APERTIS

The next-gen Apertis application framework

Contents

As a platform, Apertis needs a vibrant ecosystem to thrive, and one of the foundations of such ecosystem is being friendly to application developers and product teams. Product teams and application developers are more likely to choose Apertis if it offers flows for building, shipping, and updating applications that are convenient, cheap, and that require low maintenance.

To reach that goal, a key guideline is to closely align to upstream solutions that address those needs and integrate them into Apertis, to provide to application authors a framework that is made of proven, stable, complete, and well documented components.

The cornerstone of this new approach is the adoption of Flatpak, the modern application system already officially supported on more than 20 Linux distributions[1], including Ubuntu, Fedora, Red Hat Enterprise, Alpine, Arch, Debian, ChromeOS, and Raspian.

The target audiences of this document are:

- for *Product Owners* and *Application Developers* this document describes how the next-generation Apertis application framework creates a reliable platform with convenient and low maintenance flows for building, deploying, and updating applications;
- for *Apertis Developer* this document offers details about the concepts behind the next-generation Apertis application framework and a high level implementation plan.

[1] https://flatpak.org/setup/

The goals of the next-generation Apertis application framework are:

- employ state-of-the-art technologies
- track upstream solutions
- expand the potential application developers pool
- leverage existing OSS documentation, tooling and workflows
- reduce ongoing maintenance efforts

The next-generation Apertis application framework is meant to provide a superset of the features of the legacy application framework and base them on proven upstream OSS components where possible.

Creating a vibrant ecosystem

Successful platforms such as Android and iOS make the convenient availability of applications a strategic tool for adding value to their platforms.

To be able to build an adequate number of applications with acceptable quality, the entire platform is designed around convenience for developing, building, deploying, and updating applications.

Given the relatively small scale of Apertis when compared to the Android and iOS ecosystems, the best strategy is to align to the larger Linux ecosystem, and Flatpak is the widely adopted solution to the previously listed challenges.

However, what makes Flatpak particularly compelling for Apertis is that Flatpak effectively creates a shared development ecosystem that crosses the distribution boundaries: while the fact that being automatically able to run any desktop Flatpak on Apertis is an amazing technological feat, the biggest benefit for Apertis is that by joining the Flatpak ecosystem the skills developers need to learn to develop applications for Apertis become the same as the ones needed to write applications aimed at all the mainstream Linux desktop distributions. This significantly expands the potential developer pool for Apertis, and ensures that the easily available online documentation and workflows to build applications for the main Linux desktop distributions also automatically apply to building applications for Apertis itself.

The next-generation Apertis application framework

The next-generation Apertis application framework is a set of technologies bringing applications to the state-of-the-art of security and privacy considerations.

With the use of modern tools, the framework is meant to grant to the user strict control over its data. Applications are meant to be run contained, and can talk with each other and with the rest of the system only using dedicated interfaces.

The containment is designed to keep the applications on their restricted environment and prevents to modify the base system in any way without being explicitly granted to do so.

Whenever possible, applications have to define upfront their requirements to access privileged resources, be it to share files across application or to get Internet access. It is up to the app store maintainers[2] to review and ensure that the requested access is sensible before it reaches final users. For other more dynamic privileged resources, authorization can be granted at runtime thorugh explicit user interaction, usually via dedicated interfaces called "portals".

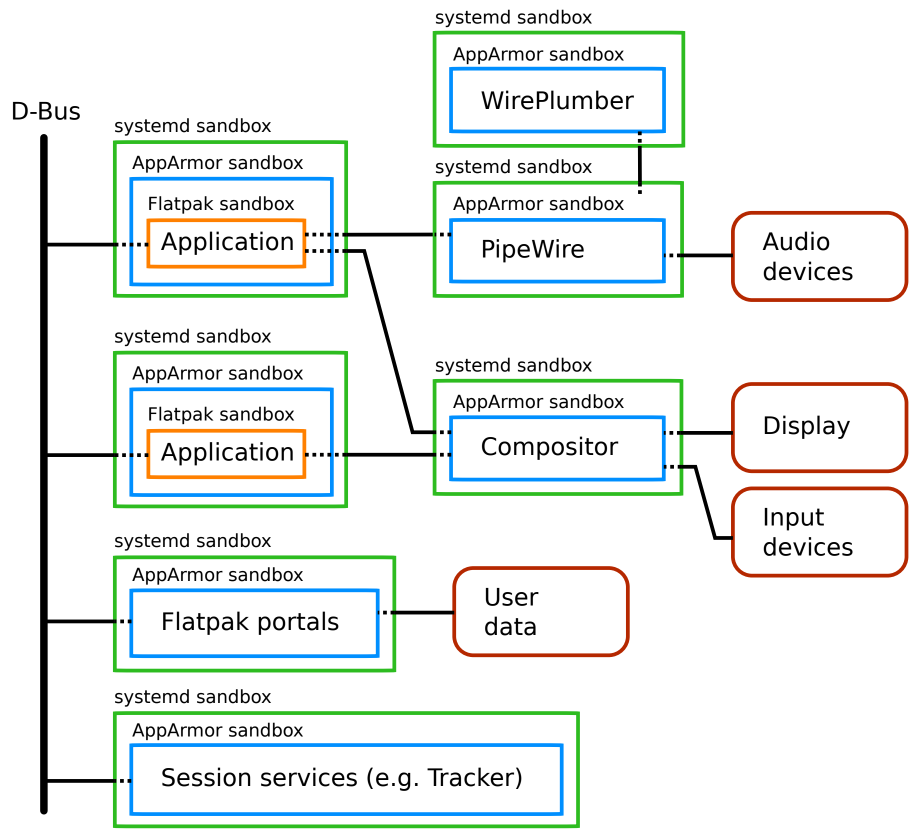
Flatpak provides those guarantees by using the kernel namespacing and control groups subsystems to implement containers similarly as what Docker does. Portals are then implemented as D-Bus interfaces that application can invoke to request privileged actions from inside their sandbox.

