extension [24].

Caption Crawler is a system that consists of a browser extension and backend server. Images in a web page and associated accessibility metadata are sent from the client to the server. For images without an associated caption, the server first checks whether a captioned version of the same image had been encountered in previous requests and stored. If not, the server then queries for web pages that also display the image and stores the associated caption, if one is present. The server responds with the captions it has that are related to the images in the request. The browser extension dynamically inserts the captions into the page [16].

The Susurrus web application produces sonifications of bar chart, line chart, and scatter plot data loaded from CSV files. Natural sounds, such as from animals or weather, are used to represent data and were found to work best in communicating categorical data. While currently a standalone prototype, Hoque et al. suggest that it could be modified to function as a browser extension that retrieves chart data embedded in a web page [19].

While not implemented, Winters et al. proposed a system for creating auditory representations of social media posts with graphical content. Unlike the approaches mentioned previously, this one translates aspects of a post beyond an embedded photo, including the username, text, image content, and engagement metrics. These features are presented using a mix of speech, auditory icons, sonification, soundscapes, and music. High-level information not explicitly encoded in a post, such as gender, emotion, and text sentiment, are proposed to be extracted using various artificial intelligence services offered by Microsoft. The auditory components would be combined in a specific order with some parts overlapping. For example, speech conveying the text of a post would be accompanied by a soundscape representing the environment shown in an attached image [30].

The Picture Smart service, available as part of the JAWS screen reader, allows users to obtain a description of a submitted image. Images on a web page, in an email, or saved locally can be

selected within the service. Images are then sent to various other services, including Microsoft Azure and Google Cloud Platform, to perform computer vision tasks such as object recognition and OCR. The results of these services are combined and read using JAWS [1].

Apple's VoiceOver on iOS includes an Image Explorer feature to supplement image descriptions. Information on objects, text, and tables can be accessed within an image either by following a preset ordering or by swiping one's finger on the screen. When the finger is above an element with more information available, that information will be spoken by VoiceOver [3].

Outside of the web, but using similar methods to make visual media accessible, are projects such as VizWiz and Seeing AI. VizWiz was created to answer various "visual search problems", such as reading labels and finding a button with a certain function. A blind person could take a picture of a scene using a smartphone, record a question related to the scene, and send the photo and associated question to remote workers on Amazon Mechanical Turk. These workers then answer the question or explain why an answer cannot be provided [11]. A related app, VizWiz::LocateIt, helps blind users find a nearby object. A picture of a scene and query specifying the desired object are sent to a remote worker who then outlines where the object is. An automated system in the app then provides sonification to help the user move towards the object and correctly identify it [12].

Seeing AI is an iOS application created by Microsoft that uses the phone's camera and several computer vision processes to perform a large set of tasks. Supported functions include OCR, gender, age, and emotion recognition for people, scene description, and currency recognition. Spoken guidance is also provided to help users properly frame photos before a description is generated [7].

None of these projects, or any others to the authors' knowledge, were developed in what could be a widely used framework for increasing web graphic accessibility, let alone web media accessibility more broadly. Many of those mentioned were not made available to the public, and those that were are proprietary systems not able to be extended by another group. This is not due to an inherent inability for accessibility software to be open source, of course. To illustrate the benefits of open-source accessibility projects, we discuss examples outside the narrow scope of web graphic accessibility.

2.2 Open-Source and Extensible Accessibility Software

There are several important open-source screen readers that serve as extensible platforms to varying degrees. The most popular is NVDA, a screen reader for the Windows operating system developed by NV Access and a community of contributors. It also supports add-ons, both open-source and proprietary, that provide new or different functionality to users of the screen reader [5]. With mobile devices, the TalkBack screen reader for Android is available under an open-source license, although it lacks the add-on functionality and community contribution of NVDA. While these reflect the most popular open-source options—NVDA is commonly used by 58.8% of screen reader users on desktops and laptops, and TalkBack by 29.1% on mobile devices [2]— less popular open-source projects are also available, such as Orca [6] and WebAnywhere [10]. Outside of screen readers, and relevant to the focus of IMAGE, there have been accessibility-related software frameworks developed to tackle various problems. For example, the ITHACA framework was developed to aid in creating personalized Augmentative and Alternative Communications products [25]. The ACCESS framework was designed to support accessibility developers in using operating system-level accessibility features by providing a simpler interface [17].

The projects and frameworks described here have experienced success to varying degrees through positive evaluations in research contexts or adoption by members of the public. The frameworks and the add-on system for NVDA focus on providing a base on which to develop that is intended to be easy to learn and use. While elements of them could be applied to aspects of the projects

discussed in Section 2.1, important components and functions would still need to be created from scratch. A different framework is necessary for this kind of work.

2.3 Requirements for Web Media Accessibility Frameworks

A software framework for the IMAGE project or web media accessibility more broadly must support the reuse of components through problem-appropriate interfaces. The IMAGE project and those discussed in Section 2.1 all represent visual information using other senses, usually hearing, which is similar to the approach taken in sensory substitution research. Lenay et al. described sensory substitution systems as consisting of three components: a sensor for the input stimuli, a stimulator for the output stimuli, and a coupling system that links them [23]. This paradigm can be extended more broadly to encompass all the projects discussed above. Each project has a means of acquiring the data necessary to accomplish its task. For example, VizWiz and Seeing AI use cameras to capture input stimuli as is often done in sensory substitution. Other cases, such as Caption Crawler and Twitter A11y, collect data from the web rather than from a direct sensor of the environment. Each project displays the collected information through various "stimulators", for example speech, sonification, and modifications to the elements of an existing web page. The coupling system between these input and output stimuli are defined by algorithms appropriate to each project. These include machine learning methods to extract semantic information from the input (Facebook News Feed, Twitter A11y, Winters et al.'s system, Seeing AI, Picture Smart, and VoiceOver), queries to relevant web resources (Twitter A11y, Caption Crawler, and Hoque et al.'s proposed extension), and providing a task to a "human in the loop" (VizWiz and Twitter A11y). These steps of data collection, data processing, and output synthesis appear to be consistently present in this class of assistive technologies, and thus are set as requirements for the IMAGE architecture.

