July 22, 2020 - GUADEC2020

Archaeology of Accessibility

Emmanuele Bassi

Can't hear you... just a cracking sound. Switch to a different microphone. You most likely have multiple ones. Webcam usually have also one. clicking stopped. But no sound yet.

That one looks -- it just muted yourself. But it was okay before. Clicking sound.

[Click, click... we'll get this figured out!]

Sorry about technical issues. We will be soon back on track.

Emmanuele Bassi: Okay. Now can you hear me?

>> Excellent.

Emmanuele Bassi: Yeah. Sadly, BBB decided to use my wrong -- the wrong input. And I couldn't get it to change. okie dokes. So, let's get started because we are already way past.

So, I'm Emmanuele Bassi. I -- my preferred pronouns are he and him. I work for the GNOME Foundation. And I work on GTK. So, let's talk about accessibility.

Accessibility is what we -- well, it's defined as the design of products, devices, services, or environments so as to be usable by people with disabilities. And a companion work to the -- the term "Accessibility" is assistive technology. We will use that term in this presentation.

So, an assistive technology is an assistive, adaptive, and rehabilitative device for people with stabilities or the elderly population.

But who are we talking about when we talk about accessibility? Who benefits from it? There's a common misconception about the topic of accessibility. When people say that an application, web page, must be accessible, the mind goes through certain images. We all know them. Like this was Stephen Hawking on his incredibly cyberpunk chair. Or this. Who is a character from the awesome 1992 movie -- no, 1990? I don't remember, 1992, probably, one of the greatest movies about hacking and social -- hacking in terms of no not computer hacking, but like social engineering. You should watch it if you haven't. Especially you, kids. Talk to your parents.

But accessibility or making an application accessible is definitely a required for input and output devices like these. But the most common consumer of an assistive technology are these two people. These are my parents. Pictured in Rome a few weeks ago. You can notice the masks because of COVID.

They are in their late 60s. They're still fairly active. They go on holidays in Rome here. They -- they also go on holidays on motor bike or they jump on a plane and go on the other side of the planet. You can also say that they are both computer literate. My mother -- when I was a teenager, my mother could outrun -- out-type me on a keyboard. Because she used to do data entry in the '80s.

And the reason why I'm a computer geek is that my father also is a computer geek. Nevertheless, these days, my parents need a lot more affordances when dealing with computers and mobile phones than I do at the moment. So, we all benefit from accessibility.

So, let's face it. I'm 39 years old. I'll be 40 later this year. I have been wearing glasses for the past 25 years, probably more. For longer than that, I have been listening to music of varying levels of intensity and volume with headphones, mostly. So, my eyesight and my hearing are probably not gonna last. Since I've started working as a professional software developer, I've had to improve my posture and switch to different input devices to avoid repetitive stress injuries. And I still have days where my ability to use a mouse is kind of severely limited. My shoulders start to like basically hurt so much that I cannot move it.

None of us are getting any younger. I'm sorry to say. And if you are in your teens or 20s, this is as good as your body gets. The days where you can keep shoving like tons of 16x16 pixel widgets on your screen are not going to last. It's all down him from now on.

So, when I say we all benefit from accessibility, I mean that we are the next batch of consumers of accessible software. Whether we like it or not. We owe it to everyone, including our future selves, to make it work.

So, since this talk is -- it's called archaeology of accessibility, let's start with some history. The accessibility support landed in GTK during the 1.3 to 2.0 development cycle which led to GTK 2 which was released in 2002. So, we are talking 18 to 19 years ago. It was the contribution of the accessibility team. And it was a push needed because Sun was starting to ship GNOME 2 on their workstations as the Solaris test environment. Or the Java test environment. I don't know what they called it. But it was on that.

The accessibility stack consisted of three components. The first one was ATK. Which is a series of abstract interfaces that are meant to be implemented by object-based toolkits. There is AT-SPI, which is a protocol designed to let applications and assistive technologies to talk to each other. And then we had GAIL, which was a GTK module that basically was responsible for turning -- for implementing the ATK interfaces and turning like GTK state into AT-SPI data to be sent over the wire to the assistive technology application on the other side.

So, the charitable reason for this design was that Sun already had an accessibility occupy for Java. And it made sense to try and write something that worked in a similar way. In reality, it seems that very few people knew why the Java accessibility API worked the way it did. And it was rather easy to copy wholesale into a G object world than it was to actually change it.

I mean, if you see the accessibility API for Java, you will notice that there's literally nothing more dot com double than having a hyperlink and a hypertext interfaces that are completely separate. And yeah. That's very like -- very 2000, late '90s.

