INFORMATION AND COMMUNICATION TECHNOLOGIES (ICT)

Open Accessibility Everywhere

Groundwork, Infrastructure, Standards

AEGIS

Collaborative Project

224348

AEGIS Open Accessibility Framework

version 1.2.5

VERSION HISTORY TABLE

Version no.	Dates and comments
1	September, 2012: Updates from D1.2.1 to include developments in mobile accessibility APIs: iOS 6 and Android 4.1 "Jelly Bean".
2	October 2012: Further updates, incorporation of Annex and sent for peer review.
3	November 2012: Final version, after peer review.
1.2.4	February 2013: Further updates as per EC final review requests: 1. Put under Creative Commons "Open Source" license 2. Add a version number (change from AEGIS Deliverable numbering) 3. Add "evaluation & testing" component to the OAF
1.2.5	Updated Section 2.5.3 "Adapting the OAF over time" Replaced some images with latest editions Final check of ALT text for new images

Table of Contents

List of Drawings

Abbreviations List

Abbreviation	Explanation
API	Application Programming Interface
ARIA WAI	Accessible Rich Internet Applications, draft specification from the Web Accessibility Initiative
AT	Assistive Technology
ATAG	Authoring Tools accessibility guidelines
CCF	Concept Coding Framework
DOM	Document Object Model
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
ICT	Information and Communication Technologies
IT	Information Technology
OAEG	Open Accessibility Everywhere Group
OAF	Open Accessibility Framework
ODF	Open Document Format
OS	Operating System
PDA	Personal Digital Assistant
RIA	Rich Internet Applications
RIM	Received Input Message
SDK	Software Development Kit
TTS	Text to Speech
UAAG	User Agent Accessibility Guidelines
UCD	User Centered design
WAI	Web Accessibility InitiativeShare and Share
WP	Work package

EXECUTIVE SUMMARY

The Open Accessibility Framework as delivered in AEGIS is two things:

- A document describing the framework of things needed for 3rd generation accessibility, as validated by the prototypes and user/developer feedback in AEGIS
- A collection of largely open source prototypes and code Deliverables implementing various aspects of the OAF, proven in AEGIS and contributed back to the open source projects of which they are part

This document contains the AEGIS Open Accessibility Framework (OAF) description. It is based on the original OAF Deliverable, "D1.2.1 AEGIS OAF and high-level architecture"[14], with modifications described below at the end of this Executive Summary.

The original AEGIS OAF Deliverable (D1.2.1) was based upon:

- The accessibility API and framework support from the existing GNOME Accessibility framework and the Java platform (the Java Accessibility API, keyboard operability guidelines, and theme support)
- The AEGIS generic accessibility framework requirements (developed as AEGIS internal deliverable ID1.2.1)

That original Deliverable (D1.2.1) was informed by the early feedback from AEGIS consortium developments – which highlighted anticipated areas that the OAF would necessarily need to cover. This includes:

- ARIA implementations on various UI elements
- Initial work on JavaFX accessibility
- Alternate input systems for users with physical impairments (both for the open desktop as well as "thought experiments" for mobile)
- Development of the RIM Blackberry mobile accessibility API and its use by the Oratio screen reader

Further, that original Deliverable was informed by developments in the field of accessibility external to AEGIS development work. That included:

- ISO 13066 work to standardize AT-IT interoperability generally, and specifically to codify the set of information that must be provided via accessibility APIs
- The U.S. Access Board release of their "Advanced Notice of Proposed Rulemaking" for the refresh of the Section 508 and Section 255 accessibility regulations/guidelines
- The Apple iPhone 3GS/4 (and iPad) which include a built-in screen reader and screen magnifier, both of which rely on a new set of multi-touch gestures for use
- The Android v1.6 and later operating system which includes the 'Talk Back' screen reading functionality

This document takes into account what we learned during the course of creating the prototypes and code Deliverables which implement various aspects of the OAF. It has also been updated where appropriate based on product and engineering developments in the field, as well as accessibility standardization efforts over the last few years. Places where we have made updates are clearly noted in the document; otherwise the text is as it was in D1.2.1. This document has also been published under the Creative Commons Attribution-Share Alike 3.0 License, with version number.

The major influences on the OAF from our AEGIS development work were:

- The complete OAF implementation – all six steps – for the Java Mobile environment
- Development of the Tecla AT for users with upper limb impairments on Android (and the commercial spin-off implementation of Tecla on iOS)
- Development of our "Accessibility Advisor" wizard for Web & Mobile accessibility work, and the work we have done to support developers in NetBeans and DroidDraw
- The work we did on user requirements, interaction models, and derived user persona (from D1.3.1 and D1.3.2), which is connected with the OAF and which is attached as an appendix.

The document is divided into the following six chapters, a set of references for the first six chapters, and a separate report attached as an Appendix:

- Chapter 1 "Introduction" describes the background for and influences on the Open Accessibility Framework. We have updated this chapter to note the work of the W3C WCAG2ICT Task Force (which AEGIS members have participated in), and the worldwide harmonization efforts to ensure that the Section 508 refresh and Mandate 376 are using the same technical "definition of accessible"- much of which is significantly derived from the WCAG2ICT work. There were also minor updates related to the most recent iOS and Android releases.
- Chapter 2 "The Open Accessibility Framework " describes the six steps of the Open Accessibility Framework, and also notes those aspects of 3rd generation accessibility that go beyond those six steps. We included a minor update related to OAF step #5 and cognitive impairments. We also included a more significant update that shows where evaluation & testing can occur.
- Chapter 3 "The Accessibility API in detail" goes into detail on one of the most important facets of the OAF – the API definition of accessibility, as implemented by user-interface elements, and supported by platforms.
- Chapter 4 "How the OAF is used by various assistive technologies, and thereby for various users" uses the almost prosaic example of alternate text for an image to example how that touches every aspect of the OAF – in the domains of HTML web content, an ODF document, a Java Swing application, and then for comparison purposes the older 2nd generation accessibility environment.
- Chapter 5 "Research Results for the OAF From AEGIS Developments" describes the OAF related research and development work we undertook in AEGIS, and notes what we learned and how that impacted the OAF.
- Chapter 6 "Conclusions" looks forward to how the OAF itself may grow and change, both within AEGIS and after the project conclusion.

- REFERENCES contains all of the references made in the main part of the document (the first six chapters)
- Appendix A “Integration of User Interaction Models and Profiles into the Open Accessibility Framework and high-level architecture” link the technical aspects of the OAF with the user requirements, interaction models and derived user persona detailed in D1.3.1 and D1.3.2

1. INTRODUCTION

Over the past 4 decades of computer adaptation to people with disabilities, three distinct approaches to providing accessibility have emerged. The first two approaches were tuned to the computing environments of the time, while the third has evolved out of an attempt to get past problems inherent in the previous approaches, and formally engineer accessibility into mainstream computing environments.

The first approach – what we call "1st generation accessibility"– was developed on the early character-based systems: C/PM, DOS, Apple II. This took place in the 1970s and 1980s. Disability access to character-based systems was focused on blind and low-vision users (using specialized video cards for magnification, and screen readers that extracted the text from the text-buffer of the video card). Emerging toward the end of this period were alternate keyboards for people with physical impairments. Also during this time specialized learning software for other disabilities was developed, including a growing body of educational applications designed with the disabled in mind.

The second approach emerged as a response to the graphical desktop environment of Macintosh, starting in the mid-to-late 1980s. On Macintosh, there was no longer any character buffer on a video card from which text could be extracted for a screen reader. Nor was there even a video card at all, making hardware-based magnification impossible. And while the computer still came with a keyboard, much interaction was done using a mouse. These numerous challenges spawned a new approach – what we call "2nd generation accessibility". The 2nd generation approach involves software magnification of the bitmap display (for low vision users), and populating/maintaining an off-screen model of the text contents and other meaningful information from the graphical display using patches to the rendering subsystem or display driver (a notoriously brittle technique). Alternative input systems grew in sophistication through replacing or hooking into the software layers for input devices, enabling on-screen keyboards in addition to physical hardware keyboard (and mouse) replacements. Also computers became powerful enough that software speech recognition became an option. And of course specialized learning software continued to be developed.

The third approach emerged out of a recognition that the patching and other reverse-engineering needed by many of the the 2nd generation wasn't sustainable. These approaches failed to work with applications that didn't render text in ways that could be captured via the display driver or text-rendering calls of the video subsystem (e.g. Java 2D, early uses of Adobe Postscript fonts, X Windows servers running on Windows & Macintosh, and Hypercard on the original Macintosh – back in 1998). Furthermore, the patching/reverse-engineering techniques had to be built anew each successive release of the Windows desktop.

3rd generation accessibility began in multiple places, nearly simultaneously. In the UNIX world around 1993, the Remote Access Protocol was developed as a layer to provide semantic accessibility information in parallel to the Motif user-interface library on X Windows. At the same time, on Macintosh a collection of assistive technology vendors worked on a specification for an accessibility API on Macintosh, to be called "Access Aware". Neither of these early approaches gained significant traction, and it wasn't until 1997 with the release of the Java Accessibility API and the release of the much more limited Windows accessibility API called Microsoft Active Accessibility that the 3rd generation approach really started to take hold. Also in 1997 the World Wide Web Consortium's Web Accessibility Initiative was launched, which began developing 3rd generation accessibility techniques for Web-based content. Even though it has been more than a decade since the introduction of this 3rd generation approach, there is still a large "legacy" of 2nd generation techniques in use on Windows – while the other two desktop environments of Macintosh and UNIX/Linux have

been firmly based on 3rd generation techniques for years. The newest accessible mobile platforms are also utilizing 3rd generation accessibility, while older mobile platforms – where they are accessible at all – achieve their accessibility via patching & reverse-engineering techniques similar to the 2nd generation desktop environments of the 1980s and 1990s.

The core of the 3rd generation approach is "access by contract", or the accessibility API. But the API is itself insufficient to give assistive technologies everything they need. Of course, the API must be implemented by the mainstream applications and operating system, and it must be sufficiently complete to cover all of the mainstream user interface modalities and it must provide all of the information needed by all (potential and actual) assistive technologies. Aiding in the exploitation of the accessibility API is the approach of pre-implementing that API on user interface element sets, such that any application that is based on such user-interface elements automatically gains an implementation of the accessibility API. The final component of exposing an accessibility API is addressing the "platform-on-a-platform" issue, where something is both an application and a platform (sitting on top of another platform) like a web browser or the Java or Flash runtime – with their own accessibility API (like WAI-ARIA or the Java Accessibility API) that must be translated to the accessibility API of the underlying platform (such as Windows or UNIX/Linux).

But we need more than a complete, and well implemented accessibility API. Assistive technologies need a variety of support services – things like text-to-speech – to exist on the underlying platform / operating system. We know what some of these are; as we develop the various technical Deliverables in AEGIS we may uncover more of these.

As noted above, accessibility APIs are typically implemented on one or more user-interface element sets (or "toolkits"). Another important task is distilling how any given portion of the accessibility API should be implemented on any given style of user interface element. This is to aid developers making new user interface elements / toolkits in properly implementing the accessibility API on those element(s). Further, as developers build applications with these UI elements, they often need to annotate them with accessibility metadata (e.g. associating names with images, creating labelling relationships between test entry fields and their related labels). Therefore, another important aspect of a complete framework to support accessibility is support within the developer tool to help developers add the necessary metadata.

Building on the notion of helping developers through the developers' tools, we extend that to helping non-developers who are nonetheless creating either interactive applications or other content (such as might appear on a web page or be circulated via e-mail) authors generally. Such "designer" and "author" tool support is another important facet in supporting accessibility.

With all of these things addressed, we have one final task remaining: ensuring that all users have available to them an accessible navigation / interaction model for the user interface. Unless the assistive technology provides its own user interface navigation scheme (cf. VoiceOver), some form of built-in UI navigation (e.g. keyboard operation) is needed.

These steps above form the starting point for the Open Accessibility Framework (OAF), which is described in detail in Chapter 3. The OAF is primarily for designers and developers of ICT components and systems – an high level blueprint to help them understand what they need to do to create accessible ICT as part of a 3rd generation ICT system / environment. It may also be of use to individuals and organizations procuring accessible ICT – to help them ensure that they only acquire ICT components and systems that are accessible. The section of the OAF that speaks to document authors in addition to application developers (section 2.3.3. Use domain) may be of interest to authors – though the key thing for them to know is that they should use an authoring tool that supports them in creating accessible documents and

content. Finally, the OAF is of interest to researchers, and anyone interested in understanding how 3rd generation accessibility works.

It is worth noting briefly at this point that the OAF does not address the software development process itself. There are a number of ways of developing accessible software; and distinct from the steps of OAF, there are many ways of organizing software development teams. The OAF does not assume any one specific design method or process. Similarly, the OAF doesn't itself speak to the usability of an accessible application (e.g. whether the accessible user-interface elements have been composed in a usable fashion; whether the keyboard operation scheme is the most usable and effective possible scheme).

For a treatment of the software development process as it speaks to developing accessible software, please see Chapter 3 of the AEGIS report "User groups' and stakeholders' definition and UCD Implementation Plan". [1]

1.1. Existing 3rd generation platforms informing the OAF

For reasons noted above in the introduction, since 1997 a variety of computing platforms have adopted 3rd generation accessibility techniques. A review of these platforms and the techniques they use have informed the AEGIS OAF work.

1.1.1. Java platform release 2 and later: Java Accessibility API and the Pluggable Look and Feel of of the Java Foundation Classes

The Java accessibility API – co-developed by the primary author of this document – was the original inspiration for the Open Accessibility Framework. Born out of necessity because 2nd generation reverse-engineering techniques simply couldn't work with the Java platform, the Java accessibility API was the first attempt at creating a comprehensive API that would supply all of the needs of assistive technologies, and that would be implementable by a variety of user-interface element sets.

Key concepts of the Java accessibility API for the OAF include:

1. Having a core set of accessibility API information common to all user interface elements
2. Having a layered approach with a set of optional sub-interfaces that are implemented as appropriate by different user interface elements (e.g. the "Action" optional interface for user interface elements that can react to user actions)
3. Developing an initial enumeration of the minimum set of information that must be in accessibility API – proven by the use of several assistive technologies and implementing it on several user interface toolkits and many many applications.[1]
4. Developing an API that can be implemented by a variety of custom user interface elements and user interface toolkits
5. Abstracting the notion of what a "user interface element" is from an accessibility point of view from that of the user interface toolkit – manifest in things like collapsing the user interface hierarchy through removal non-interactive nodes (e.g. pure grouping containers), and enabling "featherweight" accessible user interface elements (where the "element" might simply be a visual piece of a larger formal user interface element such as the Tab of the JTabbedPane in Swing)

[1]See Chapter 3 for this minimum set

6. The importance of funnelling all user interface events through to all assistive technologies that need them (vs. the peer-to-peer event model build into Swing classes)

Key concepts of the Java Foundation Classes for the OAF include:

1. Having sets of "pluggable look and feel" implementations or "skins", which pick up and mimic the underlying platform visual look and operational feel, and which theme when the underlying platform themes
2. Enabling developers to define their own custom focus traversal mechanism to ensure that every manipulable user interface element is in the focus traversal loop

Because of the very nature of the Java platform – that is is a "platform on a(nother) platform" - the work of Java accessibility also illuminated that unique challenge for the OAF. This was addressed for both the Windows and UNIX platforms by way of special "bridges". [33]

1.1.2. UNIX/GNOME 2.0 and later: ATK/AT-SPI and the GTK+ theme engine

The second key inspiration for the OAF – and in fact the first fairly complete implementation of 3rd generation accessibility on a platform – is the GNOME Accessibility framework comprised of the Accessibility Toolkit (or ATK) [7] and the Assistive Technology Service Provider Interface (or AT-SPI) on top of the GNOME desktop, and implemented for a variety of user interface toolkits including the core GNOME toolkit GTK+. [20] This API is derived from the Java accessibility API – written by the same company with some of the same development staff, and designed explicitly to support accessibility of the Java platform and also of the OpenOffice.org office suite whose underlying/internal accessibility framework [45] was likewise very close to the Java accessibility API (from the same company with some of the same development staff).

Unlike Java accessibility, GNOME accessibility had to stretch across all aspects of a desktop. Keyboard accessibility had to be much more comprehensive – addressing not just focus traversal within an application, but between applications and the various desktop features (multiple screens, the top and bottom panels, etc.).

Also unique to an "underlying platform" was handling the other half of the "platform-on-a-platform" challenge: enabling the variety of platforms that sit on top of the GNOME desktop to work with GNOME assistive technologies – e.g. Java, but also web content.

Finally, in addition to developing accessible applications and user interface toolkits for GNOME, in this work we also developed a number of assistive technologies – step 6 of the OAF, and a key task in proving that 3rd generation accessibility is a viable technique.

Key concepts of the GNOME accessibility effort that inform the OAF include:

1. Several expansions to the minimum set of things in an accessibility API[2]
2. A more fully defined theme mechanism – defining theme elements such as the focus rectangle thickness to help people with low vision locate the focused user interface element
3. Bridging multiple accessibility APIs down to a single, platform-wide API (the role of AT-SPI)
4. Handling the tricky problem of object embedding, so that an accessible application or portion of an application can be embedded in another application (e.g. the gecko

[2]See Chapter 3 for this minimum set

HTML rendering engine being embedded within a web browser or a help browser or an e-mail client)

5. Appropriate user interfaces for setting user theme preferences
6. Extending accessibility support to the login prompt, the screen unlock dialog
7. Providing additional APIs for assistive technologies to capture and inject keystrokes and mouse events
8. Providing a set of assistive technology support libraries (e.g. text-to-speech and braillee output)II

1.1.3. Windows 95 and later: IAccessible2

Around the time that Sun introduced the Java Accessibility API, Microsoft shipped the first release of the Microsoft Active Accessibility API or MSAA [41] Unlike the Java Accessibility API, MSAA didn't have a modular approach – all user interface elements exposed all API calls – and the API didn't provide a means to query all of the information needed by assistive technologies.

Late in 2006 [47], IBM developed an extension to the core MSAA accessible object – IAccessible. This extension was named IAccessible2 [24] This API is both very similar to the GNOME accessibility API (deriving directly from the OpenOffice.org underlying/internal accessibility implementation).

While there is little "new" of significance in this API to inform the OAF, over the past several years of use on the Windows platform the API has been refined, and so we reviewed this API as well for any potential additions to the API portion of the OAF.

1.1.4. Windows Vista and later: UI Automation

In 2005 in advance of it's release of Vista, Microsoft introduced their second accessibility API UI Automation. ThisI API incorporates two of the key lessons of the GNOME accessibility framework and the Java Accessibility API: a much more comprehensive and composable API (through the use of control patterns), and notably the notion that bridges (or proxies in their terminology) should be used to take API information from a variety of sources and present them all under a single unified API to assistive technologies.

