State of the Shell

Carlos Garnacho, Florian Mullner, Gary Stafford, Jonas Adahl, Jonas Dressler

>> Sorry, got mixed up with times. We start now with Carlos Garnacho, Florian Mullner, Gary Stafford, Jonas Adahl, and Jonas Dressler, with State of the Shell, please.

Georges Stavracas: Hi, everyone. Hopefully you can hear me. If not, I'm sure you're going to let me know in Rocket Chat.

>> We can hear you.

Georges Stavracas: Okay. Let's wait for the other presenters to join as well. Which I guess they already did.

Jonas Dressler: Hello, can you hear me?

>> Yes.

Jonas Dressler: Great. Let's see if the webcam works. Yep.

Georges Stavracas: All right. Three more and we are good to go. This is probably the presentation with the most -- the highest number of presenters during this GUADEC. We have one left. Let's go! Let's go! Looks like we're missing Jonas Adahl. Let's wait for one extra minute for the last one to join us.

>> He's coming.

Georges Stavracas: Okay. Fantastic.

>> I checked the notes, it's definitely the presentation here. Here we go.

Georges Stavracas: There you go.

>> State of the Shell, please.

Georges Stavracas: Let's start. So, welcome, everybody. This is yet another instance of State of the Shell. As was said, this is an acquired taste kind of presentation. Hopefully this one's going to taste very good. So, yeah, let's get started because we have a lot to talk about. And Carlos, please, your turn.

Carlos Garnacho: Yeah. So, I'm going to talk about input. And yeah. Next slide. Yeah. So, in 3.36, we introduced ClutterSeat, which is the actually first step towards a number of input refactors that we introduced. We moved a lot of codes from Clutter so it lives together with the rest of the backend code in Mutter. Which means that actually creates for the most part backend development. And the backends stay with the backends. Which is good to have all the backend code together. So, yeah, ClutterSeat is the first refactor that we were allowed to do with this.

And, yeah. Well, it's not that much exciting on itself. But it's more about what it's going to get built on top of. Basically. Yeah. So, ClutterSeat is an owner of input devices. It has others that we can move forward. Yeah, so, next slide, please.

Yeah. So, there's been some work on the input thread. And, yeah, well, this diagram shows very nice with the arrows. They look -- they are in the direction they go and whatnot. But actually most of the work has been in all of quite some bits in other code that was put in different ways into the backend code. The final goal of this is to make the input thread the new circle that you see there, the blue circle there, to be actually, well, to be able to -- to figure out from one point of position and the number of events what would be the next point of position without external agents, basically.

So, the easy part is sending events to the UI. Which is the thread. And, yeah, well, this is just the queue of events. It's nothing really -- really complex on itself. So, next slide.

Yeah. This is more of the -- more where the main refactors happen. It looks similar. There are a number of interactions between the UI thread and the input bits which will be virtual devices, warping pointers. Warping pointer is when that pointer moves from one place to another in the screen. But there's also a number of others like, yeah, the Py squared, modify square, pointer barriers or pointer constraints or pointer lock-in. Yeah, there's quite a few of these. So, this has been mostly strings. So the interactions are easier in both threads. And, yeah, they will hopefully pay out. So, next slide.

This is where we involve KMS and whatnot. KMS could be the -- the lower level parts doing the cursor play and position changes and whatnot. So, yeah, in this case, well, actually, it could be the other way around. Because it relies on the same mechanisms that the device queries in the UI thread. So, the final result is that the KMS builds are aware of the final cursor position at that point of rendering and they can use it and upload the new state to the cursor plane.

And next slide, please. And then there's the actual painting of the cursor. There's the UI thread is going to take care of this. It's, of course, it needs to be on a position and figure out what makes sense based on that position. And then do the updates. Eventually, it will nice to -- to make the cursor be not wake up yet that can be pushed to the sprite holding all the animation details like the clock, a spinner and whatnot. So, it's all -- happens in the game as in the UI thread.