Many of these systems are not fully self-contained and rely on a server running separately from the software with which the user interacts, such as smartphones and web browsers [3, 7, 11, 16, 24]. These platforms that typically have less computing power or resources available than native applications on a personal computer, constraints that become especially relevant when large machine learning models are used. Additionally, systems such as Caption Crawler build upon requests from multiple users and require a means of making this information available to everyone using the product [16]. This observation motivates support for computation off of user devices as an important additional desideratum for our reusable software framework.

Along with these technical requirements, it is also necessary to consider how a system is used by teams as part of their own design process. Such accessibility tool teams must typically consider:

- design tasks, focused on creating new methods of interacting with web media for accessibility purposes;
- research tasks, focused on understanding the needs of prospective users and meaningfully evaluating elements of the system; and
- development tasks, focused on quickly implementing functionality required for the previous two types of tasks, and integrating improvements made in other fields, such as computer vision.

Depending on team organization, these tasks may be allocated to the same or different individuals. The background of each accessibility practitioner will influence how they prefer to work and the technologies with which they are proficient. Differing backgrounds must be supported in the design of a web graphic framework. Microservices are small, independent modules that operate as a system through lightweight protocols, such as HTTP [20]. For the sort of projects discussed here, a microservices-based approach offers productivity advantages that support the development challenges and diversity of tasks on which individual team members may need to work. Specifically,

microservices allow flexibility in choice of programming language, libraries, and other aspects of the implementation strategy [8] and reduce the technical skills required by each practitioner to contribute to the overall system.

3 THE IMAGE ARCHITECTURE

The IMAGE architecture benefited from adoption of such a strategy, which allows designers and developers to make significant changes to the system's behaviour, without needing to modify the system as a whole. As a concrete example, our development team regularly updated code to leverage advances in machine learning (ML) tools to produce improved audio-haptic representations. Because our architecture was based on microservices, the meaning-extraction elements could be updated independently, thereby allowing us to take advantage of such ML tools without needing to touch a line of code responsible for generating the end-user experiences. In the remainder of this section, we describe the design and arrangement of these components, and the customization of renderings they support, as appropriate to individual circumstances.

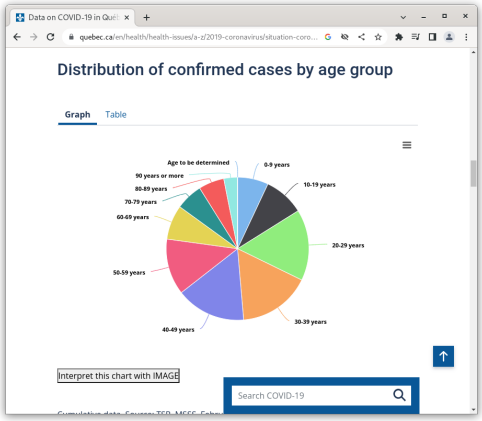
To facilitate an understanding of the IMAGE architecture, we first provide a basic description of the end-user experience that the system presently enables. Use of the IMAGE functionality merely requires installation of a free browser extension, which can be activated for photographs, embedded Google maps, and Highcharts[1] charts, either by alt-click or selecting buttons inserted into the HTML.[2] Examples of the UI elements in the extension are shown in Figure 1. During this process, users continue to be able to access any existing assistive technologies or accessible representations of the graphical content, e.g., alt text via a screen reader. This is critical, since IMAGE does not seek to replace any existing accessibility tools on which the user might already rely. Conversely, the browser extension is designed to work with popular screen readers with which users are likely to be familiar.

3.1 Overview

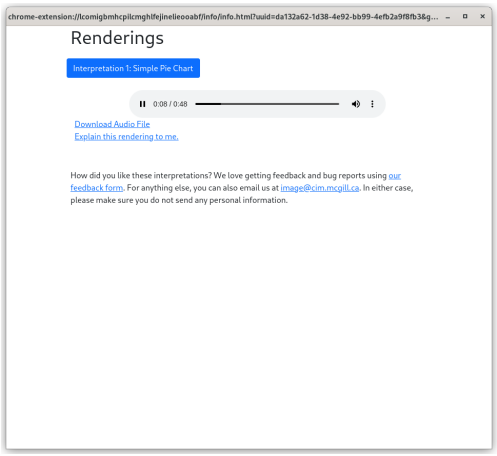
We designed and implemented the IMAGE architecture to follow the three-step pipeline and requirements specified in Section 2.3. Collection of graphical data is performed by a client, which is in our implementation the aforementioned Google Chrome browser extension. The processing (implemented by *preprocessor* microservices) and synthesis (implemented by *handlers*) steps occur on a server across numerous microservices running as Docker[3] containers. These microservices were containerized so they can be more easily reused by other teams while avoiding the added computational overhead that full virtualization would incur [9]. While several containerization technologies are available, Docker was chosen due to its extensive documentation and widespread use in the developer community. These factors increase the likelihood that accessibility practitioners would have a degree of familiarity with Docker and could otherwise learn to use it quickly. Additionally, an *orchestrator* microservice exists to detect available preprocessors and handlers, manage communications between the client and server, and manage the flow of requests within the server. Various *helper services* may also exist to perform functions common to multiple handlers, for example speech synthesis or sound spatialization. An overview of how these different components are organized is shown in Figure 2.