To date, there have been no significant impacts on the OAF from UI Automation.

Separate from these accessibility APIs on Windows – UI Automation and IAccessible2 – the Windows platform was the first to have a rich, desktop-wide themeing mechanism. This kind of mechanism was discussed above in the GNOME accessibility section, but it is worth noting that most of the themeing innovations began with Windows 3.1 and then were expanded with Windows 95.

1.1.5. Macintosh OS X: Mac OS X Accessibility Protocol

Soon after Apple released OS X – their operating system re-write based on the UNIX Mach kernel – they began work on a 3rd generation accessibility API. The core concepts developed in the Java and GNOME accessibility APIs – and mirrored in UI Automation – are realized in the Macintosh accessibility API. These include an API that is comprehensive and extensible, implementable by custom user interface elements, and is composable (not every element implements every part of the API). It also enshrined the notion of bridges or proxies, to deal with the mixed environment of both Carbon and Cocoa applications.

OpenOffice.org on Macintosh exposes it's internal accessibility framework via the Mac OS X accessibility protocol – supporting use by the built-in VoiceOver screen reader – further proving bridging technology and also further demonstrating cross-platform accessibility.

Beyond such proofs, there is little "new" of significance in this API to inform the OAF.

1.1.6. Apple iOS accessibility architecture and included screen reader VoiceOver

The Apple iOS accessibility architecture is arguably the first 3rd generation accessibility API on a mobile platform – iPhone, iPod Touch, and iPad. Some of the AEGIS mobile prototypes are being developed for iOS, and as those prototypes are developed and mature, we will be feedback back what we learn into the OAF, to see whether there is anything unique in the iOS mobile accessibility environment that should enrich the OAF.

iOS comes with the built-in screen reader VoiceOver, which uses the iOS accessibility API that is implemented on the standard iOS user interface components to provide speech and braille output to those applications, and to web content via the built-in Safari web browser. Apple developed a set of multi-touch gestures for controlling VoiceOver and through VoiceOver of interacting with applications on the touch screen display. VoiceOver thereby provides a key facet of the "definition of accessibility" (see section 2.2.1. Step 1: Define what "accessible" means): the navigation/manipulation scheme for the user interface elements. This functionality is how Apple addressed a major accessibility challenge: how a blind person can manipulate a touch screen with no physically discernible buttons.

Apple has continued to improve the accessibility of iOS since publication of the original OAF deliverable in 2010. During that time, Apple has made to major release of iOS: iOS 5 and iOS 6 – each containing significant accessibility improvements. While most of those improvements were incremental new features in the built-in assistive technologies (e.g. support for turning VoiceOver or other built-in AT on/off with a triple click of the home button, adding vertical navigation support in VoiceOver, allowing header navigation in VoiceOver with web content from a connected keyboard via the letter 'h'), there are a few that have implications for the OAF, as they address new user interaction modalities.

These new modalities are:

- programmable vibration patterns associated with callerID for users with hearing impairments (also LED flash alerts)
- AssistiveTouch provides a a multi-step set of single finger gestures to users with some physical impairments who cannot make the multi-touch gestures (akin to StickyKeys, but for gestures)
- Guided Access for users with some cognitive disabilities that allows a clinician or assistant to limit the iOS device to only a single application, as well as removing some user interface options from that application.

Of these new iOS accessibility user interaction modalities, AssistiveTouch is already part of the original OAF (step #1 "Define Accessibility" - specifically "Defining a navigation / manipulation scheme" which described below). Programmable vibrations for incoming calls likewise doesn't add any new facets to the OAF itself, though it provides new functionality not previously available.

However, the Guided Access feature does raise a new idea not already encompassed by the OAF: that of limiting access to a subset of the user interface as an aid to individuals with some forms of cognitive impairments. This is a useful addition, and has been incorporated into OAF Step 5: The accessible application (or content) itself below.

1.1.7. Android release 1.6 and later: the Android Accessibility API and Google-provided TalkBack and Eyes-Free Shell

The Android accessibility architecture is arguably the second 3rd generation accessibility API on a mobile platform. The approach taken by Android is quite different from that of every other accessibility API on every other platform. Instead of annotating user interface elements with a standard set of accessibility API calls that can be used by assistive technologies, it uses a model that is perhaps derived from emacspeak – the talking emacs environment invented by TV Raman (who not coincidentally works on Google's accessibility effort). This model is realized in two free applications: TalkBack and the Eyes-Free Shell. User interface elements fire AccessibilityEvents when an accessibility service registers for them.

Successive releases of Android (through 2.3.3 "Gingerbread" on phones and 3.0 "Honeycomb" on tablets) have seen improvements to both TalkBack and the Eyes-Free Shell. These successive releases have also introduced new general APIs and interfaces relating to speech output, speech input, access to sensors, etc. CodeFactory also recently introduced a commercial screen reader for Android which also includes several purpose-built self-voicing applications.

Another facet of Android that can be leveraged for accessibility is the Android Input Method Service, which allows software virtual keyboards to insert keystrokes into any Android application.

In AEGIS we developed the assistive technology Tecla[49] for use on Android systems (and it is now available as a free download from the Android Play store[50]). In developing Tecla, we leveraged the Android Input Method Service to generate keystrokes both to input text as well as control the Android device.

Since our initial development of Tecla – and potentially influence by discussions we had with the Google accessibility engineering staff – the newest Android release version 4.1 "Jelly Bean" was released with a key new accessibility features. It now includes an Android accessibility API improvement called "Accessibility Focus". Thanks to this feature, an "accessibility focu"s can be moved through on-screen elements and navigation buttons using accessibility gestures, accessories, and other input (including by linear navigation through all controls). Furthermore, visual focus highlighting can be applied to the control regardless of whether the app developer tried to suppress focus highlights.

This new functionality closes two OAF gaps that had existed on Android (lack of full programmatic control of focus movement, and lack of a clear visual focus indicator).

1.1.8. RIM Blackberry accessibility API

Informed by previous implementations of 3rd generation accessibility approaches on desktop computing platforms (such as the Java Accessibility API), and developed in collaboration with Assistive Technology partners to ensure the needs of third party developers were met, Research In Motion (RIM) developed the BlackBerry Accessibility API [8] for the industry-leading BlackBerry smartphone platform. In addition, RIM updated the native user interface element library used by applications on BlackBerry smartphones to ensure the individual UI elements programmatically exposed their relevant information (such as Role, State, and Context) to Assistive Technology via the BlackBerry Accessibility API. To further help third party developers in mainstream and Assistive Technology companies to develop applications and solutions that are accessible by customers of varying abilities on BlackBerry smartphones, RIM released the BlackBerry Accessibility Development Guide [9] and a sample screen reader utility with the BlackBerry Java Development Environment (JDE) [10] tool chain.

To realize the capabilities of the BlackBerry Accessibility API and accessible native user interface element library in an application that directly benefits customers with disabilities, RIM implemented an embedded text-to-speech on the BlackBerry platform, implemented and exposed the Java Speech API (JSR) [35] for third party developers, and collaborated with Assistive Technology vendors HumanWare [23] (Canada) and CodeFactory [12] (Spain) to enable the development of the Oratio for BlackBerry screen reader [46] for use by blind, partially sighted and print-disabled customers.

While this work by RIM and the Assistive Technology community is ongoing, these efforts expended demonstrate that the 3rd Generation Accessibility approach is equally applicable on desktop and mobile phone devices. Sample Assistive Technology applications developed within AEGIS will further help to demonstrate the applicability of this collaborative and innovative approach to building accessible ICT.

1.1.9. Windows Phone 7

Microsoft released a complete rewrite of their mobile platform – Windows Phone 7 – which replaced the previous platform Windows Mobile 6.5. Existing commercial assistive technologies for Windows Mobile 6.5 devices won't work with Windows Phone 7 devices. While Microsoft includes some accessibility features – such as the contrast control to help users with some vision impairments, content zoom capabilities in their HTML rendering component, speech recognition for some common tasks, and text prediction that may also help some users with cognitive impairments. But there is no accessibility API defined for this platform. Windows Phone 7 is therefore still a 2nd generation platform for accessibility purposes, and further lacks the reverse-engineered 3rd party assistive technologies that were present in the previous version of the platform.

1.1.10. Windows 8

Microsoft recently released a new version of their desktop (and now table) operating system: Windows 8. Windows 8 has two user interface "modes" - the traditional desktop mode, and a new "Metro" interface that uses "tiles" and is designed for table/touch interaction. Windows 8 introduces many new accessibility features.[43] These include a number of built-in assistive technologies and tools for enabling them. It also carries forward Microsoft UI Automation into the touch interface.

While Windows 8 provides a number of accessibility improvements over Windows 7 – particularly around the built-in assistive technologies such as full screen magnification – it doesn't introduce anything new to the accessibility field overall. As such, there are no implications for the OAF.

1.2. Regulatory & Standards efforts informing the OAF

There is a lot of attention being paid worldwide to ICT accessibility – particular in both regulatory and standards contexts. In many of these instances, there is broad recognition that multiple different pieces of technology need to come together to successfully realize an accessible ICT system. The AEGIS project is actively participating in most of these efforts, and monitoring the others.

The following regulatory and standards efforts in one form or another reference various aspects of the Open Accessibility Framework. Through our participation in and monitoring of them, they also inform the ongoing development of the AEGIS OAF. Conversely, the AEGIS OAF insights inform our participation in these standards and regulatory efforts.

1.2.1. ISO/IEC 13066

ISO/IEC 13066 “Information Technology — Interoperability with Assistive Technology (AT)” is a projected multi-part standard or technical specification for accessibility APIs. Part 1 will cover “Requirements and recommendations for interoperability”, part 2 the Windows Automation Framework accessibility API, part 3 IAccessible2, part 4 the Linux / UNIX graphical environments accessibility API and part 6 the Java accessibility API.

Other accessibility considerations are covered by ISO/IEC TR 29138. [31]

Through collaboration with mainstream platform vendors and developers of several other accessibility APIs (Microsoft, IBM), as well as developers of assistive technologies, thelped further d[73]velop a rich and platform-neutral understanding of what the minimum set of information must be for a comprehensive accessibility API to support assistive technologies. These “minimum items” from 13066-1 are:

- Making information available about user interface elements
  - role, state(s), boundary, name, and description of the user interface element
  - current value and any minimum or maximum values, if the user interface element represents one of a range of values
  - text contents, text attributes, and the boundary of text rendered to the screen. concerning a number
  - the location of the user interface element in relation to other user interface elements
    - in a single data value, whether the user interface element is a label or data element, and the data or label that it is related to
    - in a table, the row and column that it is in, including headers of the row and column if present
    - in a hierarchical relationship, any parent containing the user interface element, and any children contained by the user interface element

- Available actions on user interface elements
  - programmatically expose a list of available actions on a user interface element and allow assistive technology to programmatically execute any of those actions.

- Focus and selection attributes of user interface elements
  - programmatically expose information necessary to track and modify:
    - focus
    - text insertion point (where applicable)
    - selection attributes of user interface elements

- Changes related to user interface elements

- programmatically expose notification of events relevant to user interactions, including but not limited to:
  - changes in the user interface element value
  - changes in the name of the user interface element
  - changes in the description of the user interface element
  - changes in the boundary of the user interface element

Thus far in the development of ISO/IEC 13066, we have not discovered anything that is not already within the OAF. We have, however, used early internal drafts of this document, as well as the work of the Open Desktop, to inform ISO/IEC 13066.

1.2.2. The TEITAC Report & the U.S. Access Board Advanced Notice of Proposed Rulemaking

The U.S. Access Board is in the process of refreshing the technical Accessibility Standards for ICT acquired by the U.S. Federal Government under Section 508 of the Federal Rehabilitation Act (commonly know as the "Section 508 Accessibility Standards"), as well as the technical Accessibility Guidelines which will be applied to all ICT under the jurisdiction of the Federal Communications Commission through Section 255 of the Telecommunications Act (commonly know as the "Section 255 Accessibility Guidelines"). Though this process is not yet complete, three published interim documents have been widely circulated toward that final document, and both of them serve a useful reference documents for the OAF.

The first document is a consensus report delivered by the Telecommunications and Electronic and Information Technology Advisory Committee (or TEITAC). This committee was formed by the U.S. Access Board and charged with delivering a report advising the Board on how the both the Section 508 Accessibility Standards, and the Section 255 Accessibility Guidelines, should be updated. Known as the "TEITAC Report" [55], it included a complete and comprehensive set of provision recommendations [54] (including a detailed set of proposed technical requirements [51]) that could apply to both ICT under Section 255 and also under Section 508.

The second document is the Advanced Notice of Proposed Rulemaking issued in 2010 (2010 ANPRM) from the U.S. Access Board – their "first draft" of a new set of provisions again to apply to ICT under both Section 255 and 508. Titled the "Draft Information and Communication (ICT) Standards and Guidelines",[57] it was issued in 2010 and draws heavily from the TEITAC report.

The third document is a second Advanced Notice of Proposed Rulemaking, issued in 2011 (2011 ANPRM). titled "Information and Communication Technology (ICT) Standards and Guidelines",[2] it is a very significant re-write of the previous ANPRM in 2010. This new draft trims the document significantly – it has 6 chapters compared to 10 in the 2010 ANPRM. Part of how it achieves this trimming down is by stating that the WCAG 2.0 A/AA success criteria should also be applied to non-web software (more on this below), with the Access Board stating that "it is straightforward to apply the WCAG 2.0 Success Criteria and Conformance Requirements to user interface components and content of platforms and applications."

These documents have had a significant impact on the OAF. Specifically, from the TEITAC Report: Section 3 "Requirements for User Interface and Electronic Content" [52] with

- provisions for themeing (3-D "User Preferences" and 3-E "Color Adjustment")

- the provisions which speak to keyboard operation (3-S “Keyboard Operation” and 3-SS “Visual Indication of Keyboard Shortcuts”)
- provisions which speak to programmatic information for assistive technologies (3-F “Non-text Objects”, 3-M “Reading Sequence”, 3-O “Information and Relationships”, 3-P “User Interface Components”, and 3-U “AT Interoperability)
- the provision speaking to a platform's responsibility for an accessibility API (3-V “Accessibility Services”)
- the provision that speaks to an assistive technology's responsibility for using a platform's accessibility API (3-VV “Assistive Technology”);

and all of the provisions in Section 7 “Additional Requirements for Authoring Tools”. [53]

From the Access Board 2010 ANPRM: Chapter 4 “Platforms, Applications, and Interactive Content” [65] with

- provisions for keyboard operation (item 404 “Keyboard Operation” [58])
- content structure (item 406 “Navigation” [59])
- themeing (item 409 “User Preferences” [60])
- platform support for assistive technologies (item 410 “Interoprability with Assistive Technologies” [61])
- the accessibility API information that must be provided to assistive technologies (item 411 “Compatible Technologies” [62])
- an assistive technology's responsibility for using a platform's accessibility API (item 412 “Assistive Technology Function” [63])
- the functions in support of accessibility that must be in an authoring tool (item 413 “Authoring Tools” [64])

and Chapter 5 “Electronic Documents” [66] with a collection of provisions enumerating the information that is needed for accessibility in electronic documents.

Notable in the 2011 ANPRM is that it explicitly allows the use of WCAG 2.0 AA in place of most of the provisions in Chapters 4 and 6, and all of Chapter 5.

1.2.3. EU Mandate 376

On December 7, 2005 the European Commission gave a mandate to the European Standards Setting Organisations to develop a solution for common requirements and conformance assessment for public procurement of accessible ICT. Known as EU Mandate 376 [18], it is being worked on by CEN, CENELEC, and ETSI in two stages. The first stage – an inventory of existing requirements and standards [17] – was completed by the ETSI Specialist Task Force 333 (STF 333).

The second stage – a set of accessibility requirements and technical specifications – is nearing completion, and will soon be (or perhaps has just been) released to the CEN/CENELEC/ETSI Joint Working Group, who will review it. This work is being carried out by the ETSI Specialist Task Force 416 (STF 416). After review, it will go out for National Standards Body voting.[40] Several public drafts of the requirements and technical specifications have been published, with the last published set released at the end of July 2012.[39] STF 416 is producing 3 documents:

- EN 301 549 Accessibility requirements for public procurement of ICT products and services in Europe
- TR 101 550 Documents relevant to accessibility requirements for public procurement of products and services in Europe
- TR 101 551 Guidelines on accessibility award Criteria for ICT products and services in Europe

Of these three documents, the EN contains the technical standards for ICT accessibility. The most recent drafts of these technical specifications have tracked fairly closely to the 2010 and 2011 ANPRMs from the U.S. Access Board. Most specifically, the most recent draft followed the lead of the 2011 ANPRM in seeking to apply WCAG 2.0 A/AA success criteria to non-web ICT.

The members of STF 416 found there were some challenges in applying WCAG 2.0 A/AA success criteria to non-web ICT, and several of those members joined a new effort in the W3C which emerged to provide such guidance (described below).

1.2.4. ANSI/HFES 200.2 / ISO 9241-171

ANSI/HFES 200 sets forth a set of standards for Human Factors Engineering of Software User Interfaces. ANSI/HFES 200.2 explicitly addresses ICT accessibility. It was edited by Sun Microsystems, published in 2005, and then revised and harmonized with ISO 9241-171, which became ISO 9241-171 "Ergonomics of human-system interaction -- Part 171: Guidance on software accessibility".

Both of these documents formed the basis for much of the work in the TEITAC report (including the sections cited above). Their influence on the OAF is through the TEITAC report.

1.2.5. WCAG / ATAG / UAAG

The Web Accessibility Initiative [71] of the World Wide Web Consortium (W3C) develops accessibility standards for the Internet and intranet. Three key areas for these standards are: web content (as addressed by their Web Content Accessibility Guidelines [70]), authoring tools for creating web content (as addressed by their Authoring Tools Accessibility Guidelines [67]), and web user agents which display web content (as addressed by their User Agent Accessibility Guidelines [69]).

All three sets of guidelines have had a very significant influence on the Open Accessibility Framework. The WCAG 2.0 guidelines were written in a very broad and general fashion, and as such are broadly applicable to all kinds of authored content. The ATAG 2.0 guidelines are still under active development. Like WCAG 2.0, they are broadly written and to a significant degree applicable to a broad array of authoring tools (though not of developer tools). The UAAG 2.0 guidelines are also under active development. While they contain many concepts applicable to the OAF for platforms generally, unlike WCAG 2.0 and ATAG 2.0, many of these guidelines are fairly specific to HTML content.

1.2.5.1. WAI-ARIA

In addition to the three broad areas of W3C WAI guidance described above, they are also developing a specific set of technical documents for Accessible Rich Internet Applications – the WAI ARIA document suite. Similar to TEITAC report provision 3-U and the Access Board Advanced Notice of Proposed Rulemaking item 412, WAI-ARIA enumerates the information that must be provided by applications so that assistive technologies can function – as a 3rd generation accessibility platform.