At the moment, all updates and all updates, for example, to the next frame in the -- in the cursor animation could be happening in the UI so far. That's -- yeah. That's not yet to change. Next slide, please.

And the potential benefits. Well, they will mediate to pie in the sky. But, yeah. There's some things that are getting closer thanks to this. So, yeah, well, they must admit a benefit is that we don't miss libinput events. This happens that the shell would be blocked. The UI thread would be blocked. Say some stanchion doing something. And we will input them in internal queues. So, the tasks if that happens. And the next one would be the blocked pointer cursor. No, no, no. Sorry.

Previous one, please. Yeah. So, no blocked pointer cursor. It would mean that the UI thread is not actually involved in updating the cursor plane position. So, it could happen independently of the -- of the UI thread being blocked or not. It should result in more reactive pointer motion and whatnot.

It also helps in better handling of high frequency devices like the one with the mice and whatnot. So, we currently are doing motion compression in the stack. And this design actually pushes the -- or makes it more mandatory to put this event compression higher up in the stack. Which actually means that we could, for example, handle, or deliver Wayland events at full rate without, but only handling compression motion events before handing them to the Clutter plates. So, the shell, it would update at 60 frames per second or 75 frames per second. And you are able to still get the full motion that -- the full stream of motion events to the clients.

And another thing would be that it would allow a better reuse of the cursor plane. Yeah. Accessing devices and whatnot. The software plane was the last resort. It would be nice to move them between devices. For example, if you have a pointer pusher which is currently rendered with software. So, this helps. Yeah, next slide, please.

Yeah. And the button, we have the touch mode. Which, yeah, it's basically a low-level toggle which will allow the different UI to adopt to this change. It, for example, can -- there's a bunch that should be -- at some point to allow the rotation lock to just disappear if you put the keyboard on a two-in-one device and whatnot so it behaves like a doublet when it's one, and a laptop when it's a laptop. And that's exciting on top of that. Yeah. This is my bit.

Jonas Adahl: So, this will be about the changes we have done to the frame clock that drives the -- drives the rendering of the shell. Next slide, please. So, first, a bit of history of how it looked before. Clutter, as you know, started out as an application toolkit. Every application would create the one or more stages which more or less is the same as a window. And we would have a single frame clock that painted -- that drove the windows in a kind of funny way. It worked fine enough for this. For years. Even when we started to use Clutter as a compositing manager for the X11 golden days. It worked almost just as well because with the XLM user manager, we have user clutter, and we would still paint to a single window and we would just care about driving that window. Not too much different from a regular X11 application. Next slide, please.

Then things becomes more complicated as we enter the Wayland era with the native backend. That talks directly to the kernel mode instead of having the server do that for us. It works quite differently from an application because we have more responsibilities in dealing with the hardware beneath. So, after having forked both code and -- to our own version, we started to split the rendering passes up into different pieces. Initially, we did this by rendering in certain areas of that -- of the stage separately. And we called those areas "Views."

If you have multiple monitors, we would render the area they occupy individually. We got, for example, monitor rotation using OpenGL. I do this, and fractional scaling by doing this. Eventually, we took it to the next step. And instead of just speeding the region up, we actually did it per-monitor or actually per-CRTC which is -- it's a component in the -- in the graphics card that drives the monitor. Next slide.

Until quite recently, we still used the good old frame clock from the application days of Clutter. But it didn't really fit the way we -- the way we want to implement the display server that has to talk to different monitors. That render different clocks. So, how it works, it's more or less a monitors driven by a CRTC that has its own clock. Usually the clock ticks at a certain reference rate, more or less. And configuring in those things. It will run at that interval unless you have a version of refresher monitor. But ignore that for now.

These different monitors, they render a different -- they update at different times. Even though they have the same clock frequency or they have different clock frequencies. There are 144 hertz and 60 hertz and 75 hertz and they simply have -- they render at their own pace. Next slide.