Additionally, the reason why the whole thing worked off as a GTK module, GAIL, was that the communication channel was because of CORBA. Was using CORBA. If you are lucky enough or young enough to not know what CORBA is, imagine the enterprise version of any reasonable modern IPC mechanism. Then make it even more corporate. And design it so that it requires a centralized body to assign names because of course this thing must work on a distributed network. Once upon a time GNOME worked on that. It's where the object model of GNOME comes from. Since the CORBA implementation that we use in GNOME was kind of big and had weird dependencies and it wasn't going to be made a dependency of GTK, the accessibility stack basically had to live out of tree from GTK. And like theme engine or an input method. And to be fair, it was also subjective to pretty much the same amount of design and security as those two. So, it wasn't great. But, hey. It allowed a product manager to check a box that says "Supports accessibility." so, it's difficult to say if it's bad or not.

Sadly, accessibility is not a checkbox. It never ends. You are not -- you are accessible in this moment or you aren't. But if you are accessible in this moment, the next moment might not be the case. Because accessibility is a process. And we did end up forgetting about it for the following 18 years. And the world has changed. We are not using CORBA. We are using DBus now. We don't want to use out of tree modules. We have built in functionality. We used from X11 to Wayland. And we have sandboxing. I mean, it's kind of drastic. The entire stack was designed for a world that doesn't exist anymore.

With GNOME 3 we managed to port AT-SPI away from CORBA and to DBus. And we moved the code that was in the GAIL module into GTK. But the architecture isn't exactly the same. And it actually relies a lot on thinking that it's X11 and do you have access to any application has access implicitly to the entire user's systems. And to all the other applications that exist.

And, I mean, your applications were installed by a sys admin, rights? You can trust them. It's not like you downloaded them from random places on the Internet.

So, in reality, accessibility in had GNOME needs to change too. It's not just moving pieces around. It needs to be redesigned. Because it's been stuck. And it's also a problem in the future -- it's a problem for the future users because unlike any issue that we have in free software, you cannot ask people that are using the assistive technologies to actually contribute to the stack. Unless the stack is already there and working.

It's... these people -- the people that are using assistive technology cannot perceive what is not available. Which means -- let's say imagine you find -- you have your like -- you're using a free software project and you decide to scratch and enrich, right? Now imagine not being able to access the project. Not the code. Not the issue tracker on whatever. The actual project. It doesn't exist. Because you cannot perceive it, you cannot see it, you cannot hear it, you cannot do anything with it, you cannot manipulate it. So, it's -- that's why you don't go and ask users of assistive technologies, is there aggression in my application or a missing feature? They don't have a point of comparison. If it doesn't exist, it doesn't exist.

So, how do we change it? Well, we have to consolidate the effort from the technological standpoint. We have the ATK, AT-SPI bridge and wherever else we have it. And it's been mostly untouched for the past 20 years. In recent times, new functionality has been added. Mostly to match what web browsers provide. But it's -- it's all over the place. It doesn't really work. We have an abstraction layer that's not really abstracting anything because any time you have to change something, you have to change it in three different places and do the exact same thing. And then from a resource standpoint, we need to invest back in accessibility in both the core platform and the application development process.

So, application developers do not really care about this enough. And are convinced that GTK will deal with everything for them. Which is hilariously untrue. And, of course, even if they do care about it, some people do, and they would be confronted with a pretty terrible API that's basically undocumented and really, really hard to understand. There are no tools or documentation for helping application efforts and guide them into is making their own widgets accessible. Let alone their applications.

So, we also need funding. Since we cannot ask people to volunteer their time to fix accessibility because it's such a comprehensive effort and we cannot ask people that use the accessibility stack to scratch their own itch. Which is what we usually do for everything else. We will need help -- also in terms -- not just in terms of contributions, but also in like monetarily. Because we need programmers. We need designers. We need testers. It's not a trivial thing.

So, one of the things that happened this year was that the GNOME Foundation kindly gave me the direction of working on the accessibility stack in had time for GTK4. It's one of the steps that we need to get into the stack before we can release it. Release a stable version. So, I have been working on this for the past 6 months. There was a good deal of archaeology in terms of understanding how the thing was put together so we can actually fix the mistakes in the next design instead of repeating them.

And one of the changes that I have been working on was trying to minimize the API needed for application developers. And then reintegrated most of what was abstracted away for the benefit of the platform that doesn't exist anymore back into GTK. So, in the future, you won't have to look at the GTK and PR reference and switch to the ATK API reference in order to implement the accessible side of things in your widgets, in your application. Because ATK doesn't exist anymore.