As such, WAI-ARIA has a significant influence on the Open Accessibility Framework.

1.2.5.2. W3C WCAG task force WCAT2ICT

The intention of the U.S. Access Board and also the Mandate 376 STF 416 to apply WCAG 2.0 A/AA success criteria to non-web ICT raised the concerns of a number of stakeholders. [13] One notable comment came from the U.S. Information Technology Industry Council, which stated:

> "In the case of applying WCAG to platforms, software, and electronic content of all forms, we believe the Access Board may have oversimplified the requirement. While WCAG 2.0 was designed to be technology neutral, as we mentioned, the designers did assume that the web content was being rendered by a web browser or other user agent. Advisory E207.2 states: "*...it is straightforward to apply the WCAG 2.0 Success Criteria and Conformance Requirements to user interface components and content of platforms and application.*" However, we have found many cases where it would be confusing for agencies and suppliers to determine whether or how to apply such success criteria to anything other than web pages." [32].

To address these concerns – which were also shared by many in the advocacy community as well as the U.S. Federal agencies who would be applying the Section 508 standard – stakeholders from the ICT industry, from the disability advocacy community, from U.S. Federal agencies, and from the Mandate 376 STF 416 worked with the W3C and WCAG Working Group to create the WCAG2ICT Task Force [73] whose job it would be to provide guidance on how to apply WCAG 2.0 to non-web ICT. As noted in the WCAG2ICT Work Statement: [72]

> The objective of WCAG2ICT Task Force is to develop documentation describing how to apply WCAG 2.0 and its principles, guidelines, and success criteria to non-Web Information and Communications Technologies (ICT). As part of this work, the Task Force will also review WCAG 2.0 Conformance in the context of how it might apply to non-web ICT.
>
> This Task Force was formed in response to suggestions that WCAG 2.0 could be applied to non-web ICT (Information and Communications Technology) that are not web content. It is not the intent of this Task force to judge as to whether WCAG 2.0 or any particular WCAG 2.0 provisions should be used with non-web content, but rather to comment on the meaning of the WCAG 2.0 guidelines, principles, success criteria and conformance requirements and how they should be understood if they were to be applied to non-web ICT.

The AEGIS Technical Manager Peter Korn was one of the key proponents of the creation of this Task Force, and is one of the co-editors of their planned guidance document. Their first draft of that guidance document, "Applying WCAG 2.0 to Non-Web Information and Communications Technologies" [5] was released for public review in July 2012. Another draft is expected in November. The work of this task force has proven to be very useful to Mandate 376 STF 413, and is reflected in their upcoming draft of EN 301 549.

2. THE OPEN ACCESSIBILITY FRAMEWORK

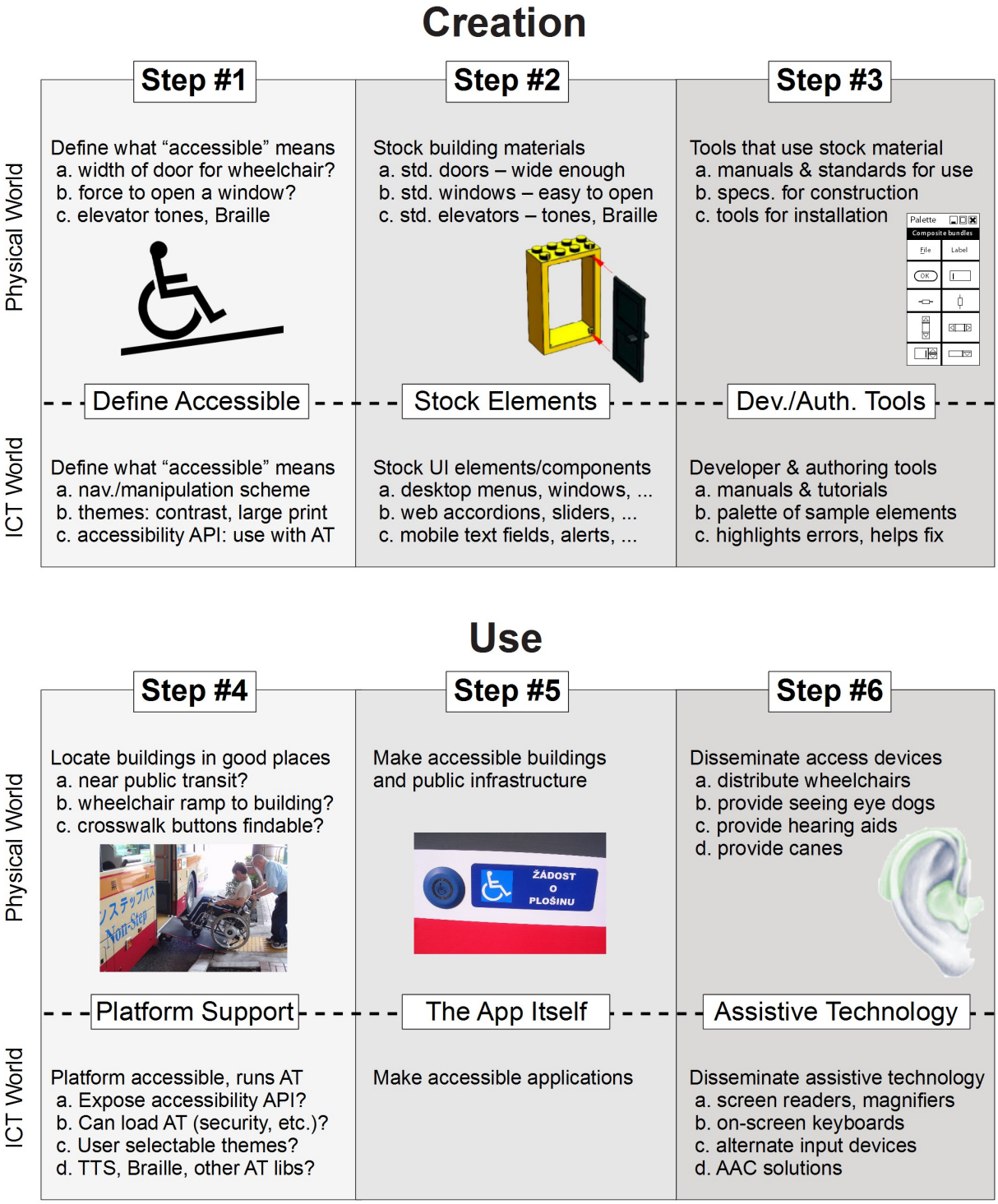
2.1. Basis for the OAF – accessibility in the physical world

Making the physical world – also known as the "built environment" – accessible involves paying attention not only to how things are created, but also how things are used. It is important to create wheelchair ramps, but they are only useful if people have wheelchairs!

And it isn't enough to simply build ramps – we need to know how steep to make them, how wide to make them. We need to *define* what the maximum ramp angle is.

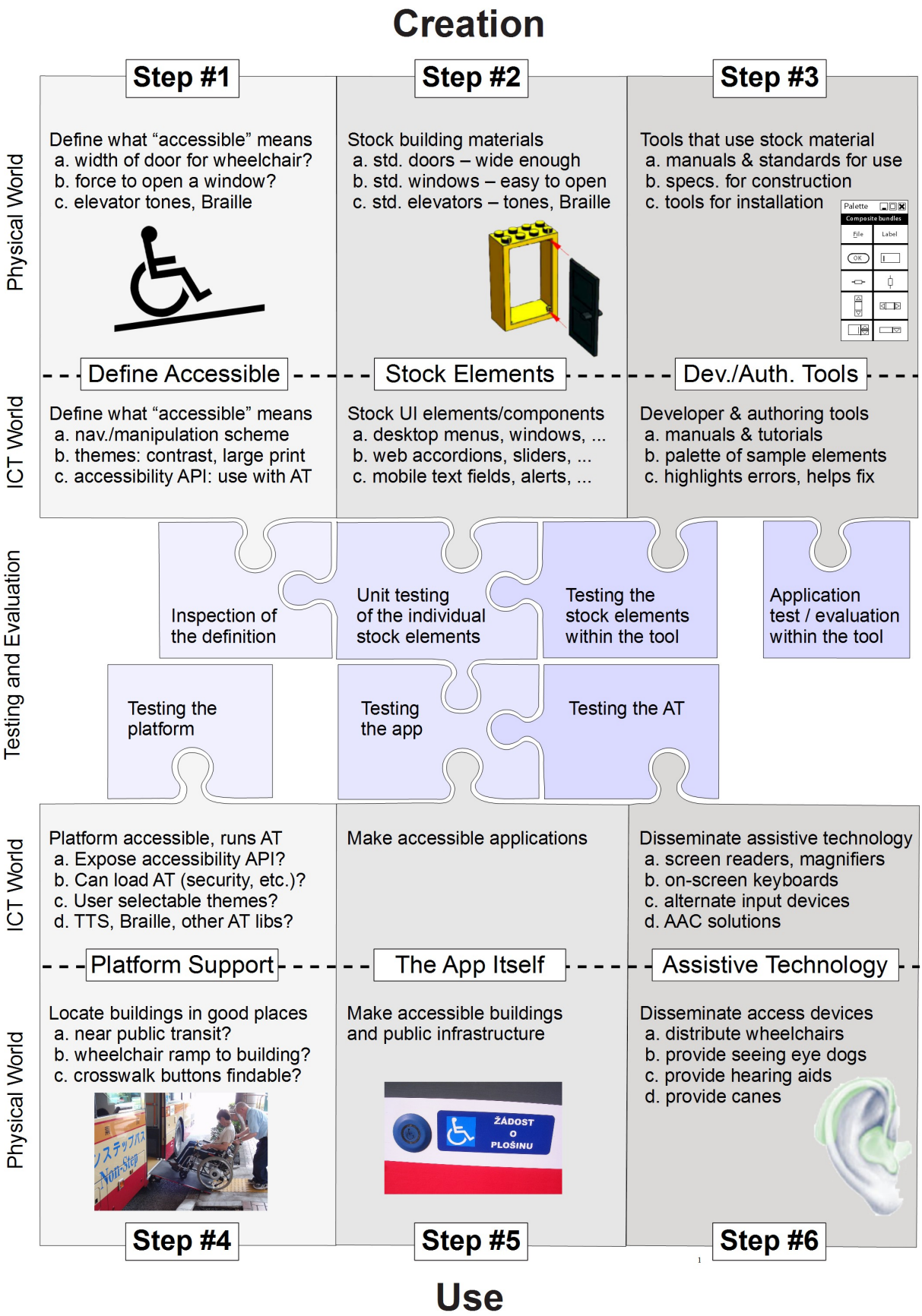
The same is true in the ICT world. It is important to create the "accessible on-ramps to the web", to software in general, but these are only useful if people have the assistive technologies they need to use these on-ramps. And in order to build these on-ramps – we need to know how to build them. And both with ICT and the physical world, tools play an important role in helping us create things with these ramps.

We put all of this together in the figure below – showing the 3 "Creation" steps and the 3 "Use" steps of building both an accessible physical world and an accessible ICT world.



Drawing 1: The six steps of creating an accessible world - physical or ICT

In addition to these six steps, it is often useful to evaluate and test the various steps as they are taken. The drawing below illustrates the role of testing in these 6 steps. The ICT portion of this next drawing is the Open Accessibility Framework.



Drawing 2: The six steps, including testing and evaluation

The following two sections describe in detail the three steps of the creation domain, and of the use domain of the Open Accessibility Framework. The section after that describes testing and evaluation

2.2. Creation domain

2.2.1. Step 1: Define what "accessible" means

Here we define the various aspects of what it means to be accessible, for a given technology. This includes:

- **Defining a navigation / manipulation scheme** – typically (but not necessarily) a keyboard navigation scheme. This must be done for all user interface elements. For example, a common desktop navigation scheme for within a window might define the TAB key on the keyboard as the primary way to navigate between user interface elements, with arrow key for navigation between groups of items (check boxes in a group, tabs in a tab paned), and F6 between "panes". This is further refined for navigation within complex user interface elements (e.g. arrows, CONTROL-arrows, home/end for navigation within a text field, adding SHIFT for selection). And manipulation is also a component of this: SPACE to toggle checkboxes on/off, left/right arrow to expand/close tree nodes, etc.

  This navigation / manipulation scheme may be a de facto or de jure standard for a given technology (e.g. the Java Swing Look and Feel Guidelines define the keyboard navigation scheme for the Java Foundation Classes; ACE Centre documents the Microsoft Windows XP keyboard operation [22]).

  Note: the navigation / manipulation scheme need not be based on physical keys that a user can distinguish tactilely and which depress. Touch based devices such as the current crop of phones and tablets can still define such a scheme. For example, Apple has defined a navigation and manipulation scheme for iOS that is activated when the VoiceOver screen reader is running. A swipe across the screen with a single finger becomes the equivalent of the TAB/shift-TAB or left/right arrow key of desktop GUIs: it moves the "VoiceOver focus" to the next/previous manipulable item. Once "VoiceOver focused", a double-tap activates that item (equivalent to a single tap on that item hat VoiceOver not been running). Care must be taken in such situations to ensure that all functionality of the UI are available via accessible gestures that (a) don't require vision, and (b) which can be generated by someone who may not have hands (e.g. generated via software and/or via a connected peripheral).

- **Defining theme behaviour for vision impairments** – typically things like high-contrast / large-print for a desktop, CSS for HTML content, etc. This may be a de facto or de jure standard for a given technology (e.g. CSS is a W3C standard).

- **Defining an accessibility API** for interoperability with assistive technologies. Note that depending upon the technology in question, this may be an API that is directly used by assistive technologies (e.g. if we are defining a platform like Macintosh OS X), or it may be an "intermediate layer", or a "platform on a platform" (e.g. the Java Accessibility API which is realized differently on GNOME vs. Macintosh vs. Windows; or WAI-ARIA which is realized different by different browsers on different platforms).
  This may be a de facto or de jure standard for a given technology (e.g. WAI ARIA is a candidate W3C standard). Though neither ISO/IEC 13066, nor the refreshed Section

508 Guidelines are final, drafts of both documents provide a minimum list of things that any accessibility API must do. This document also contains such a list (see Chapter 3 below).

2.2.2. Step 2: Make stock user interface elements that implement "accessible"

Here we build re-usable user-interface elements that implement the definition of "accessible" above, for a given technology. Note that this step isn't necessary for all use of all technologies. For example, there are still quite a few people who create web pages by editing the raw HTML (rather than using a user interface element set technology like Java Server Faces).

These re-usable user-interface elements (or element sets or toolkits) must implement all aspects of the definition of accessible from step 1:

- **Implementing a navigation / manipulation scheme** – typically (but not necessarily) a keyboard navigation scheme. Whatever is defined – on a user-case by use-case, element by element basis, it must be implemented by the stock user interface elements. Where the elements are cross-platform (e.g. the Java Swing user interface toolkit), there should be a way for those UI elementss to behave appropriate to the underlying platform (e.g. the GTK+ Look and Feel for Swing uses the GTK+ keyboard navigation behaviour when running on the Open Desktop, while the Windows Look and Feel for Swing uses the Windows keyboard navigation behaviour when running on Windows).

- **Implementing theme behaviour for vision impairments.** Again, the user interface elements or toolkits must realize the platform accessibility definition – or that of the underlying platform in a cross-platform environment (e.g. the Java Swing user interface toolkit, wherein the GTK+ Look and Feel for Swing uses the GTK+ theme and behaviour when running on the Open Desktop, while the Windows Look and Feel for Swing uses the Windows theme and behaviour when running on Windows).

- **Implementing the accessibility API** for interoperability with assistive technologies. Whether the user-interface elements are tied to a specific platform, or are cross-platform (e.g. Java), the accessibility API implementation is the same: that of the platform in question (e.g. Java). It becomes the responsibility of the platform (in step 4 below) to ensure that a connection is made between applications using these user-interface elements and any assistive technologies.

2.2.3. Step 3: Developer tools for creating accessible apps (with stock elements); authoring tools for creating accessible documents

Both developer tools and authoring tools can provide significant assistance to the task of creating accessible applications and documents. Common to both is the ability to provide examples and templates of accessible applications and documents respectively. Also common to both is the ability to provide prompts for recognized and needed accessibility metadata (e.g. the alternate text for an image – in a document or displayed in an application), and to perform checks to see if such metadata is missing from the known places where it should go.

In addition, developer tools may have specific knowledge of one or more of the user interface element sets used to create applications with the tool. Such knowledge can not only improve the nature of any prompting for accessibility metadata, but can also be used to enhance any

GUI design and layout functionality in the developer tool (for example, a GUI designer might include pre-labeled text fields as a combined UI elements on an element palette; or it might provide additional support for connecting labels to the fields they label).

Developer tools may also include accessibility analysis reports, disability visualization tools, and guided accessibility error fixing wizards.

Authoring tools may also include additional support (beyond what is common to developer tools) to aid the author in creating accessible documents. They may include sets of pre-defined styles containing semantic meaning (e.g. "emphasis" instead of "boldface", "Heading 1" instead of simply "18 point bold"). They may automatically notice the likely creation of structure and prompt the user to use that structure (e.g. noticing a series of lines starting with a space & a dash and prompting the user to turn that automatically into an unordered list). Authoring tools may also include accessibility analysis reports, disability visualization tools, and guided accessibility error fixing wizards. And when they convert (or export) to a variety of different formats, preserve accessibility information from the source to the destination format.

2.3. Use domain

2.3.1. Step 4: Accessibility support in the platform (and also the platform-on-a-platform case)

As recognized by the regulatory & standards efforts cited in Chapter 1, there are several facets to platform support for accessibility. These facets are all realizations of the "defining" aspects in step 1:

- **Implementing a navigation / manipulation scheme** – typically (but not necessarily) a keyboard navigation scheme. While much of this work is done in the step 2 user interface elements, the platform has a key role to play here. This includes implementing overall or platform-wide navigation / manipulation gestures (e.g. the ALT-TAB keyboard gesture for switching between applications on Windows, Macintosh, and the Open Desktop; Alt-F1 to bring up the first desktop menu on the Open Desktop – and Ctrl-ESC to bring up the Start menu on Windows), and mediating navigation / manipulation gestures between the underlying platform and the platform-on-a-platform scenario (e.g. keyboard gestures within a web browser vs. keyboard gestures for the desktop that the web browser is running on).