So, here's an attempt at an illustration. You will see that there's supposed to be red lines saying where we render. Let's see. They seem to have disappeared on the converting to PDF. But I will try to explain where they are supposed to be. At the beginning of each blue arrow, there is a frame that is drawn. And at the end is when the frame is presented -- or the monitor is updated. And that first two blue arrows we would render at the -- let's say the first monitor has 75 hertz update rate, and the lower one has a let's say 45 or something. And we render at the upper monitor rate for a couple frames, three frames. And then something would change on another monitor, the slower one. And more or less the result that we would end up with is that both monitors will render at the slow monitor's rate because we have to wait for the second monitor to finish before we can render the next one. Which if you had a fast and slow monitor, the fast monitor would render at the speed of the slow monitor if the slow monitor had any updates.

And that's kind of all right if you have a fancy monitor that you can't really make use of. Next slide.

So, that single frame clock that tried to juggle all of these kind of scenarios. More or less, for making any significant progress in this area. As you could see in that previous slide, they were waiting for each other to paint -- to paint it. And it was kind of complicated to try to juggle different monitors with different capabilities and requirements and so, it was designed at one point to just split it up into multiple ones. Let's take the next slide.

So, we created a completely new frame clock. Called it the ClutterFrameClock. Already, we had split up the rendering or the painting of the stage and the screen of all the screen content into different dedicated views. Each view would correspond to a monitor. So, we already had that in place. So, we put one of these frame clocks in the view. And then have one frame clock drive a single view. It's pretty much how it works. It makes the clock much simpler because it doesn't need to handle as many things at the same time. Next slide.

Here you can try to put it in your head again how it's supposed to look when the red squares are there. The fast monitor now doesn't care at all about the slow monitor. Just draws as its own pace. And the slow monitor starts to render its triangle conf thingy and does it also at its own pace. So, next slide.

That's not enough. Because Mutter is not only a tool cut for compositing windows and painting on to monitors, it's also a compositor and user interface used by GNOME Shell to implement all the fancy animations and widgets in Chrome and everything. It has an animation framework. In the past we used something called Tweener. But since a few versions ago, we used ClutterZone, a pretty good animation framework. But there was one culprit for this. And that is that the framework assumed that there was only a single frame clock that drives everything and it will just automatically attach to it. And all will be well. And we'll just get the animation frames and just animate on. Next frame.

Next slide. So, we would have to make a connection. Each of the animations would be driven or so by an object, a Clutter object called timeline or transition. We hope to have these objects. If we tie one of these objects to an actual widget on, for example, a window or a button or a menu or something like that, we would tie animation to an actor. An actor would already know on what views it would be painted on or was painting on. And now with a new frame clock thingy, we -- each view had a clock. And voila! We could find a suitable clock by looking at the animation it was tied to. And by that, we would just -- next slide. Yeah. By that, we would have animation objects that would just automatically animate at the speed that they were actually assuming you put the -- made the right connection to the right actor.

And that required apex changes in clock monitor. but GNOME Shell pretty much completely wraps all the animations into a few helper functions or a helper object. These so-called "Ease" functions take active properties and put a fall and then create a transition of a smooth movement between the two values. And then adjustment is similar to that. We will just fix these two -- these two kinds of animation methods to automatically find the right actor. And then it just kind of worked. If I remember correctly, there was just two lines of JavaScript code that was needed after all of those in GNOME Shell. So, for convention goers, won't need too much changes with the Clutter animation framework. It's hard to read, but on the left one, 75 hertz monitor that I run GX, and on the 60 hertz monitor on the right side. And they run at their supposed to run speed. And that's the 60 hertz, yeah.

And that's the whole goal of the split up. Yeah. Next slide. And that I think -- that's it for me. Thank you.

Jonas Dressler: Next slide. This is about clutter's layout machinery. That machinery got quite a few improvements last year of the cycle. And the most important was the introduction of the shall row relayout mechanism, done by Daniel. So, thanks to him. The way this works is like the basic waylayout works is it goes from top to bottom, usually. You start allocating actors. And the top-most actor, which is usually the stage. And then you go recursively down to the actor which shoots the relay out. So, you basically always have to allocate the actors up the chain to allocate an actor down the chain. And that's quite expensive.