Expanding on the pie chart example in Figure 1, the rendering described there is generated in the following steps:

[1] https://www.highcharts.com/
[2] Readers are encouraged to try the extension themselves, available from https://image.a11y.mcgill.ca.
[3] https://www.docker.com/



(a) An IMAGE-compatible pie chart of COVID-19 case information present on a Quebec government website. As the chart uses Highcharts data, a button to interpret the data using IMAGE is inserted below it.



(b) A renderings window generated by the browser extension. For this pie chart, only one interpretation was returned. When this interpretation is navigated to and expanded, the user can play an audio file containing a sonification of the chart.

Fig. 1. Screenshots of the IMAGE browser extension being used to produce an audio representation of a pie chart.

(1) The user triggers the creation of a request, structured as a JSON object, that contains the chart data itself as well as information about the graphic's context in the web page, user preferences, and a list of features the browser extension supports.
(2) The orchestrator receives this request and sorts microservices running on the same virtual network into preprocessors and handlers based on labels identifying the containers as such.
(3) The orchestrator sends the request in parallel to all preprocessors in a priority group, starting with the group with the lowest number and moving to the one with the highest. It waits to receive a response from all preprocessors in a group and appends the data in these responses to the original request. For the pie chart, this is limited to commonly used statistics, but for other graphics would involve machine learning models building upon the outputs of one another.
(4) The orchestrator sends the request, including all preprocessor data, to the handlers in parallel and waits for them to respond with a list of *renderings* that represent the graphic. For the pie chart, one handler might create a spatialized audio experience of the wedges, and another handler might create a haptic experience to be displayed on a connected refreshable pin array.
(5) Handlers may use a helper service, not automatically detected by the orchestrator, to perform certain common tasks. In the COVID-19 chart example, helper services are used to synthesize speech and generate the actual spatialized audio sonifications of the pie chart data for use in the overall rendering being created by the handler.
(6) All of the renderings from the handlers are concatenated by the orchestrator and sent back to the extension as a response to the original request. These results are then displayed to the user, who can choose between them to listen to the audio representation of the chart or touch the tactile representation.

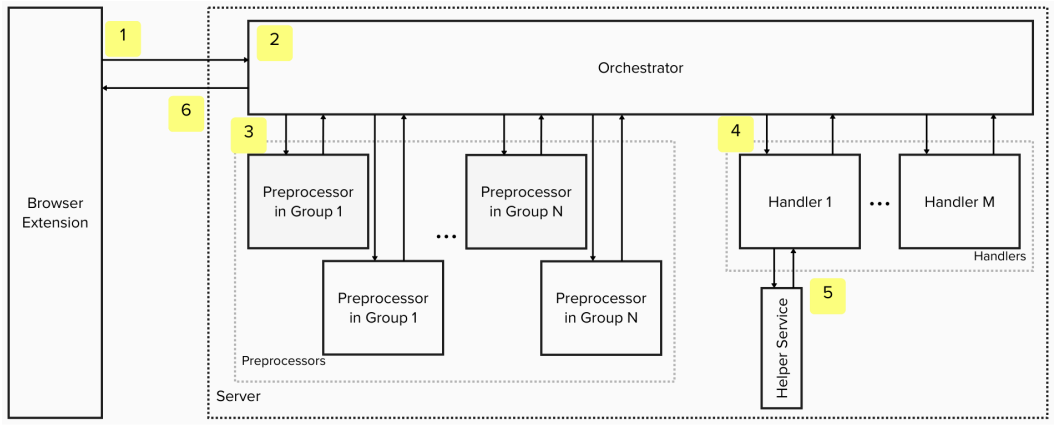


Fig. 2. Connections between major components in the IMAGE architecture. The browser extension communicates solely with the orchestrator running on the server. The orchestrator then communicates with preprocessors, each belonging to a priority group, and handlers. Data produced by a preprocessor are available to those that run in later groups since they are appended to the request by the orchestrator (step 3). Handlers may send requests directly to helper services to accomplish tasks common to multiple handlers. Number annotations are added corresponding to the steps to produce renderings of a graphic.

Each time data are exchanged in these steps, they are validated against a JSON schema[4] that specifies the structure and content required. Although this necessitated a significant amount of additional effort, we found that the rigour enforced by the schema helped in subsequent debugging efforts when problems with renderings were detected. Consider what may happen otherwise: a preprocessor could produce a value out of the expected range. A handler that attempts to use these data would then either experience a runtime error or create an erroneous rendering. In both cases, the issue is harder to detect and would require work to determine the source of the problem in the preprocesor, either working backwards from the handler or from the extension. Since data are validated before being sent by the preprocessor, the error occurs within the preprocessor itself and debugging can begin there. Since these schema files are human readable and contain descriptions only useful to those working with the framework, they also function as documentation for the data structures in use. Although such methods are commonly used in software engineering in general, by baking them into the framework, best practices are more likely to be followed, raising the reliability of the system as a whole for all contributors.