- **Implementing theme behaviour for vision impairments.** For a platform, the key theme responsibility is to enable users to find and choose one or more settings that meet their needs (or to define a new one). The platform must also have a mechanism by which applications can pick up the fact that the setting(s) changed, so that they can update themselves – ideally dynamically.
  In the case where the platform is also an application – sitting on top of an underlying operating system – then it has the additional responsibility of exposing/translating those underlying operating system theme settings to applications within it. For example, the Java SE platform provides APIs for determining what the theme settings are of the underlying operating system. The GTK+ Look and Feel uses these APIs when running on a GNOME desktop to mirror the GNOME theme settings. Likewise the Windows Look and Feel uses the same APIs when running on a Windows desktop to mirror the Windows theme settings.

- **Implementing the accessibility API** for interoperability with assistive technologies – which is primarily about the inter-process communication plumbing necessary for applications and assistive technologies to communicate though it. On the platform, this typically includes a discovery mechanism (a way for assistive technologies to discover what applications are running, and to initiate API-based communication with them), as well as various other services (which will vary by the specific nature of the accessibility API and platform). For the platform-on-a-platform situation (e.g. a web browser with the WAI-ARIA API, or the Java platform with the Java accessibility API), there is an additional, translation layer needed. This is sometimes called a bridge (e.g. the Java Access Bridge for Windows or UNIX), or it may be contained entirely within the runtime for that platform (e.g. the IAccessible2 implementation inside Firefox for Windows, as compared to the ATK implementation inside Firefox for UNIX – in each case exposing WAI-ARIA API information from Rich Internet Applications to assistive technologies running on the underlying platform).

2.3.2. Step 5: The accessible application (or content) itself

Applications (and specifically the application developers who create them) have the responsibility for being accessible. This is the case whether or not the application uses solely user-interface elements (from user-interface element sets) that fully implement the definition of accessibility for the platform. And also whether or not the application is developed using developer tools that help the developer create accessible applications (though certainly doing both of those things can help tremendously).

Where the application utilizes accessible user-interface elements, there is still often some additional work that needs to be done. Developers need to ensure that all editable text fields have labels, and has to ensure all images and icons have names associated with them. The specific additional work will vary depending upon the details of the user-interface elements and the accessibility API; but there is almost always some additional work – in addition to testing! And of course, other accessibility rules apply that aren't formally part of the Open Accessibility Framework – things like ensuring colour is not the only means used to differentiate parts of the user-interface or to convey meaning.

Where the application creates custom user-interface elements that are extensions or subclasses of existing accessible user-interface elements, it is highly likely that additional work will be needed to make those custom user-interface elements accessible. This includes things like setting the appropriate Role and States for the custom user-interface element, implementing various accessibility API calls, etc. Keyboard and other navigation mechanisms may need to be developed and implemented, as well as care taken for theme support.

Where the application creates custom user-interface elements from scratch, then all of the accessibility support work must be done by the application: implementing the accessibility API, implementing the navigation & manipulation scheme, and of course theme support.

Documents (and specifically again, document authors) likewise have the responsibility of being accessible. Not only should all accessibility metadata be present (e.g. alternate text for graphics), but proper user of document structure is critical. Text should be marked up for meaning, not just for visual style and layout (e.g. a paragraph is a “Heading”, and not simply “18 point bold”). Layout should never be the sole indicator of meaning (e.g. using a series of spaces instead of indentation, using a set of numbered paragraph instead of a “list”, using TABs to create columns of text [with a boldface row at the top] instead of using a table with a header row). Authors should correctly set the language of a document and of any sentences, quotes or phrases that are in a different language than the immediately surrounding text. When exporting to another format, they should use settings that also export the accessibility

features of the content, for example, checking the "tagged PDF" option when exporting to PDF.

An in-depth analysis of document authoring support is being developed as part of the Accessible Digital Office Document Project [27] at AEGIS consortium member the Inclusive Design Research Centre.

As there are far more authors in the world than programmers – and no less author training compared to programmer training (e.g. no University degree offered) – it is all the more important to provide documentation and authoring tool support to authors, to help them with their discharge their responsibility to create accessible documents (hence step 3 above).

One notable addition for OAF Step #5 comes from developments in iOS version 6 – the idea that some users may benefit from having the set of options exposed by applications limited in some fashion (or that the set of applications they have access to is limited). This idea is something we discuss in more detail in section 6.1 "Future OAF research directions".

2.3.3. Step 6: The assistive technology and AT support libraries

Finally, assistive technologies have the responsibility of re-presenting the user interface of accessible applications to users with one or more significant disabilities. This re-presentation might be using text-to-speech or braille in place of the screen for someone who cannot see. It might be turning the screen into a magnifying lens – perhaps with a variety of colour enhancements as well – for someone whose vision is poor. It might be through presenting a replacement to the keyboard, such as a switch-driven on-screen keyboard, or through an alternate text entry system driven by eye movement. It might be through a voice recognition system, enabling someone to completely interact with the computer through using only their voice.

Assistive technologies don't need to be "platform-wide". Enhancements to specific applications – such as a plug-in to Firefox to specifically enhance web page rendering for vision impairments, or a plug-in to OpenOffice.org to help people with cognitive impairments create or understand documents – are also considered assistive technologies. Because they aren't "platform-wide", they may not make use of a formalized "accessibility API" to obtain and re-present information. But they should still be considered "3rd generation" AT if they use supported APIs (and eschew reverse engineering techniques) to provide or enhance accessibility.

Also included in this logical step of the OAF are assistive technology support libraries: things like a text-to-speech engine, or a Grade 2 braille translation library.

2.4. Testing and evaluation of the six steps

In addition to these six components of the Open Accessibility Framework, there may be any number of developer / designer / deployer / validator aids inserted at various points within these six steps to help ensure that the OAF is being properly followed – that the technology(ies) in question are being used properly in order to give an accessible result. These aids include:

- Document validators (such as HTML and ODF validation tools)
- Accessibility standards test tools (such as any of the commercial WCAG test tools)
- Impairment simulators (such as DIAS from the ACCESSIBLE project, or …)
- API test tools (such as Accerciser or AccProbe)

These may be stand-alone tools, or plug-ins to other programs (such as an IDE plug-in). Or they may simply be part of a developer or authoring tool, and may be part of or an adjunct to the developer/author work flow.

Note that in addition to such tests/aids in each of the steps, developments in later steps often serve themselves as tests of the earlier steps. For example, failure of an application (step #5) may be due to failings in the tool that built it (step #3), in the stock elements that comprise it (step #2), or in the definition of accessibility (step #1). Alternately, the application may be fine by the user is nonetheless unable to use it because of problems with the assistive technology (step #6).

2.4.1. Testing/evaluating the definition – step #1

There is no tool or automated means of testing or evaluating the definition of accessibility. This can only be a manual step,

2.4.2. Testing/evaluating the stock components – step #2

Stock elements can be tested individually – unit testing – as well as with any tool that checks the implementation of the accessibility API (the accessibility definition). Such tests might be part of a developer tool like an Integrated Development Environment (IDE).

2.4.3. Testing/evaluating the developer/authoring tool – step #3

A developer or authoring tool is also an application. And like any application (step #5), it can be tested for accessibility. But as a developer or authoring tool, its main role in the OAF is to aid developers and authors in creating accessible applications (generally through the use of stock components that implement the definition of accessibility for the platform). Therefore, testing and evaluation become a feature or function of the developer/authoring tool itself, wherein the tool helps the author or developer evaluate the accessibility of their application (or document), and helps them add any accessibility metadata (for example, ALT text for images, linking labels to the things they label, etc.). It is also possible to include visualization and simulation features – illustrating to a developer or author how their application or document would appear to people with a variety of disabilities. Of course such functions, and many of the other functions that may be built into developer/authoring tools, might also be stand-alone applications.

2.4.4. Testing/evaluating the platform – step #4

Generally the way platforms are tested is by running applications on them. In the case of accessibility and the OAF, this means running not only general applications, but those specialized applications which are assistive technologies – and making sure that the two are properly communicating with each other. Likewise testing to ensure that information about any specialized themes for vision impairments are properly communicated to applications. In come cases, specialized test applications (and "pseudo assistive technologies") may be built to specifically instrument such tests. And if there are specific platform features for accessibility – such as text-to-speech, magnification, and braille libraries – such specially instrumented applications may test those features explicitly.

2.4.5. Testing/evaluating the application – step #5

Just as with platforms, testing an application for accessibility is generally done by using it with an assistive technology, by exercising various accessibility features (such as keyboard operation and theme support). And just as with platforms, specially instrumented test applications may be used – for example to test the accessibility API implementation of the

application, and otherwise exercise the application in the same way that an assistive technology would use it.

2.4.6. Testing/evaluating the assistive technology – step #6

As the final link in the accessibility chain with the user, testing of the AT is generally best done as user testing. API logging tools may be helpful in tracking exactly what API calls are being made by the AT, in which situations, but this far this has rarely been done in practice.

2.5. Additional topics related to the OAF

There are a number of techniques – both novel and longstanding – that are broadly related to the OAF.

In just the last few years multi-touch user interfaces have moved from the research lab into the mainstream. Whether based on physical touch via pressure or capacitance, or through video motion capture, gesture driven interfaces are now mainstream. And these interfaces commonly involve multiple simultaneous fingers or both hands in the gestures.

Though these interfaces are novel, and companies like Apple have started building assistive technologies to work in and with these interfaces, the basic OAF principles remain the same in these environments. To wit:

- There must still be a "definition of accessibility" that includes an accessible navigation/manipulation scheme, theme mechanism, and accessibility API.
- There are still stock user interface components that make up the (multi-touch) gesture user interfaces. These stock user interface components should respect the accessible navigation/manipulation scheme, theme definition, and properly expose the accessibility API. Note: as Apple has demonstrated, the accessible navigation/manipulation scheme need not be the "default" one, so long as a user can enable it (as is the case with iOS, whose blind user navigation/manipulation scheme is invoked via VoiceOver).
- Developer tools remain important, to help developers ensure their applications properly use the stock (or custom) user interface components and thereby that their applications respect the navigation/manipulation scheme, theme definition, and properly expose the accessibility API.
- Platform support remains important.
- Accessible applications are needed.
- Assistive technologies are needed.

Multi-touch user interfaces are introducing a number of novel ways of doing things that have been done with a mouse in desktop GUIs. For example, pinching with two to shrink/enlarge images is a multi-touch gesture for the task that graphics programs on the desktop would have the user click on the image, then click and drag an affordance that appears at one of the edges/corners of the image to shrink/enlarge it). Similarly swiping with a single finger is a gesture for the task that on desktops would be scrolling in one fashion or another.

And just as desktop GUIs have developed keyboard navigation/manipulations for mouse-driven operations, similar accessible navigation/manipulation mechanisms are needed for multi-touch gesture-driven operations. Apple has done this for a number of gestures used by blind users of iOS when using VoiceOver. And Apple's announced iOS version 5 is

introducing ways that external hardware devices can generate a broad range of gestures from those external devices.

In summary: the OAF is completely relevant to multi-touch/gesture user interfaces, and simply needs to be applied to the specifics of the multi-touch/gesture platform.

2.5.1. Self-voicing (and other self-accessible) applications

Since the beginning of ICT adaptation for people with disabilities, there have been two broad approaches to ICT accessibility: (1) system-wide assistive technologies that enable users with disabilities to use mainstream applications, and (2) purpose-built applications optimized for use by one or more specific disabilities. The OAF is concerned with the first approach, while the second approach can be termed "self-accessible" applications, of which the most common is "self-voicing" applications for the blind or visually impaired.

Self-accessible applications don't, by their nature, fit into the OAF. For example, an audio-only e-mail application custom built for a blind person running on a GUI platform isn't an application that is using the stock GUI components. Nor is it an assistive technology that is utilizing an accessibility API to make other applications accessible. Potentially very useful and powerful for the blind, such a self-voicing e-mail application is totally inaccessible to a deaf person.

However, self-accessible applications can *also* be GUI applications that utilize user interface components that implement the definition of accessibility and work with assistive technologies. Examples include most desktop software DAISY players, and the Arkenstone/Freedom Scientific OpenBook scanning software. These applications use stock user interface components and are fully visual applications – that also implement platform accessibility APIs and implement the keyboard navigation/manipulation definition of the platform. They work well with assistive technologies like screen readers. But they are also self-voicing applications that can be used by blind individuals without the need for a screen reader.

Thus the characteristic of self-accessibility doesn't make an application part of or apart from the OAF. Note: this should not be taken as an argument against self-accessible applications. Such applications can – and often do – a better job at providing an efficient and productive user interface to a specific task than the combination of the mainstream accessible app when used with an assistive technology. And we continue to see their use today – in such applications as the suite of self-voicing Android apps that are included with Code Factory's Android screen reader, the Eyes-Free Shell for Android from Google, and the recently released ZoomContacts low-vision contact manager for iPad from Ai Squared.

2.5.2. Virtual Buffers in screen readers with web browsers

The most popular screen reader on the most popular platform when working with the most popular browser – JAWS on Windows with IE – extracts the web content from the browser DOM into a virtual buffer, with its own navigation mechanism rather than obtaining the content via a platform accessibility API. This approach has proven to be a very effective technique for many blind users and a broad range of web content.

It is not, however, a technique that is within the OAF. At least not as it has been done by JAWS with IE on Windows. The key reason why it is not part of the OAF is because the API that IE exposes for DOM access is not a platform API (whether tagged explicitly as an "accessibility API" or no). Other applications don't implement that API. It is thus not part of the platform's definition of accessibility. It is instead a programming contract available only from one application – IE. Another place that this implementation falls short of the OAF is that it is not sufficiently comprehensive: it doesn't expose bounding rectangle information of

the DOM objects and the text within text objects (see section 3.1. below for a discussion of the minimum information that any accessibility API should convey about user interface elements). Specifically this means that a screen magnifier cannot rely on the IE DOM exposing API to provide low-vision users access to web content.

However, just as with self-accessible applications (discussed above in section 2.5.2.) just because a technique isn't part of the OAF doens't mean it is a "bad" or "disapproved" technique. The OAF is a holistic approach to ICT accessibility, but not the only approach – nor the only effective one.

2.5.3. Adapting the OAF over time

The OAF principles, while well established and proven on the Open Desktop, are a set of living principles. However well they stand the test of time and user interaction advances – such as multi-touch gesture interfaces – there is always the potential for growth as new insights learned from making these interfaces accessible mean we should expand aspects of the OAF. Particularly likely is the potential for growth as new assistive technologies are developed, which may demonstrate places where the accessibility API should be extended[3].

For this reason we have published the OAF on the Open Accessibility Everywhere Group (OAEG) website http://www.oaeg.org. We have also added a feedback form therein, in order to allow to user interface and accessibility experts not only learn from the OAF material at that site, but also contribute to it. The OAF is also part of the Wikipedia entry on computer accessibility.[28]

In OAEG section on the OAF [29], we have tried to cluster the accessibility items of each type of the most widely known platforms in desktop, mobile and web across the 6 steps of OAF, while, whenever applicable, the AEGIS project contributions are highlighted. When it is applicable, a hyperlink leads the visitor to the official pages that each solution (from AEGIS or not) is being maintained, whereas, whenever applicable, the visitor may follow it to link to the relevant semantics of the solution, as this have been implemented in the context of AEGIS ontology.

With the close of the AEGIS project, OAEG will be maintained by the ETNA project for further sustainability.[30]

3. THE ACCESSIBILITY API IN DETAIL

Accessibility APIs play a central role in the Open Accessibility Framework, and so deserve a bit further treatment. As evidenced by the references to the accessibility API in ISO 13066, and to accessibility services in HFES 200 / ISO 9241-171 and the draft Section 508 standard (as drawn from the TEITAC report and the ANPRM), there is a growing consensus of what information needs to be provided by platforms to support all currently known & understood assistive technology needs. These services may be provided through an API with the label "accessibility", or by some other supported API or service(s) from the platform. However they are provided, we will call them here aspects of the "accessibility API".

[3] A specific example under active discussion relates to HTML 5 video playback and the desire to have extended descriptions spoken by an assistive technology, with the AT telling the video playback to pause while it voices the extended description, and then resume when finished.

3.1. User Interface Element Information

A platform accessibility API must convey at least the following information to assistive technologies:

- Convey the ***role***, ***state(s)***, ***boundary***, ***name***, and ***description*** of every user interface element
- Convey the ***row*** and ***column*** an element is in, and the ***headers*** for the row and column for that user interface element, if it is in a table that has row or column headers
- Convey the ***current value*** and any ***minimum*** or ***maximum values***, if the user interface element represents one of a range of values
- Convey as appropriate any of the following relationships this user interface element has with one or more others:
  - As a ***label*** for another user interface element, or being ***labelled*** by another user interface element
  - As a ***member of a group*** of user interface elements (such as a radio button group)
  - As a description for another user interface element (typically non-text content), or being described by another user interface element. (Description here means a text alternative that is too long to be presented by default and thus supplements the primary text alternative or label.)
- Convey the ***parent or containing*** user interface element, and any ***children*** user interface elements
- Convey the ***text contents***, ***text attributes***, the ***text insertion caret location***, and the ***boundary of text*** rendered to the screen
- Convey any ***keyboard short cuts*** and implicit designators associated with a user interface element
- Convey the language of the user interface (for the benefit of voice recognition software).