And the way we -- what we found was that there is a way to improve this thing because there's a no layout flag. And the no layout flag is a flag that tells clutter that it doesn't need to be activated by the machinery. Clutter can make the assumption that this needs to be made by static positioning and can take a shortcut. Can basically allocate directly instead of going to the stage and going down the whole hierarchy. And this has made -- this has been quite an improvement for performance. Because it has made allocating actors like windows significantly faster because we no longer have to start at the stage with allocating.

Then we've been planning various bugfixes in Clutter. Those mostly have landed because we've found bugs when implementing new layout managers in the shell. So, yeah, and those were quite a few bugs. And they're mostly fixed now, I guess. Then we have removed ClutterAllocationFlags. That was a change. So, ClutterAllocationFlags were basically used by Clutter during the allocation cycle to pass on animation to children which were allocated about the allocation cycle. And there were two of those flags. They were both pretty much unused and one of those flags was called absolute origin changed. Now, this flag was used for notifying children that an actor's origin changed. So, you would have position of an actor. And if this actor movers, you would use this flag to notify all the children of the actor that the actor has moved.

So, the problem with this flag is that to notify those children, you would have to -- so, because it's a flag which gets passed to reallocate function, to notify all those children, you would have to notify all those children. If some children don't have to be allocated, you would have to allocate because the flag would have to be passed to the child. That was expensive too, and we can completely avoid it because we can get rid of this flag. So, the next point that was quite a big change in the layout machinery was the enforcing of invariants.

So, basically, Clutter defines a few invariants for actors which are managed and set by Clutter itself. And one of those invariants is the mapped state. Now, the mapped state, if we're saying whether an actor is getting painted or not. And the way this works is -- as soon as the actor is on stage, and its parents are visible and the actor itself is visible, the actor is mapped.

And that's a trick. Like if the actor is not mapped, if the actor is unmapped, we can't obviously know it's not going to be painted. We can skip allocating this actor, for example, or do other optimizations. And we're doing a better job enforcing those invariants. So, that basically means that we found a few bugs through this change. And yeah. It should make debugging a bit easier because it should be easier to find issues with that now. So, let's go on with the next topic.

Georges Stavracas: Which is the DMA-Buf Screencast. That's my turn. I hope you can hear me. I'm sure you will tell me if I'm not. I will try to be quick here, mindful of time. And especially the DMA-Buf Screencast is a way to avoid copying pictures over different memories. In this case, we are talking about the GPU memory and the regular RAM memory.

So, essentially, when you're screencasting, what usually happens, and this is a very small slide deck here. But I guess you can all see. So, suppose this is your monitor. You're seeing GNOME Shell here, beautiful, as it always is. And then you want to screencast your monitors, right? So, you want to record something. So, when you start recording, the Mutter starts copying what's available on your screen to your RAM memory. Which is over here. And that's extremely expensive. After that, Mutter hands the application who is recording this -- the frame that you just copied over. The application does whatever it wants with the frame and then it's destroyed. That's usually how it works.

With DMA-Buf, we get a different cycle here. So, instead had of moving the frame down to RAM memory, we create a copy on the GPU side of things using the operation which is extremely fast compared to downloading it to RAM. We have to create a copy because we don't want to lock your monitor frame until the application is using it. So, we create a copy. We copy it over a different buffer. And we get -- we give the application a file descriptor, essentially a number, an ID, where this is in the GPU. The application can do whatever it wants with this file descriptor. It can even download it to RAM memory. But at this point, we don't care because Mutter is already not doing that.

Which means you can still have a smooth Screencast experience with Mutter. And then we have a recycling mechanism. So, instead of destroying it right after the application tells it's done using it, we can reuse this and copy over the next frame or 16 frames after that. And keep reusing stuff until the stream is done. And we can destroy everything. So, this is the DMA-Buf Screencast. It's a much more pleasant experience when screencasting. Next topic is paint nodes.