The pie chart example is only one possible path through the IMAGE framework. However, IMAGE is flexible enough to be used for a wide variety of accessibility scenarios using different graphics, means of representing content, and user preferences. We describe how the architecture supports this flexibility, and then demonstrate how these features could be used with use cases outside the scope of the IMAGE project.

3.2 Different Graphics, Renderers, and Capabilities

The IMAGE architecture includes functionality to work with multiple types of graphics and generate different experiences for different users. Accordingly, a single server can support multiple use cases. This is possible because preprocessors and handlers are configured to ignore data with which they cannot work.

[4]https://json-schema.org/

For example, in a system supporting both map and photograph inputs, preprocessors and handlers that can only meaningfully work with photographs will receive data on maps from the orchestrator, but will immediately respond that they can provide nothing, allowing the orchestrator to proceed. Ultimately, the client, and by extension the user, only receives a response with data from the appropriate microservices for the graphic included in the request. Support for different types of graphics is configured internal to each microservice, reducing the overhead on those responsible for managing the server instance.

Although there are not universal standards for embedded elements such as charts and maps, it is possible to include new formats in the IMAGE architecture. If a desired format is similar to a supported one, the client can be configured to convert it to the supported type so that existing server components can be used unchanged. In other cases, a practitioner can extend the IMAGE request schema to accept the new format, and add or modify components as necessary to produce renderings with the data.[5]

Multiple means of displaying the renderings produced by handlers will be necessary to support different experiences, client software, and user preferences. This information needs to be communicated in some way to a server and to the handlers running on it. To accomplish this, each request sent by a client contains two fields with complementary roles: *renderers* and *capabilities*.

Renderers define both a data structure that can be used by a rendering and an implicit interaction flow to display the rendering to a user. For example, an annotated sonification of the COVID-19 pie chart in Figure 1a would be internally structured to include an audio file of the sonification, the timestamps marking when each chart element is sonified, and text labels for these features. Any client that receives this rendering and recognizes this format would be expected to display the audio file to the user with suitable controls for navigating between the labelled content within it and playing the corresponding audio. Clients are expected to have a built-in list of renderers or to dynamically generate such a list based on the resources available on the client platform. For example, a client that cannot access a refreshable pin array must not advertise a renderer that requires such a device.

The *capabilities* field includes a list of preferences that impact content within or across various renderers, as relevant to tailoring the output to individual differences and requirements across the user community. For example, if a user only wishes to receive mono audio, this would be specified as a capability since it would impact many renderers. The combined effect of these fields is that the server is able to produce renderings that are most relevant and personalized to the user. At a basic level, this also ensures that renderings are able to be properly displayed by the client and its connected hardware. Capability selection should be determined automatically when related to available hardware, but options should be available to allow users to adjust or disable aspects of the experience according to their preferences. For example, a user who only has hearing in one ear would likely prefer to avoid renderings that heavily rely on spatialized audio. Renderers and capabilities are configured in such a way that other practitioners and groups can add new functionalities to the IMAGE architecture without the need to worry about causing a conflict with features we or other groups have made.

3.3 Applying the Architecture Outside the IMAGE Project

The extensibility of the IMAGE architecture is, we believe, its primary benefit to other accessibility researchers and developers. We describe two hypothetical examples to illustrate how this

[5]An example of this is outlined in Section 3.3.1.

architecture would reduce the resources that need to be allocated to such software infrastructure development.[6]

3.3.1 Interactive Radar Maps for Weather Systems. Although much of weather forecast information is included as text and is thus fully accessible to people using screen readers, visualizations of these systems changing over time is a notable exception. For the sake of discussing a path to implementation, we use Environment and Climate Change Canada (ECCC) radar maps as an example. The open data available via the Meterological Service of Canada GeoMet platform[7] is used to produce weather systems visualizations, but could instead equally be used to produce sonifications.

Deploying such a data processing pipeline within the IMAGE system would involve:

(1) Creating a new input type in the request data structure that specifies the geographic location from which to retrieve radar data and the desired time range.
(2) Implementing a feature within the existing IMAGE browser extension to obtain these data from an ECCC forecast radar map on a web page.
(3) Writing a new schema file to describe the structure of the data that will be obtained from the GeoMet platform. This could be identical to the format used in the platform itself or something more generic so that it would apply to other weather data services in the future.
(4) Implementing a new preprocessor that sends a query to GeoMet and returns the data in the format specified in the previous step.
(5) Selecting an existing audio renderer.[8]
(6) Implementing a new handler that generates the sonifications in the renderer format specified in the previous step and implementing a new renderer in the client if necessary.

The team working on this project would only need to be concerned with the implementation of new functionality. The infrastructure to connect components is already present, and benefits increase as other components can be reused. A new client or renderer can be implemented if a practitioner determines that one is necessary. Creating a new project-specific client does not require modifications in other components since the interfaces between them are unchanged. A new renderer, however, does require support to be present in handlers and clients that produce or consume data in the format used by the renderer. Changes or extensions to valid requests will require the schema consumed by the orchestrator to be updated, which can be done at runtime without modifying the underlying Docker image by mounting schema files into the container.