3.2. Event Information

A platform accessibility API must inform assistive technologies when any of a series of events occur to or for a user interface element:

- Changes in the ***role***, ***state(s)***, ***boundary***, ***name***, and ***description*** of the user interface element
- Changes in the user interface element ***value***
- Changes to the ***text contents***, ***text attributes***, or the ***text insertion caret location*** for editable text in a any user interface element

3.3. Programmatic Invocation of Actions on User Interface Elements

A platform accessibility API must allow assistive technologies to programmatically manipulate the user interface as follows:

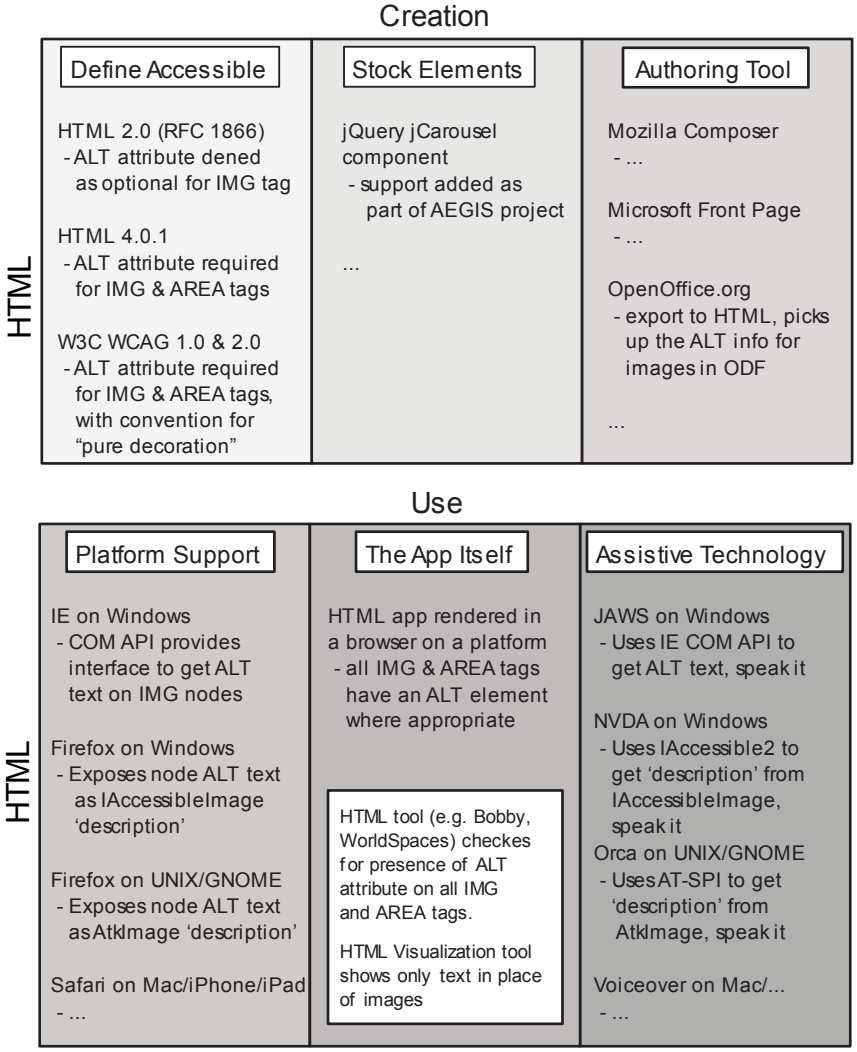
- Provide an ***enumeration of all actions*** available for all user interface elements
- Provide the ability to programmatically ***invoke those action***(s)
- Provide an ***enumeration of all selectable children*** user-interface elements
- Provide the ability to ***select and de-select selectable children*** user-interface elements
- Provide the ability to ***navigate between user-interface*** elements (commonly this will be ***keyboard focus*** on systems with a keyboard)
- Provide the ability to ***place the text insertion caret*** to all valid locations within editable text
- Provide the ability to ***set new values*** for user-interface elements that represent one of a range of values

4. HOW THE OAF IS USED BY VARIOUS ASSISTIVE TECHNOLOGIES, AND THEREBY FOR VARIOUS USERS

A mapping of OAF components (mostly the accessibility API) and AT uses / user use cases is contained in the attached Appendix.

To help illustrate how the Open Accessibility Framework is used in practice, the following diagrams use the well understood requirement of associating text with an image or icon so that blind and low vision users of screen readers can have that text spoken or rendered in refreshable braille to them. These diagrams illustrate the six components of the OAF for: HTML, ODF, and Java Swing. By way of comparison, we also attempt to apply the OAF to the 2nd generation accessibility approach of early MS-Windows and early Macintosh, highlighting the accessibility problems that result.

4.1. The OAF as applied to images in HTML



Drawing 3: The six components of the OAF, as applied to image accessibility in HTML

4.1.1. Define Accessible

The HTML specification defines how to associate text to an image – via the ALT attribute to the IMG tag. Optional in HTML 2.0 (RFC 1866), this was required by HTML 4.0.1, and likewise part of the W3C WCAG 1.0 and 2.0 specifications (with a specified convention for images that are pure decorations). There is no specification for how IMG elements theme (as compared to SVG which has an implied large-print specification). And as they are static images without user interaction (unless they are part of an A element and thus a link), there is no specified navigation/manipulation scheme.

4.1.2. Stock Elements

The jQuery jCarousel user-interface element presents a user interface to a set of images. As part of the AEGIS contributions to jQuery accessibility (Work Package 3.2 - "Open web application component sets for accessible rich web applications"), this user-interface element now provides a way to associate text with images, exposing that as the ALT attribute of the IMG tags within the jCarousel.

4.1.3. Authoring Tool / Developer Tool

When creating web pages, HTML editors such as Mozilla Composer or document editors which can export to HTML such as OpenOffice.org provide an option to associate text with images, which then get encoded as the ALT element of the IMG tag. Depending upon the authoring tool, author assistance may go further – prompting the author to provide the text each time an image is added to the page, and/or as part of an "accessibility check" performed either on request or at one or more specific places in the work flow of creating the web page. Also – again depending upon the authoring tool – such assistance may be provided as a built-in feature of the tool, or as a separately installable "plug-in".

Web pages may also be created in a developer tool – such as NetBeans or Eclipse or JDeveloper. Such developer tools (also commonly known as Integrated Development Environments or IDEs) may have specific knowledge of certain user-interface element sets (such as Java Swing or jQuery), and therefore provide accessibility assistance to developers who are using those user-interface elements to create web pages / web applications. And particularly with IDEs, such functionality may be provided by a separately installable "plug-in".

While not noted in the drawing above, there is a specification for HTML authoring tools and what they should do to support accessibility – the W3C Authoring Tool Accessibility Guidelines. [68] These guidelines specify in detail what HTML authoring tools should do to support accessibility (and two of the Editors of the draft ATAG v2.0 guidelines are from an AEGIS consortium member).

4.1.4. Platform Support

For HTML, the platform is the web browser (or more formally, the "User Agent"). Because this is a "platform-on-a-platform" situation (the browser runs on top of an underlying operating system), this platform must not only extract accessibility information from the HTML – in this case the text associated with images – but it must also provide it to assistive technologies through the underlying platform's accessibility framework, just as if the user agent was itself an application (because it is!). It likewise must mediate theme support, and navigation / manipulation gestures.

Specifically for the ALT element of the IMG tag, the browser must expose that user-interface element along side the other user-interface elements of the web page, and provide the text when the appropriate accessibility API call is made on that user-interface element. For example (and as shown in Drawing 3 above), Firefox on Windows would expose the IMG as an IAccessibleImage object, with the ALT text being the 'description' field of the IAccessibleImage object. By comparison, Firefox on UNIX/GNOME would expose the IMG as an AtkImage object (with ALT text as the 'description' field). Similar things occur with other browsers such as IE on Windows, Safari on Macintosh/iPhone/iPad, etc.

Since an IMG is not something users interact with or which themes, in this case the browser doesn't have to also mediate theme support, and navigation / manipulation gestures for it.

4.1.5. The Application / Document Itself

In the simple case of an ALT attribute on an IMG tag, there isn't a lot of work for the accessible document to do. Other than being part of an otherwise well-formed and accessible HTML document, the use of the IMG tag must have useful and appropriate text attached in the ALT attribute. With more complex cases, this would be a much more involved step...

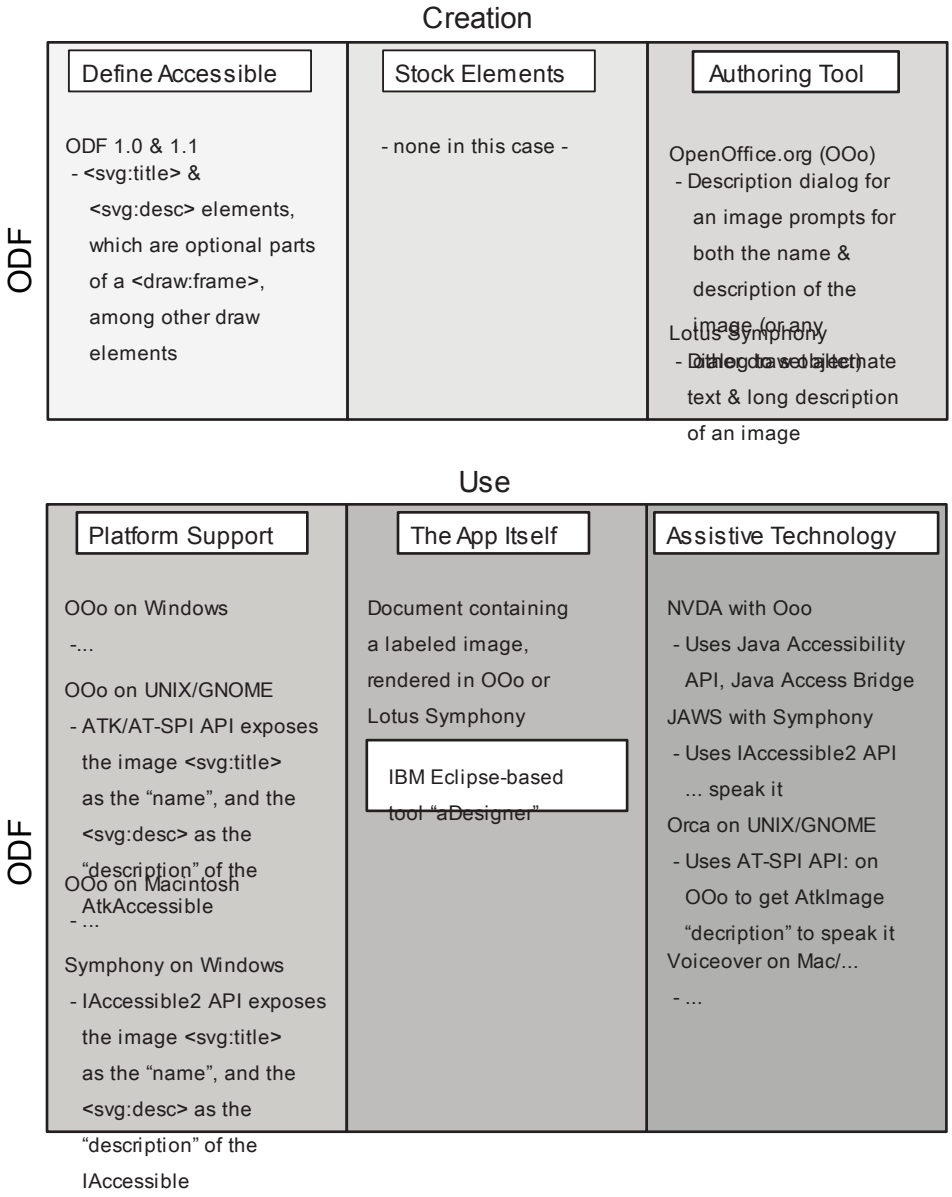
4.1.6. Assistive Technology

Typical assistive technology use of alternative text for an image is that of a screen reader presenting that text in speech or braille. The specifics of how any given screen reader obtains this information varies with the underlying platform, with the web user agent, and also with the assistive technology itself. However in all cases today they arguably use 3rd generation techniques.

For example, the Orca screen reader with Firefox on UNIX/GNOME systems uses the GNOME accessibility framework – specifically AT-SPI – to get the 'description' field of the AtkImage object associated with that IMG. It then speaks and/or brailles this information. Similarly, the NVDA screen reader with Firefox on Windows uses the IAccessible2 API to get the 'description' from the IAccessibleImage object associate with that IMG. NVDA likewise then speaks and/or brailles this information.

Thought it doesn't use a platform accessibility API, the JAWS screen reader with Internet Explorer uses the proprietary COM API of IE to likewise get the ALT text associated with the IMG in order to speak and/or braille it.

4.2. The OAF as applied to images in *ODF*



Drawing 4: The six components of the OAF, as applied to image accessibility in ODF

4.2.1. Define Accessible

The ODF specification defines how to associate text to an image – via the optional <svg:title> element of a <draw:frame> (or several other <draw> tags) for a short accessible name, and via the optional <svg:desc> element of a <draw:frame> (or several other <draw> tags) for a long accessible description. Though formally these are optional elements as per the ODF spec., starting with ODF v1.1 that specification makes it clear how this are to be used for accessibility. [48]

4.2.2. Stock Elements

ODF is a document format, and doesn't include user-interface elements the way a program would (or something like a web application in the example above with jQuery widgets). Therefore there are no stock elements here, nor any step 2 of the OAF for ODF.

4.2.3. Authoring Tool / Developer Tool

When creating documents, ODF editors such as OpenOffice.org should provide a way for an author to associate both the short name and longer description with images (encoded as <svg:title> and <svg:desc>). For example, OpenOffice.org provides this via the "Description" context dialog for any image (or for that matter, any drawing element such as rectangles, circles, etc.) - prompting the author to enter a "title" and a "description" for the image. Lotus Symphony has a similar dialog box for images, prompting the user to enter the "alternate text" and "long description" of the image. These are then recorded as the <svg:title> and <svg:desc> respectively.

Authoring tools may also include ODF accessibility validation functionality, which when run would flag images that lack an <svg:title>, and potentially also prompt for empty <svg:desc>.

4.2.4. Platform Support

For ODF, the office application has the role of exposing accessibility information contained within the document via the platform accessibility support for whichever platform it is running on. While this seems very similar to the HTML case with the web, a key difference is that HTML is a small subset of the web platform – where rich Internet applications are executed code running with within or connected to the web user agent. Thus the web user agent is the platform, whereas the office application doesn't merit such distinction.

On a UNIX/GNOME platform, the ODF office application utilizes the GNOME accessibility framework to expose the contents and structure – and accessibility metadata – of the ODF document to assistive technologies on that platform. To take a specific example, OpenOffice.org would utilize AT-SPI to expose an AtkObject object for the image. The <svg:title> element would be exposed as the "name" of the AtkObject, while the <svg:desc> element would be exposed as the "description" of the AtkObject.

On the Windows platform, the ODF office application might utilize the IAccessible2 accessibility framework to expose the contents and structure – and accessibility metadata – of the ODF document to assistive technologies on that platform. To take a specific example, Lotus Symphony would utilize IAccessible2 to expose an IAccessible object for the image. The <svg:title> element would be exposed as the "name" of the IAccessible, while the <svg:desc> element would be exposed as the "description" of the IAccessible. Also, the IAccessible would be an instance of an IAccessibleImage, and the "description" field of that IAccessibleImage would also contain the <svg:desc> element text. This is described for the various desktop platforms by the OASIS ODF Specification. [15]

4.2.5. The Application / Document Itself

In the simple case of an image in ODF, there isn't a lot of work for the accessible document to do. Other than being part of an otherwise well-formed and accessible ODF document, the images (specifically <draw:frame>) must have at least a valid <svg:title> element associated with them, if not also an <svg:desc> element. With more complex cases, this would be a much more involved step...

The author may want to make use of a tool to help validate the accessibility of their document, such as aDesigner. [16]

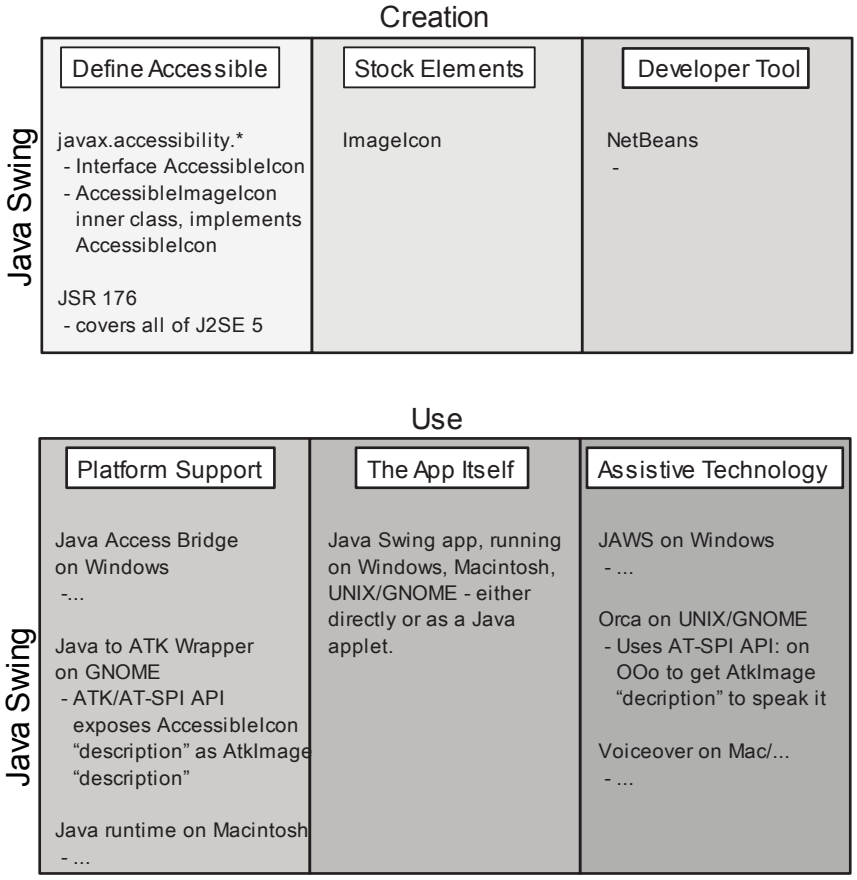
4.2.6. Assistive Technology

As with HTML, typical assistive technology use of alternative text for an image in ODF is that of a screen reader presenting that text in speech or braille. The specifics of how any given screen reader obtains this information varies with the platform – and the accessibility API used on that platform.

For example, the Orca screen reader with OpenOffice.org on UNIX/GNOME systems uses the GNOME accessibility framework – specifically AT-SPI – to get the 'name' and 'description' fields of the AtkObject object associated with that image (which as noted in step 4 above come from the <svg:title> and <svg:desc> ODF elements). It then speaks and/or brailles this information. Similarly, the JAWS and NVDA screen readers with Lotus Symphony on Windows uses the IAccessible2 API to get the 'name' and 'description' fields from the IAccessible object associated with that image. And then JAWS and NVDA likewise then speak and/or braille that information.

An additional technology we are putting into the "Assistive Technology" step of the OAF are two OpenOffice.org plug-ins developed in AEGIS: the odt2daisy and odt2braille plug-ins that will generate DAISY books or print braille output from ODF text files respectively. In the case of an image, odt2daisy extracts the <svg:title> element and transforms that to the ALT attribute of the IMG tag that it creates in the XML DAISY file. This is very similar to what OpenOffice.org does when the user chooses to export the ODF text file to HTML or to PDF → the <svg:title> becomes the alternate text associated with the image. Unfortunately neither DAISY nor HTML have a good place to put the <svg:desc> information, so that is lost in the export. odt2braille exports both the <svg:title> and the <svg:desc> content.

4.3. The OAF as applied to images in Java Swing



Drawing 5: The six components of the OAF, as applied to image accessibility in Java Swing

4.3.1. Define Accessible

The Java specification JSR 176 covers all of the Java Standard Edition version 5 APIs, including the javax.accessibility.* package. This package contains the Java Accessibility API, including specifically the definitions of what every user interface element must expose

to assistive technologies. For user interface elements that contain an image, the API specifies that the element must provide an AccessibleName and AccessibleDescription of that element, as well a description of the image itself, and the width and height of that image.