So, we still want a certain level of abstraction in order to be portable to other platforms. Oh, yeah, of course, I kind of forgotten to say that. But the accessibility support in GTK3 was not portable. It never supported Windows or macOS because the abstraction layer was meant to abstract away the fact that you're not -- that you're not using Java instead of abstracting away the fact that you're running on a different platform.

We don't want an abstraction layers that based on code. We want an abstraction layer that is based on concept. And since we have been using the web for our drawing API in terms of CSS, we thought it was an interesting idea to look at what the web was doing and what the web is doing is using these specification called ARIA.

Which defines an ontology of accessible roles. What a UI component is. Sorry. Yep. What an element does, sorry. It's based off elements in your UI and every element has a role, which is what the element does. So, it's a checkbox. It's a button. It is a switch. It is a speed button, whatever. And also whatever -- what an element has in terms of attributes. They are properties. So, it's visible or not. It's priced or not -- sorry, not priced. It has a label or not. The relation between elements in case you have a widget that is pointed out by another widget in terms of a label or like a box or whatever.

>> 5 minutes.

Emmanuele Bassi: And you also have a state which is the button checked is the -- is it act every? Is it whatever? So, we have -- sorry about that. We have in the toolkit at the accessible occupy, which is an interface for widgets. The accessible role, the accessible property relation and state. That -- those map to the ARIA concept. And then we have the AT complex, which is underneath it. And it's completely opaque and you don't have to actually see it. So, you have simple API that sets the accessible role in the class. Updates the state. If it's pressed or not. Binds property.

If you change an adjustment on a scroll bar, you resynchronize everything. Or you relate another widget to a label, for instance. And for applications, you have to follow the rules. ARIA has five rules. The first one is reuse existing accessible UI elements. Do not try to do your own thing. Not in terms of do not try to make your own widgets. But do not try to change to add stuff from scratch. Try to reuse existing concepts, roles and properties and states.

The second one is do not change the semantics of a UI widget -- sorry, an accessible widget. If a widget is accessible by pointer, it should also be accessible by keyboard. Don't try to make something that is only accessible by pointer. If it's focusable, do not hide it from the accessible stack. And if entirely possible, try to add an accessible label either directly using the accessible label property or setting the labeled by relation. For applications, we should have the gtk-builder-tool, allows you to port from GTK3 to GTK4 your UI description files.

We should also have a validation mechanism that lets you ensure that you're actually adding the accessible information to your UI. We have a TestATContext which is automatically used when you export an environment variable inside your test suite. Which is used locally so it doesn't try to poke stuff into the actual accessible stack. And it's used to test things with these APIs. Which is the gtk_test_accessible_assert. It's the same as the accessible cert, but ensures that accessible things are updated whenever you change something so you can verify things work. The current state is optimizing the API. It's almost done. It's literally -- it's very small had. There is not much to finalize here.

I'm porting GTK widgets that are shaped by GTK so that they are described by this occupy. And every time it changes state or changes property or use the API, we update the accessible state for you. I'm also writing the new test suite. Because now it's different. And we can verify that stuff works. Instead of just dumping the accessible state on a log and then doing like text search to verify that stuff matches what we thought you should be. Which, spoiler alert, never worked.

Documentation is in progress. There is a -- an introduction and I'm trying to port the ARIA authoring guide to GTK so that we can piggyback on that. And, of course, I'm implementing the AT-SPI backend because otherwise it won't do anything. How can you help? You can help by writing tests in GTK. You can help by writing documentation. Look at the API. Look at how it's working and how it -- it seems it should be working and then poke me and tell me if it's not what you expect.

If you are experienced with Windows and macOS, help us support on the GNOME platform. And we are planning to do some changes in AT-SPI on the lower level occupy mechanisms so that we can have sandboxed applications isolate the accessible -- the assistive technology from the technology so that we don't -- A, we don't Spam your accessibility bugs for the entire session. And B, we don't basically leak everything on a sured resource. Thereby completely neglecting the point of sandboxing applications. And if you want to contribute to this, and you don't have any skill in terms of the four points before that, then definitely funding is welcome for the GNOME Foundation so I can keep working on this and we can keep working on maintaining this new stack.

So, shoutout to the accessibility team for keeping the Torch lit over the past almost 20 years. Thanks to the GNOME Foundation for funding this. And thanks to Hypra which is a company, a French company. And they sent a bunch of engineers and users to our GTK access in January and it was incredibly useful to understand how this entire stack is consumed and put together. And we have a plan to move it forward now. So, you have any questions, now is the time.

>> Thank you. Thank you Emmanuele. You can find the questions if you click on shared notes and open the link which is track 1 Q&A. There's a lot of questions for you. We will run --

Emmanuele Bassi: Okay.

>> We will run a bit over dinner time. But I think you can easily use that because I think this topic is very interesting. Carry on.