3.3.2 Automatic Captioning System. To further illustrate the disparate scenarios that the IMAGE architecture can support, we consider an automatic captioning system for audio that includes rich descriptions of non-speech elements. Such a project would likely require evaluating a variety of components for different tasks (e.g., speech recognition, speaker identification) and potentially different implementations of each component. Deploying a pipeline to accomplish this captioning task would entail the same set of tasks as described for the radar maps project. The client would be modified to work with audio files and new data types would centre on producing caption data. Creating an end-to-end experience could involve detecting embedded audio elements in a browser extension, adding a button to generate a request for these elements, implementing new

[6]Further details for developers is available on our GitHub project's Wiki https://github.com/Shared-Reality-Lab/IMAGE-server/wiki/3.-Creating,-testing,-and-deploying-preprocessors-and-handlers
[7]https://eccc-msc.github.io/open-data/msc-geomet/readme_en
[8]For example, https://github.com/Shared-Reality-Lab/IMAGE-server/blob/schemas/renderers/segmentaudio.schema.json

Team Member	Primary Responsibilities
M1	Audio design and user research
M2	Browser extension development and preprocessor code review
M3	Haptic design and implementation
M4	Machine learning and preprocessor implementation
M5	User interface design and user research
M6	Haptic design and implementation

Table 1. Brief profiles of the interviewees.

preprocessors and handlers that label sounds and speech, and writing a new renderer that displays the results of these components as WebVTT captions.[9]

While new components are necessary in these hypothetical scenarios, these additions are to provide use case specific functionality. A client and renderers would already be available in both cases to use or modify to meet the needs of the projects, and modules can be replaced as the project develops as improvements are made available. For example, if a more efficient sound classification model is released while the automatic captioning system project is under development, a new preprocessor could be written using it that could act as a drop-in replacement for one developed around another model.

Not included here are the processes of testing and refinement that are necessary to ensure the new data structures, new microservices, and modifications to the client come together to produce the best possible experience for users. Supporting practitioners in this design process is a challenge for any complex project. Naturally, we have faced this challenge within the IMAGE project and have assessed our response to it, developing practices and identifying important areas that we believe will be of use to groups that choose to use our architecture for their own work. In the following section, we describe some of these practices and lessons learned from our experiences as developers.

4 PROJECT MEMBER INTERVIEWS

While practitioners outside the IMAGE project have used components developed for the IMAGE architecture, we are not yet aware of any who have developed end-to-end experiences using it. In order to understand the ways in which the framework can be used throughout a project and to identify real problems relevant to others working with it, we sought to understand the experience of our team members. The outcomes of these interviews may be useful in the planning of other web graphic accessibility or similar projects, especially those that may develop or use components developed in our framework.

4.1 Overview

Six members of our team who worked on aspects such as user experience and testing, haptic design, audio design, machine learning, and software development agreed to participate in semi-structured retrospective interviews conducted by the first author. Over the course of an hour, interviewed team members were asked to discuss their work experiences and challenges from the time they joined the project through the time of the interview in the context of phases of brainstorming and exploration, prototyping and testing, and deployment and maintenance. Brief profiles for each team member are shown in Table 1. Note that none of the interviewees were directly involved in the design or implementation of the IMAGE architecture, which occurred in parallel with other

[9]https://www.w3.org/TR/webvtt1/

tasks from the start of the project. Topics of discussion were not limited to strictly architectural or technical concerns. Instead, a goal was to form a more complete understanding of the interactions between the challenges that people experienced, and to identify areas missed by the architecture, if applicable. Each interview session was transcribed by the first author and analyzed by the first and third authors to determine commonalities relevant to the use of the architecture and to the project as a whole.

4.2 Challenges Identified and Improvements

Several recurring issues were identified in the team member interviews. First, multiple participants reported experiencing difficulties learning how to use the framework or the technologies it uses, or having observed others face these problems. The initial documentation prepared and collected for the IMAGE architecture focused on its operation at a high level, rather than responding to specific problems that people using it would face. To address this challenge, we produced new resources tuned to common tasks performed by members of the IMAGE project, such as developing, testing, and deploying a new preprocessor.

Second, the internal state of an IMAGE server instance or a preprocessor within one were difficult to determine, leading to confusion among users of the system. For example, M3 reported a case where he was unable to determine which version of a handler was in use. M1, M4, and M6 shared experiences where they struggled to understand what computer vision preprocessors were determining about particular inputs and how their outputs contributed to the rendering. We wrote a server-side script to display the preprocessors and handlers detected by an orchestrator, and the order in which these preprocessors would respond to a request to address the issue of overall server state. Continuous integration processes were also developed to ensure that production and staging components are built in a consistent manner. For within-preprocessor visibility, we added debugging-specific handlers to visualize data that are difficult to understand in JSON outputs, e.g., the locations of detected objects in a photograph (Figure 3). These handlers are inactive unless a debugging flag is sent in the request.

Third, there was a lack of understanding by some team members of what enables components to be meaningfully reusable. Several components developed early in the project, especially adding initial haptic functionality, were properly written for the IMAGE architecture, but were so narrowly tailored to a particular use case that were unlikely to be practical to reuse. We later determined that this was due to team members not understanding this distinction and attempting to implement complex functionality in a single feature. Additional instruction on, and hands-on experience with, reusing IMAGE components were beneficial, even for people with a background in software development.

4.3 Support for Practitioners

A key goal of the interviews was to determine the experience of practitioners using the architecture in the development, design, and research tasks described in Section 2.3. Since the work of the participants in these interviews touches on all these aspects of accessibility projects, their relevant comments are synthesized here. Perhaps unsurprisingly, development tasks were the most clearly supported by the architecture. M2, M3, and M4 all reported that this framework was helpful in tasks where communication across components developed by different people was required. M6 also shared that the structure afforded by the architecture helped him focus more on the tasks on which he worked.