4.3.2. Stock Elements

Included in the Java Foundation Classes is a stock element for display images – the ImageIcon class. [26] This stock element expressly implements the Java accessibility API through its inner class delegate, AccessibleImageIcon. [4] This inner class provides the image description, width, and height of the AccessibleIcon interface. [3]

Since images themselves aren't user-interface elements within the Java Foundation Classes, some other user-interface element must contain the image for it to be displayed. This is commonly an element of the Swing JButton class [37]. JButtons, like all Swing user-interface elements, implements the Java accessibility API through its inner class delegate AccessibleJButton. This inner class provides the AccessibleName and AccessibleDescription common to all accessible user interface elements.

4.3.3. Authoring Tool / Developer Tool

For creating Java Foundation Classes applications (more commonly known as Swing applications), the common integrated developer environment (or IDE) is NetBeans. [44] ImageIcons are typically displayed within a user-interface element that can show anything with the Icon interface [25] (such as a Swing JButton [36]). NetBeans provides a GUI builder interface (Metisse), which allows developers to compose a user-interface by dragging user-interface elements from a palette and placing them in the desired locations within a window or dialog box. Once the developer places a JButton user-interface element onto a window or dialog box, she can interact with it, and examine its properties. A collection of accessibility properties is exploded for every JButton – including the AccessibleName and AccessibleDescription of that button. This is part of the developer support for accessibility provided in NetBeans (and it extends to all user-interface elements that implement the javax.accessibility.Accessible interface). In addition, the developer can set the icon property of every JButton, which will automatically display the selected image. Unfortunately at this time NetBeans does not then expose the AccessibleIcon description for JButton icon property.

4.3.4. Platform Support

For accessible Java applications running on the Java platform – itself running on top of an underlying operating system such as Windows, GNOME/Unix, or Macintosh – the Java platform accessibility information is exposed via either the Java runtime or via a separately loaded bridge, which bridges or translates the Java accessibility API to that of the underlying operating system.

To take just one example, we will examine a Java application containing an ImageIcon (in perhaps a JButton), when running on a GNU/Linux or Solaris environment with the GNOME desktop. In GNOME, the Java Access Bridge for GNOME will be loaded into the Java runtime, and will expose the accessibility information of JButton – the AccessibleName and AccessibleDescription – as an AtkObject whose atk_object_name is the AccessibleName, and whose atk_object_description is the AccessibleDescription. Further, because the JButton contains an accessible icon, it will be exposed as an instance of AtkImage (a subclass of AtkObject), and the AtkImage width, height, and description will come from the JButton's AccessibleIcon width, height, and description.

Similar things happen on the other platforms.

4.3.5. The Application / Document Itself

In the case of a Java Swing application with ImageIcons contained inside of JButtons, the developer must ensure ensure that appropriate AccessibleNames, AccessibleDescriptions (where needed), and ImageIcon descriptions are set. Of course, the rest of the application must be accessible, and any manipulable images must be contained in user-interface elements that support keyboard operation, and are in the TAB order. If there is any visible text associated with an image (such as a label), then a labelling relationship must also be set, linking the label to the image (or more specifically, to the user-interface element containing the image).

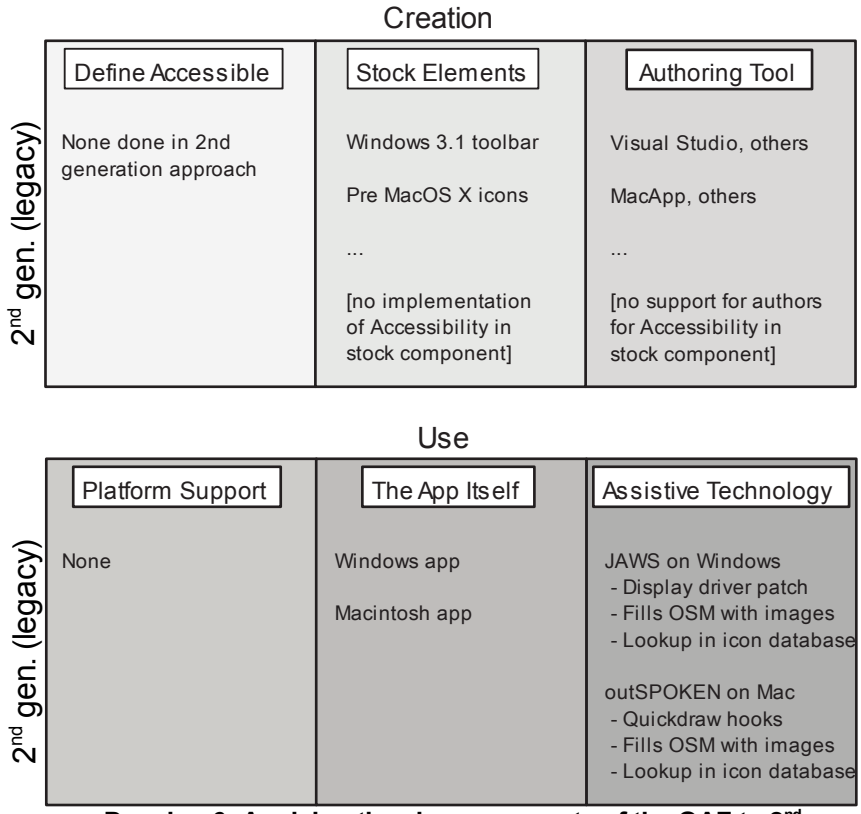
4.3.6. Assistive Technology

The typical assistive technology use of alternative text for an image in a Java application is that of a screen reader presenting that text in speech or braille. The specifics of how any given screen reader obtains this information varies with the platform – and the accessibility API used on that platform.

For example, the Orca screen reader with a Java Swing application on UNIX/GNOME systems uses the GNOME accessibility framework – specifically AT-SPI – to get the 'name' and 'description' fields of the AtkObject object associated with the user-interface element containing the image (which as noted in step 2 above is commonly a Swing JButton or some other Swing user-interface element class). It then speaks and/or brailles this information. It is rare for an assistive technology to further query for the actual distinct image description, but were it to do so, it would obtain that from the AtkImage interface that is exposed for AccessibleIcons of the Java accessibility API.

Similarly, the JAWS and NVDA screen readers with Lotus Symphony on Windows uses the IAccessible2 API to get the 'name' and 'description' fields from the IAccessible object associated with that image. And then JAWS and NVDA likewise then speak and/or braille that information.

4.4. For comparison: the OAF as applied to 2nd generation (legacy) environments



Drawing 6: Applying the six components of the OAF to 2nd generation (legacy) environments

4.4.1. Define Accessible

In the legacy 2nd generation world, there is no definition of accessibility to use. So specifically, there is no way to explicitly associate a name and description with an image.

4.4.2. Stock Elements

There are any number of stock user-interface elements on 2nd generation platforms that display images. These include the Win32 toolbar of Windows 3.1, and the icons of the pre-OS X Macintosh. Because there is no definition of how to create an accessible image, there is no way to incorporate that into these stock user-interface elements.

4.4.3. Authoring Tool / Developer Tool

For creating applications that include images within their user-interfaces, there are any number of developers tools on 2nd generation platforms. These include MacApp for pre-OS X Macintosh, and Microsoft Visual Studio for Windows 3.1. Since there is neither a definition describing an accessible icon, nor any stock user-interface elements implementing that definition, there is nothing for a developer tool to do to help a developer create an accessible application with accessible images.

4.4.4. Platform Support

Since there is no definition of accessibility for the platform, there is no explicit platform support for accessibility and for enabling assistive technologies to get information from accessible applications.

4.4.5. The Application / Document Itself

In a 2nd generation platform, there still may be conventions an application can follow in order to be more friendly to the reverse-engineering techniques used by 2nd generation assistive

technologies. For example, a common screen reader technique for obtaining images from a toolbar is to walk the HWND window hierarchy, and if it finds an HWND of type TOOLBAR – containing some number of child HWNDs of type TBBUTTON – it will then obtain the pixels from the images associated with those TBBUTTONs, and use them to lookup possible names associated with those images (see section 6 below). Other assistive technologies may use other – perhaps less common – techniques (such as patching the BitBlt graphics call, and collecting images based on the pixels in the BitBlt calls).

Thus, in a 2nd generation platform, an application is said to be "accessible" if the assistive technology (or collection of assistive technologies) a particular user or organization is using works with it. Unfortunately, this means that an application is highly restricted in how it can realize its user interface. It has no option to be innovative, and yet explicitly expose accessibility information to assistive technologies.

4.4.6. Assistive Technology

For assistive technologies to be effective in a 2nd generation platform, they need to do all of the work of obtaining the information on the screen. In the case of names/descriptions of images, this is typically done in a screen reader by having an external library of pre-named images. Then whenever exactly the same pixels are used in an image on screen – and those pixels are somehow obtained by the screen reader - the screen reader is able to provide the (pre-defined) name. Otherwise, it will simply say "icon".

5. RESEARCH RESULTS FOR THE OAF FROM AEGIS DEVELOPMENTS

As noted in the Executive Summary at the start of this document, the OAF as delivered in AEGIS is two things:

- A document describing the framework of things needed for 3rd generation accessibility, as validated by the prototypes and user/developer feedback in AEGIS (this document), and
- A collection of largely open source prototypes and code Deliverables implementing various aspects of the OAF, proven in AEGIS and contributed back to the open source projects of which they are part

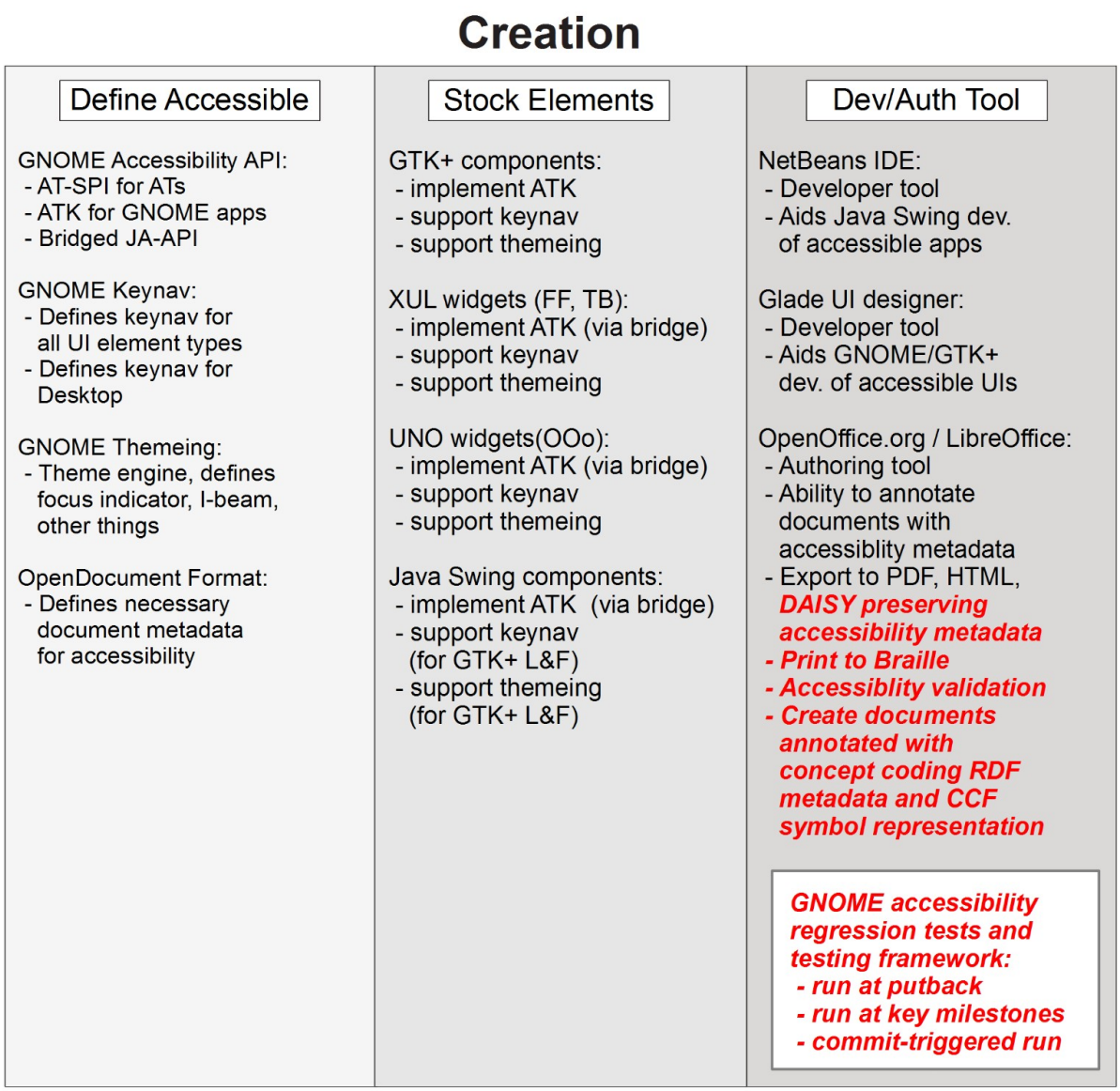
It is the second of these that represent the research directions for the OAF in AEGIS. The sections below outline what that work was, and where that work fits into the 6 steps of the OAF. More detailed information on how each Deliverable supports and implements facets of the OAF can be found in the reports that accompany those Deliverable and prototypes.

As the AEGIS project developed prototypes in three broad technology areas, we describe the research we did in AEGIS by those three areas below.

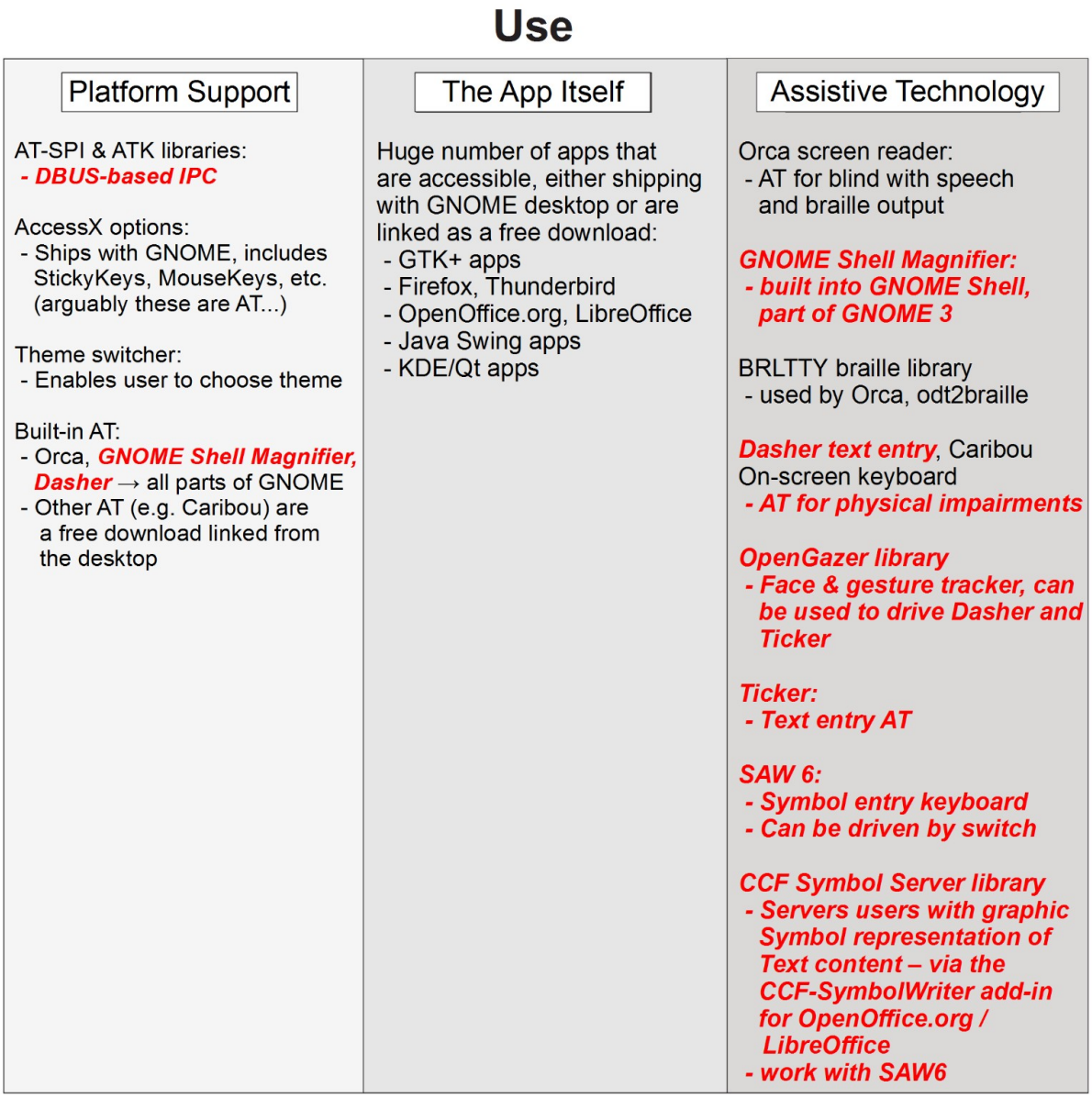
5.1. OAF Research Relating to the Open Accessible Desktop

The AEGIS desktop work was primarily about proving the existing 3rd generation accessibility techniques that already exist on the UNIX/GNOME desktop, and pushing the boundaries of what is there to see whether the APIs and "accessibility definitions" in UNIX/GNOME are sufficient or need to be updated. Further, in this research area we also focused on document authors, and the authoring tool support that they need.

This desktop research work is also illustrated in the two OAF diagrams immediately below, with AEGIS work highlighted in ***boldface italicized red*** text.



Drawing 7: The three Creation steps of the Open Accessibility Framework for the Open Desktop



Drawing 8: The three Use steps of the Open Accessibility Framework for the Open Desktop

5.1.1. OAF Research Relating to Magnification on the Open Accessible Desktop

As part of the research and development around building magnification support into GNOME Shell, we implemented the groundwork for a more sophisticated, and context-aware/context-sensitive magnification system for users with low vision. Furthermore, we implemented a range of colour/contrast settings, and mouse cursor tracking aids – which were tested in our end-user pilots. In doing this work, we determined that the the existing accessibility API information described in the OAF was sufficient to meet the needs of our magnification work, and appears to also be sufficient for contextual magnification.

This GNOME Shell Magnification work has been folded into GNOME Shell itself, and is shipping as part of GNOME 3.6.

5.1.2. OAF Research Relating to OpenOffice.org Work

We pushed the boundaries of 3rd generation accessibility, and looked at what was needed in the OAF to support authors in creating accessible documents. We did this in three ways: (1)

we determined what was needed as far as a "definition of accessibility" for document content in order to support the alternate accessible content formats of DAISY and braille; (2) we determined what was needed as far as "authoring tool support" to help authors meet the "definition of accessibility" for documents; (3) we created an accessibility checker to help authors ensure their documents were accessible; and (4) we built a specialized add-in to OpenOffice.org for users with cognitive impairments, to determine whether this specialized use case with this specific application is cause for additions/expansion of the OAF.