So, paint nodes is basically the core of what we -- what was going to be Clutter 2.0. And it's composed basically of a 2-step rendering process. So, first, we go there all the UI elements around and ask them, and collect the paint nodes. Which is -- which are basically, operations on how we want to draw. And then after the collecting phase, you go to the painting phase where we go through each node. And ask them to do what they want to do. So, this is a pretty small self-contained example here of a paint node tree. So, we start with the root node and then we -- I don't know. We want to paint some color here. And we want to paint something. But this other something is clipped. So, we paint inside a very limited region. We paint a texture here. Then after that, we want a color painted on a layer node. Basically, a GIMP-like layer where you can combine them later, or not. This is a very small example of how paint works -- paint nodes work. And this is where we're going in the direction of.

This is allow us to add some cool stuff. Cool features on top of other rendering process. So, at this point, right now ClutterEffect is fully or , offscreen effects is ported. Shell is there to look at the paint nodes. It requires more work because we couldn't express all the operations that were required to paint a blurred version of something one-on-one paint node. Used more API, and voila, it works.

Next step is adding a way to profile regions of paint nodes. Paint nodes subtrees. So, for example, if I want to know how long does it take for me to draw this subtree here, I'll be able to create a new node. A profile node that will check how long it takes to draw that.

And then after that, we can be working on Shell entirely. And hopefully make all of the rendering completely shader and GPU-based. And yeah. So, switching gears a bit. So, let's talk about GNOME Shell now.

Probably most or all of you are already using it. But we have a new lock screen. I can't live demo it. But I'm -- there you go. It's very -- it's a very -- it's a more beautiful experience in my personal opinion. It's long in the queue to be done. And it has landed in the GNOME 3.6. And this is part of what happened in past cycles. What's new for GNOME 38 is a customizable upgrade. I'm not sure if it's a very good idea for me to try to screencast this right now to share it. I mean, I could do that because it would prove that DMA-Buf Screencasting is fantastic. But I'm afraid of doing that and crashing everything.

Anyway. Here you can see I'm on the second page of the icon grid. It's completely out of order. It's in the precise order that I want to. And, okay, I'm being -- I'm being -- I have been asked to live dangerously. So, let's do this! Hopefully you can see my screen. I'm sure you'll let me know if you can't. There you go. I can see my screen. That's a good sign. So, there we go. We have a customizable app grid here. As you can see, this is my personal grid. I've adjusted it with the applications I use the most. I can drag something here. For example, GNOME Box is in this position. It's probably not as smooth as I'm seeing because I'm Screencasting. But anyway, I can do this. For example, get this folder, drop it here. There you go. So, yeah. This is good. The customizable app grid for you.

Back to the slide deck. I'm handing it to Jonas to talk about the dialogues redesign.

Jonas Dressler: Yeah. So, something else that we landed in GNOME 3.36 cycle, was the redesign of dialogs. And we did that because there was an open issue, doing them for quite some time by designers. And they rightfully complained that the dialogs were quite inconsistent. The design of different -- over different dialogs. And yeah. Some of them also looked outdated. They have been wanting to redesign them for quite some time. And so, they did mockups for every single dialog. Thanks to them. That was quite a lot of work. It turns out there's a lot more dialogs than you think in GNOME Shell. And landed in GNOME 3.36. You should have seen it. In case you haven't, we have a few screenshots in the next slides. Gorgeous. And thanks. Yeah, right. So, on the left are the old ones and on the right, the new ones. And we have even one more slide. Yep.

So, in my opinion, that looks way better. But, yeah. Next up are the changes to workspaces and the overview. So, in the overview we basically like -- we have this -- those windows which are shown in the view. And we have overlays on top of the windows that show the title and the close button for those windows.

And previously, we've -- we've implemented them -- those labels completely separate from the windows. So, they were positioned statically in a separate class and we've basically had to track the window -- the movement of the windows. And everything the windows did and replicate that thing for the overlays so they stay positioned correctly. Now, that thing led to quite a few interesting bugs. I guess most of you have already seen the bug with the titles of windows being either too small or too wide in size. And the label doesn't really fit or something. So, that is hopefully fixed now.