Research and evaluation were supported as well, albeit to a lesser extent than development. While concerns about the consistency of renderings prevented the full end-to-end system from being used in studies, components could still be used outside of the architecture. For example, handlers were

used to quickly produce representations for evaluation by manually modifying requests to meet the specifications of the researcher. These renderings could then be used to drive different user interface prototypes to demonstrate functionality that could easily be added to the complete system if it provided a benefit to users. Outputs of preprocessors could also be used directly without a handler producing IMAGE renderings. The ability to adapt components in this manner was viewed positively during the interviews and has been used by at least one external research group.

The design of new experiences is an aspect where the impact of the architecture was more mixed, and participants shared cases where they experienced problems. In one case, team members implemented an end-to-end design concept using the framework without sufficient brainstorming or refinement. The architecture increased the difficulty of making major changes to the design in response to later user feedback since modifications needed to be coordinated across several components. In another case, a team member was brainstorming sonifications for a type of graphic, and reported experiencing high cognitive load due to a perceived need to make these ideas fit into different architectural components, e.g., preprocessors and handlers. Interviewees indicated that attempting to fit possible experiences into the structure of the IMAGE architecture yielded lower quality results early in the design process. However, factors unrelated to the architecture, such as the COVID-19 pandemic, were noted by the team members as being far more significant. Further, the effect of the framework was reversed when team members worked on more clearly defined issues or more mature prototypes. In these cases, the support provided by the architecture was appreciated. Issues primarily arose when issues of development within the framework were considered too early for the scope of the task, and served to constrain how practitioners approach a problem rather than provide paths to resolving it. We have adjusted our own internal practices to encourage more effort spent "outside the architecture", and have had observed fewer instances of these problems occurring.

The response from those who worked with the architecture on the IMAGE project was generally positive, and all of the participants expressed that the architecture benefited their work and the project as a whole. Specifically, the broad roles of components, data validation by schemas, and ease in replacing or deploying microservices facilitated the adoption of new use cases, software development and maintenance, and the progression of designs from prototypes to production. With the improvements made in response to the key issues identified in Section 4.2, the framework as it has been developed thus far appears to meet the requirements identified for web graphic accessibility projects. Further refinements are planned based on the needs of the IMAGE project and in response to future use by other research groups.

5 LIMITATIONS AND FUTURE WORK

While other researchers have used components developed in the architecture as part of their own work, development and deployment with the full framework has, to our knowledge, only been performed by the IMAGE team itself. As a result, our evaluation has focused on our team's usage since it is the most extensive. However, other challenges could be faced by teams using the IMAGE architecture in contexts we have not anticipated and considered in our design or evaluation.

The most significant technical limitation is that inputs and outputs must be encapsulated in one HTTP request and response, preventing support for large requests or use cases that require ongoing communication between client and server. We plan to address this limitation in future improvements to the framework, for example, by adding support for user sessions that persist beyond a single request.

The architecture has not yet received sufficient load testing under real-world conditions to ensure it will not encounter unforeseen bottlenecks. We expect that scaling the microservices of an instance could be accomplished in a straightforward manner using tools such as Docker

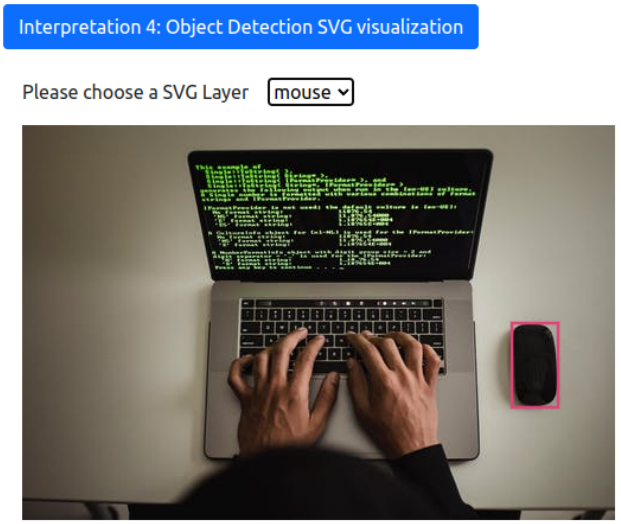


Fig. 3. A screenshot of visual debugging output as shown in the interpretations list within the browser extension. Note that the mouse is outlined in red in the original photograph, where the object detection preprocessor found it. Other types of objects can be selected from the dropdown menu.

Swarm[10] or Kubernetes,[11] although we have not tested them. A federated model in which separate instances communicate with one another to combine their offered services could also address these same concerns while reducing the resource requirements in deploying an IMAGE server, either for production purposes or for integration testing of a component. Similarly, we expect that the framework could be modified to run on another containerization platform with only moderate changes to the orchestrator.

We are also extending IMAGE to new platforms, such as mobile devices, as well as incorporating new haptic devices. Although we expect that the current server architecture will be sufficiently flexible to support these use cases, the browser extension may be unusable on such platforms, requiring us to create custom client applications instead. Some of these devices may also require more dynamic interactions on the end-user device, necessitating local rendering of audio and haptic content. Extensions to the architecture to allow some assets to be created on the server and others locally on the client are currently being designed.

Finally, an authoring tool would help support designers in testing ideas and integrating them into the architecture. While difficulties were faced by interviewees when working within the architecture in early design phases, working fully outside of it, however, leads to greater technical and design difficulties when later implementing and evaluating a selected idea. An environment that uses the same technical base for displaying renderings as in the architecture, but targets quick examples over production-ready experiences, provides a compromise between these two extremes. We envision an authoring tool that allows for designers to quickly mock up and modify audio-haptic renderings in an interface that requires less technical expertise than is needed now, a useful feature since few designers are equally comfortable working with audio as they are with haptics, or vice versa.