What we found was that the earlier work of the OASIS ODF Accessibility Task Force had in fact made the necessary additions to the ODF specification ("definition of accessible") to fully support DAISY and braille documents. In our work on supporting users with language/learning cognitive impairments in OpenOffice.org, we found that it would be helpful to create TrueType [56] fonts to better enable them to be displayed within the OpenOffice.org Writer frame. We then found these same fonts could be re-used for our mobile application for users with related cognitive impairments, as well as for general messaging applications that we also built within AEGIS.

We have already spun off these four developments. Odt2daisy and odt2braille are add-ins to OpenOffice.org and LibreOffice that have been downloaded many thousands of times and are in active use in DAISY and braille production centres. The AccessODF add-in has likewise been downloaded many times, and is finding use in those same DAISY and braille production centres, as well as in all organizations that have a need to produce accessible output. Finally, the CCF Symbol Writer add-in, along with the TrueType and OpenType fonts for Blissymbols [11] and ARASAAC [6] pictograms, are also available and are finding use among centres serving the needs of people with cognitive language/learning impairments.

5.1.3. OAF Research Relating to the GNOME Accessibility API shift to DBUS

The 3rd generation accessibility API on the GNOME desktop was based on the CORBA inter-process communication mechanism (in GNOME 2.x). Other 3rd generation accessibility APIs were likewise built on fairly sophisticated inter-process communication mechanisms (e.g. Microsoft COM for MSAA and IAccessible2). One goal of the research and development activity of shifting the GNOME accessibility API and framework to DBUS was to determine how "lightweight" an environment can run 3rd generation accessibility. DBUS is the lightweight IPC mechanism used on the One Laptop Per Child Sugar environment, as well as the Maemo and MeeGo Linux mobile environments used on the Nokia Internet Tablet and potential future Linux mobile products. DBUS is also used for inter-process communication in the KDE/Qt desktop, and this R&D work will enable interoperability with KDE/Qt (which may also surface additions to the OAF).

We were able to not only demonstrate that the 3rd generation GNOME accessibility API could work over DBUS, but the GNOME community adopted our work which is now part of the GNOME 3 desktop. In fact, because of the work we started, accessibility is now "always on" starting in GNOME 3.6, because the performance impact of having it always on is now negligible. Furthermore, the KDE/Qt environment has likewise adopted the work we began in AEGIS, and now a growing number of KDE/Qt applications are accessible on the Open GNOME desktop.

5.1.4. OAF Research Relating to Accessibility Testing

A different use of the "definition of accessibility" is that of programmatic testing – to ensure that an application properly implements the definition of accessibility (to the extent that programmatic testing can do so – which will never be 100% of the case). As part of this R&D activity, we tried to push the boundaries of the GNOME accessibility API with programmatic tools to help find

accessibility bugs. As expected, this work showed some implementation flaws in the GNOME accessibility API, but didn't show us any missing facets of the OAF.

5.1.5. OAF Research Relating to Desktop Real-Time-Text

Real-Time-Text communication is fundamentally an application use case, not that of an accessibility API. In our work implementing a prototype real-time-text application on the Open Desktop, we didn't find any new facets missing from the OAF.

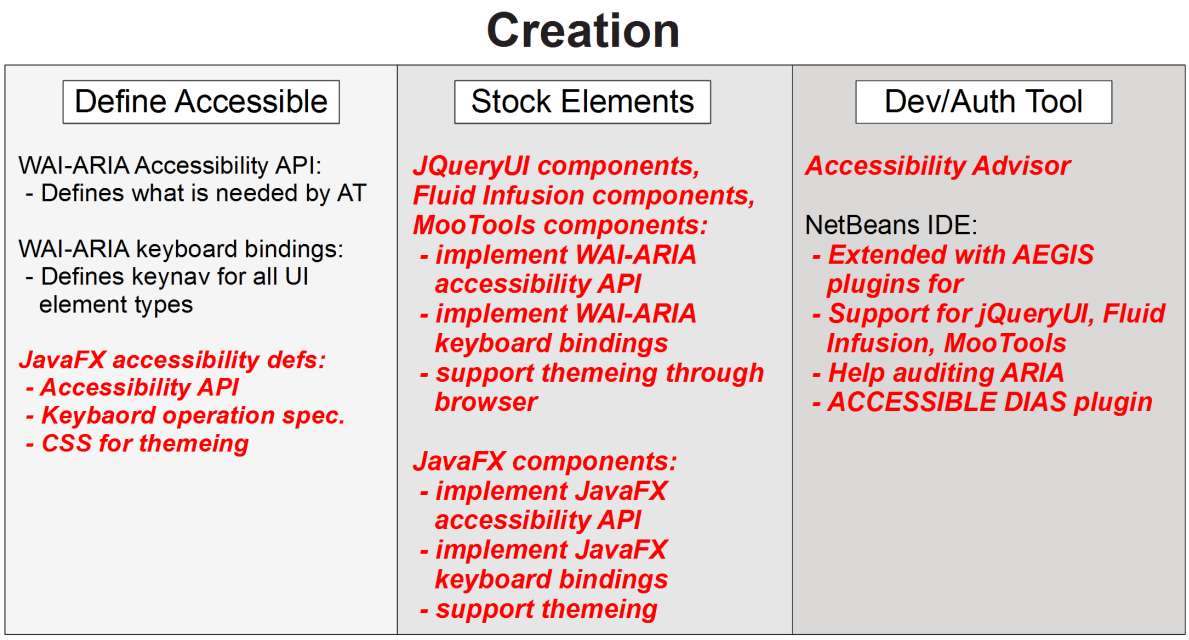
5.1.6. OAF Research Relating to OpenGazer

The final desktop R&D activity was around developing eye-tracking and gaze-switch systems. While we found several obvious and "brute force" ways of utilizing such technologies – e.g. treating the gaze switch only as a switch, treating the tracker only as a mouse – we were able to also explore a few more innovative and efficient user-interaction modalities as part of this work. In particular, we developed a new assistive technology – Ticker – which was able to dynamically compensate for some of the inherent time delays in the gaze-switch system, and found that a number of users with locked-in syndrome were able to be far more efficient in communicating via text thanks to the gaze-switch combined with Ticker.

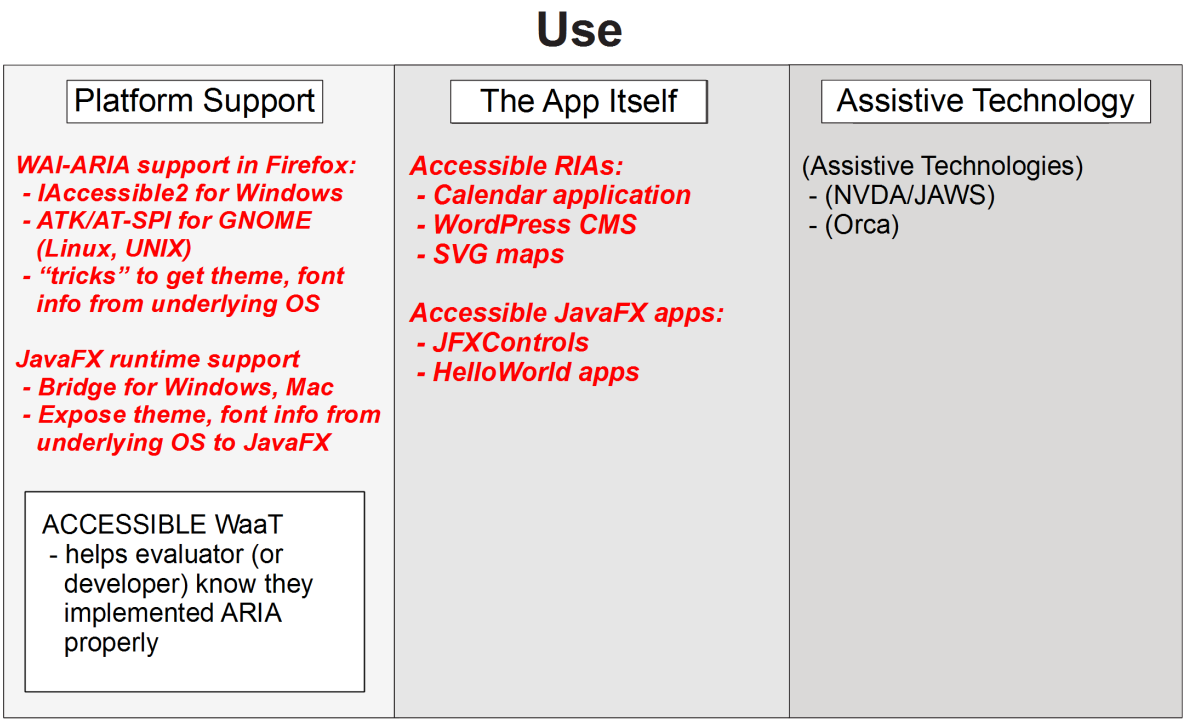
5.2. OAF Research Relating to Web Applications Accessibility

For web application accessibility, AEGIS research and development involved the first five of the six parts of the OAF. The research also had a "cascading effect" → work in one part of the OAF flowed forward to the next part(s), and the successes (or issues) that arose there flowed backward the previous part(s).

This desktop research work is also illustrated in the two OAF diagrams immediately below, with AEGIS work highlighted in ***boldface italicized red*** text.



Drawing 9: The three Creation steps of the Open Accessibility Framework for Web applications



Drawing 10: The three Use steps of the Open Accessibility Framework for Web applications

5.2.1. OAF Research Relating to the Definition(s) of Web Application Accessibility

JavaFX is a new graphical environment for Java, for developing highly graphical and animation-based user interfaces. In AEGIS we developed an initial, prototype accessibility API for JavaFX. We anticipated the possibility that in order to deal with this very different graphical environment, we might find that the existing minimum set of API calls and situations aren't sufficient, and would need to be extended (and thus the OAF would need to be extended). There also were particular implications for keyboard operation and themeing in this highly graphical and animation-based environment. However, we found that no additions/extensions were needed for the OAF in this work.

5.2.2. OAF Research Relating to the User-Interface Elements of Web Application Accessibility

We built accessibility support into multiple use-interface element sets – both based on JavaFX and using the new accessibility API there, as well as based on AJAX/DHTML and the WAI-ARIA specification (which comprised jQuery, Fluid Infusion, and MooTools). In this work, we provided multiple proof-points for the WAI-ARIA specification, and our early prototype work for JavaFX has likewise shown – thus far – that the JavaFX accessibility API appears to be solid. While this work didn't raise any new issues we needed to take into account for the OAF, it did help prove the OAF model.

5.2.3. OAF Research Relating to Developer Tool(s) of Web Application Accessibility

As part of our work for developers, we will built a number of tools to support developers in the task of creating accessible applications – all using the user-interface elements also developed in AEGIS. In the web domain, we built two main tools: the Accessibility Advisor tool to help developers understand what they should do, how they should do it, and why they

should do it; and a plug-in for NetBeans for development of rich Internet applications using our ARIA-enabled user interface components (from jQuery, MooTools, and Fluid Infusion).

In AEGIS, these tools helped developers find accessibility errors, and made the development process more efficient and effective. These tools were used to create some of our sample accessible applications. They provided further support for the OAF model.

5.2.4. OAF Research Relating to Platform Support of Web Application Accessibility

A core idea in the AEGIS OAF is that the underlying platform or operating system has certain responsibilities for supporting accessibility. This idea was clearly demonstrated in our web work. We made key early contributions to the larger effort of supporting WAI-ARIA in Firefox on Windows and GNOME, and proved that approach sufficiently for it to be taken up by Microsoft for Internet Explorer and Apple for Safari and Google for Chrome. Most of our accessible web applications demos were conducted using Firefox on Windows, using JAWS and NVDA and their support of IAccessible2 in Firefox. Similarly our initial prototype JavaFX runtime work on Windows and Macintosh likewise provided these aspects of the OAF for platforms.

Our contributions to Firefox for ARIA support are included in the last several major Firefox releases. And Firefox's ARIA support continues to lead the industry, with everyone else playing catch-up. [21] As of September 2012, Firefox has the leading score of 86/100. Next runner up is IE with 45/100. On Macintosh the best score is Safari with 72/100. This confirms the value and contributions AEGIS has had on the web accessibility field.

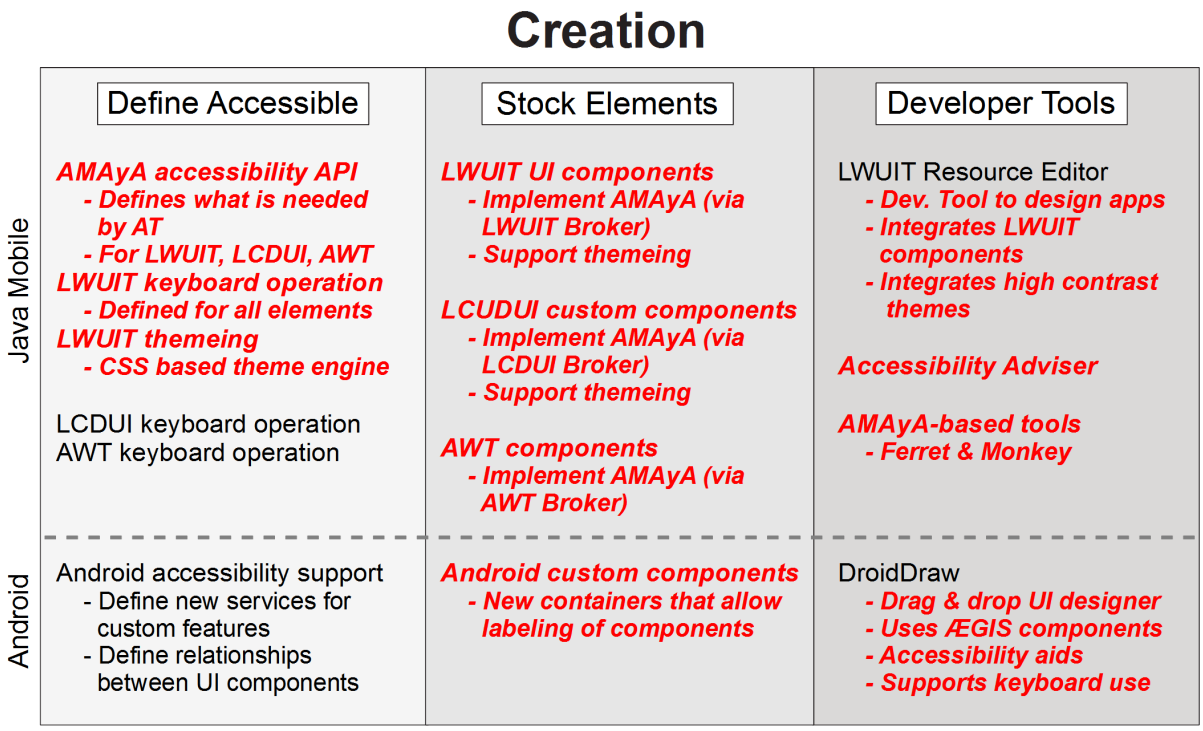
5.2.5. OAF Research Relating to the Applications of Web Application Accessibility

Finally in the Web Application Accessibility area in AEGIS, we built a number of sample accessible web applications. These sample applications utilized the user-interface elements developed in AEGIS (jQuery, MooTools, Fluid Infusion), and several were built using the developer tool support we created in AEGIS. We used these applications in our end-user pilot tests – primarily using Firefox – and thereby tested "five sixths" of the OAF (steps #1 through #5).

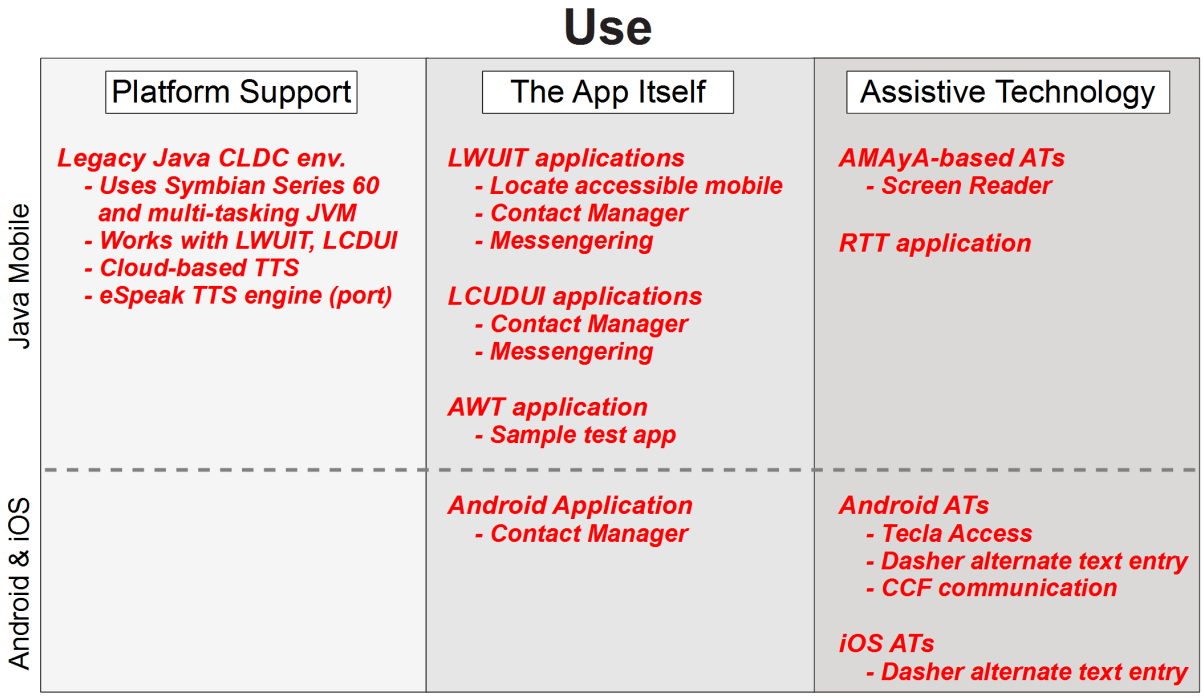
5.3. OAF Research Relating to Mobile Applications Accessibility

For mobile application accessibility, the AEGIS research and development involved all of the six parts of the OAF. The research also had a "cascading effect" → work in one part of the OAF flowed forward to the next part(s), and the successes (or issues) which arose there flowed backward the previous part(s).

This desktop research work is also illustrated in the two OAF diagrams immediately below, with AEGIS work highlighted in ***boldface italicized red*** text.