Yeah, we've also moved the code of the overlays into the class of the windows itself. So, that should hopefully be a nice cleanup. And we've also switched to positioning those overlays using ClutterConstraints now. That's the new way you're supposed to do things in Clutter. Which has been around for quite some time

And we're using if for quite a few things. It's neat because we can tell Clutter to bind the position and the overlay to the actual window, and Clutter does the rest for us. So, we basically no longer have to care about those things. Next slide, please.

Yeah. This is about the layouting of the windows in the overview. And it basically was the same thing for the windows as it was for the overlays. Which also positioned those statically. We had set like a fixed X and Y position and a fixed width for the windows. And the old code was quite complicated. It was quite intertwined with different classes and stuff. And, yeah, so, it was not great. And we've now changed this completely to be positioned using a custom Manager. If you want to do the layout. And making it go to the code where we can theoretically reuse it parts of it elsewhere. So, if we need to. That's it, I guess. Yeah, next.

Florian Mullner: Okay. So -- okay. So, this one is on me. Next slide, directly. So, we got better extension support in 3.36. What we see here is new-ish extensions application. It's actually based on a helper tool that has been in GNOME Shell forever. But it has been split out into a backend part that shows the actual -- like the actual extension code. And a redesigned frontend that has been split off from GNOME Shell itself and can be split separately. And it's, of course, available on Flathub as well.

You see the orange markers. Those mark extensions that have updates available. Because after 10 years, we finally are porting updates. Yay! So, next.

This is what I commented on the previous image. There's also now a CLI tool to manage and interact with the extensions. It's mostly tied to developers. But if you are using the old GNOME Shell extensions cool, commandline tool, the one thing has batch completion. So, it's a huge difference in practice. Next, please.

There have been some improvements to the calendar drop-down. And there are more to come. So, what we see here is there were some refinements in the weather sections. Events have been split off from notifications and moved, well, to the calendar. Which is kind of obvious. And we now have "Do not disturb" functionality that was always there. But it has now like a proper control inside GNOME Shell itself. Next, please.

And there have also been like the backend used by GNOME Shell to interact with Evolution data server has been largely rewritten and should be like a lot faster now. And there are like -- we have like an ongoing Summer of Code Project which will hopefully give us like new and better -- better notification presentation. Go to the lightning talk on Monday. Next, please.

And, of course, there have been like a lot of smaller changes over the year -- the last year. Some of the notable ones are that -- that we have like better -- better support for Systemd user sessions. Which basically mean that every process that is launched by GNOME Shell is placed into a separate scope. Which will allow for some like resource control in the future. There has been like a talk specifically about this topic by Benjamin Berg two days ago. Go watch it.

The next one is parental control support. Which -- which is like a new panel that is being added to GNOME control center. And if applications are affected, they no longer show in GNOME Shell if parental controls are enabled. Again, there has been a talk about this earlier by Philip Withnall. And the recording should be out or should be out soon. And another, like, minor improvement is that the GNOME Shell server that provides color picking now previews the color under the cursor. So, if you look at these -- who controls the red dot? Oh. Yeah. So, the dot under the cursor there does actually show the color that is underneath the cursor and that will be picked when you confirm the selection. Just like a cute little UI hack. Okay?

Next.

Georges Stavracas: I guess this is all that we have. Are we on time?

>> Yes, you are really on time.

Florian Mullner: One minute left, I think. One minute left.

>> Yes, you can start your guitar play.

Georges Stavracas: So, I heard that there's gonna be a social event. We can play there.

>> So, you can switch, yeah, to the Ether Pad for the questions. There is a lot of them.

Georges Stavracas: I'm actually answering the first question, would you like to play some music at the end?

>> Sorry, please go on.

Georges Stavracas: The next question here, probably for Jonas Adahl. Do all K MS drivers support separate scan-out of the cursor?