[10]https://docs.docker.com/engine/swarm
[11]https://kubernetes.io/

6 CONCLUSION

We have presented the IMAGE architecture, a platform for converting online content between modalities. From this description, we expect that practitioners will be able to decide whether IMAGE is an appropriate framework on which to base their own work, especially for generating novel multimodal experiences for accessibility purposes. In addition, we have conveyed the results of an internal team review of using our architecture, highlighting pitfalls and practical advice useful both for those building on top of it, or those contemplating building their own framework similar to IMAGE.

We designed IMAGE as a platform to encourage accessible, multimodal transformations of graphical content to be deployed to end users, and look forward to others adopting it for their own purposes. We welcome feedback and ideas that can help guide IMAGE's direction so that the it will be even more flexible to support new use cases. Our hope is, by publishing our platform details, we will encourage practitioners to extend it so that we can identify its strengths and weaknesses and continue making improvements.

ACKNOWLEDGMENTS

The authors thank all current and former team members in the IMAGE project, especially those who participated in interviews. The McGill Research Ethics Board indicates that the interviews included in Section 4 are not subject to ethics approval as they consist of team members reflecting on their individual experience with the project and their perspectives on what was done during that time.

The first author is supported by the Natural Sciences and Engineering Research Council of Canada under Canadian Graduate Scholarship (Doctoral) no. 569236 and the Fonds de recherche du Québec—Nature et technologies under doctoral scholarship no. 315050. During the development period discussed, IMAGE was also funded by Innovation, Science and Economic Development Canada under project no. 513221 of the Accessible Technology Program, and by Healthy Brains, Healthy Lives under an Ignite grant.

REFERENCES

[1] 2019. What's New in JAWS 2019 Screen Reading Software. Retrieved 2023-01-04 from https://support.freedomscientific.com/Downloads/JAWS/JAWSWhatsNew?version=2019

[2] 2021. WebAIM: Screen Reader User Survey #9 Results. Retrieved 2024-01-10 from https://webaim.org/projects/screenreadersurvey9/

[3] 2022. Use VoiceOver for images and videos on iPhone. Retrieved 2024-01-10 from https://support.apple.com/en-ca/guide/iphone/iph37e6b3844/ios

[4] 2022. WebAIM: The WebAIM Million - The 2022 report on the accessibility of the top 1,000,000 home pages. Retrieved 2024-01-04 from https://webaim.org/projects/million/2022

[5] 2023. NVDA User Guide. Retrieved 2024-01-10 from https://www.nvaccess.org/files/nvda/documentation/userGuide.html

[6] 2024. Orca Screen Reader. Retrieved 2024-01-10 from https://help.gnome.org/users/orca/stable/

[7] 2024. Seeing AI – Talking Camera for the Blind. Retrieved 2024-01-10 from https://www.seeingai.com/

[8] Saša Baškarada, Vivian Nguyen, and Andy Koronios. 2020. Architecting Microservices: Practical Opportunities and Challenges. *Journal of Computer Information Systems* 60, 5 (Sept. 2020), 428–436. https://doi.org/10.1080/08874417.2018.1520056

[9] Ouafa Bentaleb, Adam S. Z. Belloum, Abderrazak Sebaa, and Aouaouche El-Maouhab. 2022. Containerization technologies: taxonomies, applications and challenges. *The Journal of Supercomputing* 78, 1 (Jan. 2022), 1144–1181. https://doi.org/10.1007/s11227-021-03914-1

[10] Jeffrey P. Bigham, Wendy Chisholm, and Richard E. Ladner. 2010. WebAnywhere: experiences with a new delivery model for access technology. In *Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A) - W4A '10*. ACM Press, Raleigh, North Carolina, 1. https://doi.org/10.1145/1805986.1806007

[11] Jeffrey P. Bigham, Chandrika Jayant, Hanjie Ji, Greg Little, Andrew Miller, Robert C. Miller, Robin Miller, Aubrey Tatarowicz, Brandyn White, Samual White, and Tom Yeh. 2010. VizWiz: nearly real-time answers to visual questions. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology (UIST '10)*. Association for Computing Machinery, New York, NY, USA, 333–342. https://doi.org/10.1145/1866029.1866080

[12] Jeffrey P. Bigham, Chandrika Jayant, Andrew Miller, Brandyn White, and Tom Yeh. 2010. VizWiz::LocateIt - enabling blind people to locate objects in their environment. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*. IEEE, San Francisco, CA, USA, 65–72. https://doi.org/10.1109/CVPRW.2010.5543821

[13] Luís P. Carvalho, Tiago Guerreiro, Shaun Lawson, and Kyle Montague. 2023. Towards real-time and large-scale web accessbility. In *Proceedings of the 25th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '23)*. Association for Computing Machinery, New York, NY, USA, 1–9. https://doi.org/10.1145/3597638.3608403

[14] Luis Cavazos Quero, Jorge Iranzo Bartolomé, and Jundong Cho. 2021. Accessible Visual Artworks for Blind and Visually Impaired People: Comparing a Multimodal Approach with Tactile Graphics. *Electronics* 10, 3 (Jan. 2021), 297. https://doi.org/10.3390/electronics10030297