Drawing 11: The three Creation steps of the Open Accessibility Framework for Mobile applications



Drawing 12: The three Use steps of the Open Accessibility Framework for Mobile applications

5.3.1. OAF Research Relating to the Definition(s) of Mobile Application Accessibility

In AEGIS we developed an accessibility API for the Java Mobile platform. We also developed themeing conventions and made a few extensions to the existing keyboard operation scheme (for navigating non-editable text content). We thus developed the complete "definition of accessibility" for the Java Mobile platform. In doing so, discovered a few unique challenges in the mobile space – namely keyboard traversal for non-ediable content. However, none of these discoveries resulted in any necessary changes to the OAF to address

them. Furthermore, we found that a single AEGIS Mobile Accessibility API (AMAyA) was sufficient for the LWUIT, LCDUI, and AWT user interface component sets as used in both CDC and CLDC environments – which was another research objective.

5.3.2. OAF Research Relating to the User-Interface Elements of Mobile Application Accessibility

We developed accessibility support for multiple use-interface component sets for the Java Mobile platform: LWUIT, LCDUI, and AWT. For these, we fully implemented the "definition of accessible", and were able to use a single definition for all three, in both the CDC and CLDC environments.

5.3.3. OAF Research Relating to Developer Tool(s) of Mobile Application Accessibility

For mobile application developers we developed three tools. First, we created a version of the Accessibility Advisor tool for mobile (the same tool as for web described above in section 5.2.3 "OAF Research Relating to Developer Tool(s) of Web Application Accessibility" except with content for mobile developers). Then we modified the existing LWUIT Resource Editor tool to flag accessibility errors and help developers/UI designers create accessible LWUIT application user interfaces using LWUIT and AMAyA, paired with a modified NetBeans plug-in to consume LWUIT Resource files for creating accessible LWUIT applications. Finally, we modified the existing open source Android developer tool DroidDraw, enabling the development of accessible Android applications (through the use of pre-defined, compound components such as pre-labeled edit-text fields).

These developer tools expose much of the "definition" of accessibility as well as the specific implementations for the mobile user interface components, fulfilling their role in the OAF.

5.3.4. OAF Research Relating to Platform Support of Mobile Application Accessibility

Our platform work for mobile in AEGIS took place on the Java Mobile platform (we have no role in Android or iOS). In building our prototype solutions for Java Mobile, we identified all of the issues that need to be addressed in the platform in order to support the inter-process communication needed between mobile applications and AT, in order to automatically load AT, the necessary AT support services, etc. Some of these we were able to implement solutions for, others will need to be addressed as part of commercial exploitation. This valuable research results also contrast well with a few "OAF holes" in iOS and Android, even as those platforms offer some solutions for people with disabilities that are far more mature and accepted than our research prototypes (and one of those holes has already been addressed with Android version 4.1, as discussed above in section 1.1.7 "Android release 1.6 and later: the Android Accessibility API and Google-provided TalkBack and Eyes-Free Shell").

5.3.4.1. OAF Research Relating to the Applications of Mobile Application Accessibility

As part of our mobile application work, we developed a number of custom-built mobile applications specifically addressing the needs of users with cognitive impairments. We also built a media player for Java Mobile that supports display of captions for the deaf. In addition, our sample mobile applications showcased work we did in the Java mobile user interface components, as well as key accessibility features of the Android platform. Our sample applications were well received in our end-user pilot tests, and several of them are being commercially exploited via the Nokia Ovi store and the Android Play store.

Implementing these applications helped us find a number of places where our initial Java Mobile accessibility API AMAyA had some shortcomings, which we were able to address as part of our final work on AMAyA. Similarly end-user testing helped improve our high contrast themes for LWUIT. Finally, all of our sample apps were tested with mobile AT – either our own or the AT built into Android.

5.3.5. OAF Research Relating to Assistive Technologies for Mobile Application Accessibility

Finally in the Mobile Application Accessibility area in AEGIS, we will built a number of assistive technologies for mobile environments. First, we built a prototype screen reader for Java Mobile that leveraged all of OAF work we did in AEGIS for mobile. This proved our OAF approach to mobile. The criticisms of Java mobile accessibility from our user pilot feedback focused on things like the performance of the text-to-speech and the quality of the keyboard (and lack of touch screen) – all things unrelated to our underlying ability to provide all of the information on the screen and make a broad range of applications accessible on low-end mobile devices. We also developed a Java Mobile RTT application, to address some of the needs of the deaf community.

During the 4 years of the AEGIS project (and further, the 1.5 years prior to the project's start, counting back to when we began work on the project proposal), the mobile world changed dramatically. This period saw the introduction of the iPhone and Android, the introduction of assistive technologies for iOS and Android, the complete shift away from Windows Mobile 6.x (with the 3rd party AT options it had), the development of a BlackBerry accessibility API and screen readers for it (first commercial and then built-in provided by RIM) and the rise of a new BlackBerry environment BlackBerry 10; and finally a realignment around touch screen UIs. We adapted significantly to this shift, and developed several assistive technologies for Android and iOS in the project that either built on work from elsewhere in the project (e.g. the Concept Coding Framework of the Open Desktop) or were totally new but nonetheless taking advantage of 3rd generation accessibility features of these new mobile platforms.

Particularly notable among our mobile AT work are the things we did for people with physical impairments. Tecla is an incredibly innovative dynamic on-screen keyboard that works with nearly all Android applications, and connects to the open source hardware Tecla Shield which connects a users' wheelchair to the Android phone or table via bluetooth, so that whatever adaptive switches/controls they use to control their wheelchair can now be used to control their phone or tablet. Tecla builds on some of the key ideas of the GNOME On-screen Keyboard (GOK), itself perhaps the first 3rd generation assistive technology for people with upper limb mobility impairments.

For folks with a combination of physical and cognitive impairments, we build CCF Symbol Droid for Android, turning an Android phone or Tablet into a communication aid. While this itself wasn't new (though again not much of an option prior to the start of AEGIS), we connected this writing application into the cloud, enabling a clinician or other assistant to help expand the vocabulary of CCF Symbol Droid users, with updated vocabulary automatically downloaded from the cloud. This work – among a few other things in AEGIS – point to one of the next steps in accessibility innovation after 3rd generation and OAF: leveraging 3rd generation solutions via the cloud.

6. CONCLUSIONS

With this document we have set forth the final comprehensive treatment of the OAF within the AEGIS project. In doing so, we have met one of the key goals of AEGIS: the further

development and enhancement of the OAF, based on the knowledge gained from implementing various components of it in the desktop, web, and mobile spaces.

6.1. Future OAF research directions

One of the "low-hanging fruit" for further research and innovation involves building on the groundwork we created for context-aware magnification in the GNOME Shell Magnifier. The magnification support in GNOME Shell allows the creation of multiple "zoom regions", each which can be tracking information independently. From conversations with low vision users, providing application-specific and context-specific magnification tied to the tasks they are doing may provide them with significant efficiency and productivity gains. For example, instead of needing to periodically move the mouse to cause the magnification region to move to see whether any incoming message have arrived – a context-specific magnifier could always have a small, secondary "zoom region" either locked to the place where this application indicates new information, or might simply automatically appear when something new is rendered (just as a screen reader automatically reads new information in similar situations).

Though the AEGIS project contained a significant focus on cognitive impairments, the ICT needs of these users remains under served. The OAF doesn't yet fully reflect these needs. Therefore a key research direction involves developing platform-wide AT for users with cognitive impairments – and then understanding what the needs are of such platform-wide AT such that they might be fed back into the OAF and enhance the various steps of the OAF (e.g. accessibility API additions, or minimum theme facets).

Another important research focus is around speech recognition and dictation. An early demo of the Java accessibility framework [34] utilized the JavaSpeech API and IBM ViaVoice to control a Java/Swing application with voice. The GNOME OnScreen Keyboard utilizes similar API calls for controlling GNOME applications. These two examples suggest that the OAF accessibility API requirements may be sufficient to address the needs of a voice control and dictation system, but as none has really been built, we cannot know for sure. Building one would be a very useful research undertaking.

Also of interest in exploring potentially new and novel assistive technologies – things that we haven't seen emerge from the 2nd generation environment. This is perhaps the most promising research direction. An example of such a potential new and novel AT is embodied in the User Interface Façades work undertaken jointly at York University in Canada and the Université Paris-Sud in France, which seeks to re-render the user interface in a simplified fashion, limiting the number of options a user with cognitive disabilities can have access to and potentially rendering them in an easier to use fashion. [74]

Finally, we believe that significant accessibility innovation will happen in connection with cloud-delivered accessibility services. The combination of incredibly powerful mobile devices that contain not only always-on network connections, but a wide range of sensors; coupled with super computers in the cloud that can both provide virtually unlimited information resources but can also take over more complex computing tasks (cf. Siri voice recognition on iOS), present tremendous opportunities for advances in accessibility. Super-computer video analysis can help with not only identification but also wayfinding. Techniques for building 3D models through accelerometer-connected video capture can likewise aid in locating objects.

All of these are just some of the tremendous accessibility research potentials going forward, many of which build upon the work we established in the AEGIS OAF.

REFERENCES

[1] "User groups' and stakeholders' definition and UCD Implementation Plan" AEGIS Deliverable D1.1.1

[2] [U.S. Access Board Advanced Notice of Proposed Rulemaking from 2011 draft rule: http://www.access-board.gov/sec508/refresh/draft-rule.htm

[3] AccessibleIcon documentation: http://download.oracle.com/javase/1.5.0/docs/api/javax/accessibility/AccessibleIcon.html

[4] AccessibleImageIcon documentation: http://download.oracle.com/javase/1.5.0/docs/api/javax/swing/ImageIcon.AccessibleImageIcon.html

[5] Applying WCAG 2.0 to Non-Web Information and Communications Technologies (WCAG2ICT): http://www.w3.org/TR/2012/WD-wcag2ict-20120727/

[6] ARASAAC picrograms: http://www.oatsoft.org/So(ftware/arasaac-pictograms

[7] ATK documentation at: http://library.gnome.org/devel/atk/stable/

[8] Blackberry Accessibility API documentation: http://www.blackberry.com/developers/docs/6.0.0api/net/rim/device/api/ui/accessibility/package-summary.html

[9] Blackberry Accessibility Development Guide: http://docs.blackberry.com/en/developers/deliverables/20100/index.jsp?name=Accessibility+-+Development+Guide+-+BlackBerry+Java+

[10] Blackberry Java Development Environment (JDE): http://us.blackberry.com/developers/javaappdev/javadevenv.jsp

[11] Blissymbols: http://en.wikipedia.org/wiki/Blissymbols

[12] CodeFactory website: http://www.codefactory.es/

[13] Comments submitted to the 2011 ANPRM: http://www.regulations.gov/#!docketDetail;D=ATBCB-2011-0007

[14] D1.2.1 AEGIS OAF and high-level architecture: http://www.aegis-project.eu/images/docs/AEGIS_D1.2.1_final-revised_2nd_Annual_Review.pdf

[15] Direction on platform support for <svg:title> and <svg:desc> is described in the Appendix E of the ODF specification: http://docs.oasis-open.org/office/v1.1/OS/OpenDocument-v1.1-html/OpenDocument-v1.1.html

[16] Eclipse ACTF aDesigner: http://www.eclipse.org/actf/downloads/tools/aDesigner/

[17] ETSI Specialist Task Force 333 inventory: http://portal.etsi.org/stfs/STF_HomePages/STF333/STF333.asp

[18] EU Mandate 376: http://ec.europa.eu/information_society/activities/einclusion/archive/deploy/pubproc/eso-m376/a_documents/m376_en.pdf

[19] Gkemou, M. et al., *AEGIS Open Accessibility Everywhere Group (OAEG) framework and status,* Deliverable 5.5.3, AEGIS project, CN. 224348, 7th Framework Programme, ICT & Ageing, December 2011.

[20] GTK+ documentation at: http://library.gnome.org/devel/gtk/stable/

[21] HTML5 Accessibility report of Windows Browser support for ARIA: http://html5accessibility.com/

[22] http://www.ace-centre.org.uk/index.cfm?pageid=0192A99B-3048-7290-FE5E750E8F5887B3&productid=01916C82-3048-7290-FEF51AA5461176B2

[23] Humanware website: http://www.humanware.com/

[24] IAccessible2 API documentation: http://www.linuxfoundation.org/collaborate/workgroups/accessibility/iaccessible2/overview

[25] Icon documentation: http://download.oracle.com/javase/1.5.0/docs/api/javax/swing/Icon.html

[26] ImageIcon documentation: http://download.oracle.com/javase/1.5.0/docs/api/javax/swing/ImageIcon.html

[27] Inclusive Design Research Center's Accessible Digital Office Document Project website: http://adod.idrc.ocad.ca/node/1

[28] Wikipedia entry on Computer Accessibility at http://en.wikipedia.org/wiki/Computer_accessibility#Open_Accessibility_Framework

[29] OAEG section on the Open Accessibility Framework: http://www.oaeg.eu/index.php?option=com_jumi&fileid=162&Itemid=36

[30] ETNA project - European Thematic Networkon Assistive Information Technologies - http://www.etna-project.eu/

[31] ISO/IEC TR 29138: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=45161

[32] ITI comments on 2011 ANPRM: http://www.regulations.gov/#!documentDetail;D=ATBCB-2011-0007-0074

[33] Java Access Bridge for Windows at http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136191.html and for UNIX (GNOME) at http://live.gnome.org/Java%20Access%20Bridge

[34] Java Accessibility API with Java Speech voice control show in a keynote demonstration at the JavaOne conference in 1998

[35] Java Speech API (JSR): http://jcp.org/en/jsr/detail?id=113

[36] JButton documentation: http://download.oracle.com/javase/1.5.0/docs/api/javax/swing/JButton.html

[37] JButton documentation: http://download.oracle.com/javase/1.5.0/docs/api/javax/swing/JButton.html

[38] Locked-in syndrome: http://en.wikipedia.org/wiki/Locked-in_syndrome

[39] Mandate 376 current public drafts: http://www.mandate376.eu/#Participate

[40] Mandate 376 timeline: http://www.mandate376.eu/#TimePlan

[41] Microsoft Active Accessibility API documentation: http://msdn.microsoft.com/en-us/library/ms971310.aspx

[42] Microsoft UI Automation documentation: http://www.microsoft.com/downloads/en/details.aspx?FamilyID=a1fe1066-bf4f-44fc-834b-676b311e83a2&DisplayLang=en

[43] Microsoft Windows 8 accessibility features: http://www.microsoft.com/enable/products/windows8/

[44] NetBeans: http://netbeans.org/

[45] OpenOffice.org 2.0 accessibility white paper: http://ui.openoffice.org/accessibility/whitepaper.html

[46] Oratio screen reader website: http://www.humanware.com/oratio

[47] See Peter Korn's blog entry: http://blogs.sun.com/korn/entry/completing_the_accessibility_picture_iaccessible2

[48] See section 9.2.20 “Title and Description”, and also Appendix E “Accessibility Guidelines” in the ODF v1.1 specification, at http://docs.oasis-open.org/office/v1.1/OS/OpenDocument-v1.1-html/OpenDocument-v1.1.html

[49] Tecla – an assistive technology for mobile devices: http://mobile-accessibility.idrc.ocad.ca/projects/tekla

[50] Tecla in the Android Play store: https://play.google.com/store/apps/details?id=ca.idi.tekla&hl=en

[51] Telecommunications and Electronic and Information Technology Advisory Committee Report Recommendations Subpart C: Technical Requirements: http://www.access-board.gov/sec508/refresh/report/#a65

[52] Telecommunications and Electronic and Information Technology Advisory Committee Report Recommendations Technical Requirements Chapter 3 Requirements for User Interface and Electronic Content: http://www.access-board.gov/sec508/refresh/report/#a653

[53] Telecommunications and Electronic and Information Technology Advisory Committee Report Recommendations Technical Requirements Chapter 7 Additional Requirements for Authoring Tools: http://www.access-board.gov/sec508/refresh/report/#a657

[54] Telecommunications and Electronic and Information Technology Advisory Committee Report Recommendations: http://www.access-board.gov/sec508/refresh/report/#a6

[55] Telecommunications and Electronic and Information Technology Advisory Committee Report: http://www.access-board.gov/sec508/refresh/report

[56] TrueType: http://en.wikipedia.org/wiki/TrueType \

[57] U.S. Access Board Advanced Notice of Proposed Rulemaking from 2010 draft rule: http://www.access-board.gov/sec508/refresh/draft-rule2010.htm

[58] U.S. Access Board draft rule #404 Keyboard Operation: http://www.access-board.gov/sec508/refresh/draft-rule.htm#i404

[59] U.S. Access Board draft rule #406 Navigation: http://www.access-board.gov/sec508/refresh/draft-rule.htm#i406

[60] U.S. Access Board draft rule #409 User Preferences: http://www.access-board.gov/sec508/refresh/draft-rule.htm#i409

[61] U.S. Access Board draft rule #410 Interoperability with Assistive Technology: http://www.access-board.gov/sec508/refresh/draft-rule.htm#i410

[62] U.S. Access Board draft rule #411 Compatible Technologies: http://www.access-board.gov/sec508/refresh/draft-rule.htm#i411

[63] U.S. Access Board draft rule #412 Assistive Technology Function: http://www.access-board.gov/sec508/refresh/draft-rule.htm#i412

[64] U.S. Access Board draft rule #413 Authoring Tools: http://www.access-board.gov/sec508/refresh/draft-rule.htm#i413

[65] U.S. Access Board draft rules, Chapter 4 Software: http://www.access-board.gov/sec508/refresh/draft-rule.htm#software

[66] U.S. Access Board draft rules, Chapter 5 Electronic Documents: http://www.access-board.gov/sec508/refresh/draft-rule.htm#documents

[67] W3C WAI Authoring Tool Accessibility Guidelines version 1.0: http://www.w3.org/TR/ATAG10/

[68] W3C WAI Authoring Tool Accessibility Guidelines version 2.0: http://www.w3.org/TR/ATAG20/

[69] W3C WAI User Agent Accessibility Guidelines version 1.0: http://www.w3.org/TR/UAAG10/

[70] W3C WAI Web Content Accessibility Guidelines version 2.0: http://www.w3.org/TR/WCAG20/

[71] W3C Web Accessibility Initiative: http://www.w3.org/WAI/

[72] WCAG2ICT Task Force Work Statement: http://www.w3.org/WAI/GL/WCAG2ICT-WorkStatement.html

[73] WCAG2ICT Task Force: http://www.w3.org/WAI/GL/WCAG2ICT-TF/

[74] Wolfgang Stuerzlinger, Olivier Chapuis, Dusty Phillips & Nicolas Roussel: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.64.3555&rep=rep1&type=pdf