Emmanuele Bassi: Okay. Okay. I'll will super-quick. So, how can we promote accessibility development to involve new hackers? This question is because some apps don't work with technology like Wayland. Yep. So, the main problem is the accessibility stack is really complicated right now. Which is one of the reasons why I tried to simplify it as much as possible and put it inside GTK, inside of spreading it across 25 different layers of the stack. And a bunch of different libraries. Ideally, the Wayland side of things is going to be taken care of. The main -- by this. Because we are not going to be using like weird global state poking or anything like that because we cannot do that in GTK.

It will require some changes inside the assistive technologies stack as well. The maintainers or Orca and other ATs are well aware of this. And they're moving towards using the compositor, for instance, instead of using like X11 API to -- to basically snoop all your key bindings and everything. Instead, they will talk to, for instance, GNOME Shell and then we'll ask, please give me the -- the coordinates of the pointer. Or please give me the window that is currently focused. And -- or please give me the -- register this key combination and wake me whenever it's pressed. Or things like that. So, the idea is that we will need to have a set of changes at the low -- lowest levels of our stack in terms of compositors and in terms of other technologies.

And this is where talking to other projects is gonna work better than -- than before. So, there is a shared interest across the desktop environment of Linux. There is an accessibility working group that is not incredibly like -- there's not a lot of -- of work going on right now. But we -- we are kind of trying to get it jump started and get it working. So, yeah. I'm currently I'm trying to get GTK4 out of the door. So, I'm focusing on toolkit. But there's a lot of stuff to -- a lot of work to do on compositors. A lot of work to do on assistive technologies. So, getting those ready is another part of the -- of the puzzle. Another piece of the puzzle.

How do you think it would be possible to marry flexibility of design application accessibility? So, one of the things that ARIA gives us is the fact that the web is terrifically flexible. Which is not always a good thing. Recommend using the DOM every time. But if you have relations between UI elements that are not mapped by the DOM because they are [terrifyingly] -- and you have a transient or a layered UI or something like that. Then you have accessible attributes that you can use to link things together. And GTK has that as well now.

So, the entire accessibility stack is meant to describe what you have on the screen. Which means state, properties, relations. In terms of visual -- in terms of UX design, we should have an accessible language that is matched to our visual language. If that makes sense. For instance, you shouldn't be using tooltips to describe your UI to an assistive technology user. Because tooltips are based on the assumption that you are, A, moving your pointer, and B, you can see the evidence that it -- they belong to. If you're reading out that disruption, it's not a good -- it's not a good idea to just read out that description. You need a separate description. Something that makes sense when read out loud.

So, I think from a UX perspective, we should have an accessible -- accessibility language that we use. And we conform to. How much of GtkAccessible can be lifted into Mutter? Most of it, I think. It's pretty -- it's just data types. It doesn't do a lot of complicated things. The overall idea is that you take the widget state and you just shove it into the accessibility -- the accessible state.

And then delegate object that sits underneath it all which is the context. Transforms it into something that can be used by the assistive technology API that is on your platform -- that currently works on your platform. So, AT-SPI on Linux, for instance. If you think that this resembles the current GSK rendering API, you would not be wrong. Because that mechanism kind of serves us well. So, I decided to basically reapply it.

After that, I think it's probably gonna be useful. Even the AT context, the AT implementation is probably going to be useful. I don't know if Mutter is going to be able to use it because it's GTK. But it can definitely be lifted in terms of design more than implementation, probably.

Does this API encompass situational accessibility needs. Like when the user can't use both --

>> There's not one minute to go. So, if you can quickly -- because we need our time for captioners as well.

Emmanuele Bassi: Sorry, sorry, sorry. I don't know. I don't have an answer for that, I'm sorry. We are gonna be enabling accessibility anyway by default. So, if you toggle it, it should work. If GTK, which is implemented a new occupy, does that mean one gets accessibility for free for bullet-in widgets? Yes. What would GtkBuilder XML for the new occupy look like? It would look pretty much what it looks like today. Except you wouldn't be creating child widget -- child objects. Which is -- yeah. Not great.

You would have an accessibility -- accessible like element and then you would get the role and properties in the XML. I don't think we want to add roles -- a role attributes like ARIA does because it makes person the XML -- yeah. Not great.

>> Okay. I think that's all we have time for now. Thank you, Emmanuele.

Emmanuele Bassi: Thank you very much.

>> And now we will have the break til18:45. Take a -- make your dinner. Make your coffee. And, yeah, thank you, Emmanuele. See you after in less than one hour.

[Break!]

>> Can you hear me?

>> Copy --

>> Great. Thank you.

>> There's echo.