Jonas Adahl: Most do. In practice, all the ones that we actually use support a kind of like a cursor plane. Some drivers support not a plane, but a way to set the hardware cursor. But... yeah. Most do.

Georges Stavracas: Okay. Our next question is probably for me. DMA Screencast. Doesn't the app need to download to RAM anyway? The app doesn't necessarily need to download the contents of the buffer. It can import the buffer using EGL and not move it away from the GPU side. But one of the cool things about sharing the DMA-Buf is that even if the app downloads the buffer from the GPU, it doesn't really affect Mutter like it used to. So, it's -- the app's gonna be slow probably when it's gonna skip some frames. But Mutter itself, it's gonna stay untouched. Hope that answers the question.

So, next question is, can somebody elaborate on why blipping is required on the GPU memory? Can't we just reuse? I said something about "We can't unlock"? the point is we don't know. When you're Screencasting, we can't just let the app -- we could, actually, it's not a great idea. Have access to the monitor contents. Because the monitor contents can switch if you're drawing something different, if you move your cursor or if an app updates. Anything can happen that makes the monitor change the contents. And we -- we don't want -- we don't want the application who is reading to get the monitor right in the middle of that, right?

And most importantly, we don't want Mutter to have to wait for the application to finish until it can switch to the next string. So, we create a copy of the buffer and then Mutter can continue rendering while the application has a fresh copy there for it to use. It can take as much time as it wants. It can do whatever it wants. It can download, import, whatever. And Mutter can still continue doing its own thing in parallel. That's a big reason.

Okay. The next question is not really a question. But a huge thanks for the blog entries on the shell-development blog work going on. It's been really helpful and I've learned a lot from it. So, that's great feedback to hear. Thanks for whoever said this. Yeah, we're happy to do that. If you don't know what we are talking about, so, we have a blog! It's blogs.gnome.org/shell-Dev. Fresh news directly from those who make the changes. The idea here is that you can have a reliable source of information when it comes to Mutter without overexaggeration or under-selling the changes that we're doing.

So, yeah, do read it. Share it. Follow the blog. Share it on social medias. We are trying to stick to a monthly development schedule of the development reports. With a success rate of about 80%. Which is pretty good. So, yeah. Thanks for this feedback. Anyone want to say anything? Or can I proceed with the questions? okay. I'm going to take that. Proceed, please. What's the plan for supporting restarting a Wayland session without losing running apps?

Jonas Adahl: I can take that. There's one being prototyped by Katy which is kind of like a session -- windows session management Wayland protocol which means that, for example, GTK would ask if it sees that the server is restarted or crashed or whatever, it would try to reconnect and ask itself to have its previous state restored. And GNOME Shell would have to store some kind of state in someplace outside of its own process memory that can -- it can look up again to restore any application that it wants to restore itself. That's the one approach that's currently being explored for this.

Georges Stavracas: Thanks. So, we got a more -- we got a bit more questions. Do we have time for them? I guess nobody's complaining, so, I guess we have time for them. Any plans for improving extensions updates story? I guess that's for Florian.

Florian Mullner: Yes. But I'm not sure I understand the question. If it means never making any changes that could break an extension because that's impossible to happen because extensions simply have access to all the GNOME Shell JavaScript code. Or if it refers to like the newly-added updates where GNOME Shell like checks, like, every day the installed extensions and if there's updates available and downloads them. I mean, if there are like particular improvements people are interested in, please file issues on GitLab. But so far it works as it's intended to work.

Georges Stavracas: I would just like to mention that there's going to be an extension reboot talk I think tomorrow, right? So, make sure if you're interested in this topic, watch it. That's by Sri.

>> I think that's all the time we have for the questions. Please, you can easily respond and ask questions to the team in the chat. Thank you, thank you very much for this presentation on the Shell. We will be back in about 8 minutes -- no, at 17: 45 now.

Georges Stavracas: Thanks, everyone! Enjoy GUADEC!

[Break]