[15] Luis Cavazos Quero, Jorge Iranzo Bartolomé, Seonggu Lee, En Han, Sunhee Kim, and Jundong Cho. 2018. An Interactive Multimodal Guide to Improve Art Accessibility for Blind People. In *Proceedings of the 20th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '18)*. Association for Computing Machinery, New York, NY, USA, 346–348. https://doi.org/10.1145/3234695.3241033

[16] Darren Guinness, Edward Cutrell, and Meredith Ringel Morris. 2018. Caption Crawler: Enabling Reusable Alternative Text Descriptions using Reverse Image Search. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/3173574.3174092

[17] Michael Heron, Vicki L. Hanson, and Ian W. Ricketts. 2013. ACCESS: a technical framework for adaptive accessibility support. In *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems (EICS '13)*. Association for Computing Machinery, New York, NY, USA, 33–42. https://doi.org/10.1145/2494603.2480316

[18] Leona M Holloway, Cagatay Goncu, Alon Ilsar, Matthew Butler, and Kim Marriott. 2022. Infosonics: Accessible Infographics for People who are Blind using Sonification and Voice. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems (CHI '22)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3491102.3517465

[19] Md Naimul Hoque, Md Ehtesham-Ul-Haque, Niklas Elmqvist, and Syed Masum Billah. 2023. Accessible Data Representation with Natural Sound. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23)*. Association for Computing Machinery, New York, NY, USA, 1–19. https://doi.org/10.1145/3544548.3581087

[20] Pooyan Jamshidi, Claus Pahl, Nabor C. Mendonça, James Lewis, and Stefan Tilkov. 2018. Microservices: The Journey So Far and Challenges Ahead. *IEEE Software* 35, 3 (May 2018), 24–35. https://doi.org/10.1109/MS.2018.2141039

[21] Ralph E. Johnson. 1997. Frameworks = (components + patterns). *Commun. ACM* 40, 10 (Oct. 1997), 39–42. https://doi.org/10.1145/262793.262799

[22] David Ledo, Steven Houben, Jo Vermeulen, Nicolai Marquardt, Lora Oehlberg, and Saul Greenberg. 2018. Evaluation Strategies for HCI Toolkit Research. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–17. https://doi.org/10.1145/3173574.3173610

[23] Charles Lenay, Olivier Gapenne, Sylvain Hanneton, Catherine Marque, and Christelle Genouëlle. 2004. Sensory Substitution: Limits and Perspectives. In *Advances in Consciousness Research*, Yvette Hatwell, Arlette Streri, and Edouard Gentaz (Eds.). Vol. 53. John Benjamins Publishing Company, Amsterdam, 275–292. https://doi.org/10.1075/aicr.53.22len

[24] Christina Low, Emma McCamey, Cole Gleason, Patrick Carrington, Jeffrey P. Bigham, and Amy Pavel. 2019. Twitter A11y: A Browser Extension to Describe Images. In *The 21st International ACM SIGACCESS Conference on Computers and Accessibility*. ACM, Pittsburgh PA USA, 551–553. https://doi.org/10.1145/3308561.3354629

[25] Alexandros Pino and Georgios Kouroupetroglou. 2010. ITHACA: An Open Source Framework for Building Component-Based Augmentative and Alternative Communication Applications. *ACM Transactions on Accessible Computing* 2, 4 (June 2010), 14:1–14:30. https://doi.org/10.1145/1786774.1786775

[26] Albert Postma, Sander Zuidhoek, Matthijs L Noordzij, and Astrid M L Kappers. 2007. Differences between Early-Blind, Late-Blind, and Blindfolded-Sighted People in Haptic Spatial-Configuration Learning and Resulting Memory Traces. *Perception* 36, 8 (Aug. 2007), 1253–1265. https://doi.org/10.1068/p5441

[27] Juliette Regimbal, Jeffrey R. Blum, and Jeremy R. Cooperstock. 2022. IMAGE: A Deployment Framework or Creating Multimodal Experiences of Web Graphics. In *Proceedings of the 19th International Web for All Conference*. ACM, Lyon France, 1–5. https://doi.org/10.1145/3493612.3520460

[28] Letícia Seixas Pereira, José Coelho, André Rodrigues, João Guerreiro, Tiago Guerreiro, and Carlos Duarte. 2022. Authoring accessible media content on social networks. In *Proceedings of the 24th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '22)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/3517428.3544882

[29] Steven A. Wall and Stephen A. Brewster. 2006. Tac-tiles: multimodal pie charts for visually impaired users. In *Proceedings of the 4th Nordic conference on Human-computer interaction: changing roles (NordiCHI '06)*. Association for Computing Machinery, New York, NY, USA, 9–18. https://doi.org/10.1145/1182475.1182477

[30] R. Michael Winters, Neel Joshi, Edward Cutrell, and Meredith Ringel Morris. 2019. Strategies for Auditory Display of Social Media. *Ergonomics in Design* 27, 1 (Jan. 2019), 11–15. https://doi.org/10.1177/1064804618788098

[31] Shaomei Wu, Jeffrey Wieland, Omid Farivar, and Julie Schiller. 2017. Automatic Alt-text: Computer-generated Image Descriptions for Blind Users on a Social Network Service. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*. ACM, Portland Oregon USA, 1180–1192. https://doi.org/10.1145/2998181.2998364

[32] Wai Yu and Stephen Brewster. 2003. Evaluation of multimodal graphs for blind people. *Universal Access in the Information Society* 2, 2 (June 2003), 105–124. https://doi.org/10.1007/s10209-002-0042-6

Received 1 April 2023; revised 10 January 2024; accepted 6 May 2024