Access to the graphical session both to render the application contents and to manage input from users is managed securely by a Wayland compositor.

Audio policies are extremely important for Apertis, specially so in automotive environments, and PipeWire provides an excellent foundation to handle those by providing the tools to wire applications to the needed resources in a secure and customizable way.

Launching applications, agents, and other services happens through `systemd`, which in charge to run both the system and the user sessions. Systemd provides a wide set of options to further secure services[3], track their resource consumption, ensure their availability, etc.

[2]https://martyn.pages.apertis.org/apertis-website/policies/contributions/#the-role-of-maintainers
[3]https://gist.github.com/ageis/f5595e59b1cddb1513d1b425a323db04



Application runtime: Flatpak

Flatpak is a framework with the goal of letting developers to deploy and run their applications on multiple Linux distributions with little effort. To do so, it decouples the application from the base OS: this decoupling also allow an application to be deployed with no changes on different variants of the same base OS, different versions of the same base OS or even be deployed alongside another application which need an incompatible set libraries.

Decoupling the base OS from applications is particularly valuable for Apertis since it allows applications to be deployed seamlessy over multiple variants while minimizing the set of components shipped in the base OS.

Another interesting effect of the decoupling is that the release cycles of applications are no longer tied to the one of the base OS: while the latter needs to go through a longer validation process, applications can release much faster and in a completely independent way.

Applications as made by the developer

A Flatpak application is a self-contained application based on a runtime, ensuring that the user runs the application the way it has been meant by the developer without depending on what is currently installed on the user machine.

Secure by design

A Flatpak application run confined under a restrictive security sandbox. Updates for the application can be done quickly and atomically or according to any system-wide policy. As Flatpak is vendor-agnostic, it allows ensuring that the applications are genuine by signing the applications and the source store.

Flatpak at the moment does not support AppArmor to further confine applications. Since Apertis makes heavy use of AppArmor to protect its service, we plan to add AppArmor support to Flatpak to add another layer of defense to keep applications confined and prevent them from doing unwanted changes to the base operating system.

Privacy

Every application ship with a security profile that describes the requirements of the application and explicit consent from the user is needed to get access to any service not described by the security profile.

Integrated into the environment

Flatpak is providing the latest standards for building applications: using reversed DNS domain name notation, AppStream and Desktop specifications from FreeDesktop.org developers have a complete control over the metadata of their applications and have the suitable tools to provide rich information describing their application.

Efficient and lightweight

Flatpak is very efficient and doesn't require to spend time configuring a heterogeneous set of tools to work on a system. With libostree at the heart of Flatpak, cutting-edge technology is used to reduce its footprint by the use content deduplication. The deduplication results in consuming less disk-space and less network bandwidth.

Release at your own pace

Flatpak decouples applications from the underlying Operating System, so that they can follow different release schedules minimizing the impact of conflicting changes: applications in Flatpak rely on basic set of libraries called "frameworks" that shield them from the actual libraries used by the OS. OSTree helps to keep this redundancy under control, minimizing the storage consumption by de-duplicating items in common. Frameworks help to keep the base OS lean and minimal as non-core libraries can be moved closer to the applications that need it, and thus development and validation can happen faster. On the application side, new versions of basic libraries can be used without fearing regressions on other applications, reducing the time to market.

Compositor: libweston

The compositor is the boundary between applications and the actual human-machine interface: it is responsible of mediating access to the screen and to the input devices, guaranteeing that each application only get the input commands directed to it and can't read or interfere with the resources assigned to other applications.

The next-generation Apertis application framework continues to rely on the Wayland protocol to let applications talk to the compositor in a secure, efficient, and well-supported way.

The compositor is meant to be agnostic of the UI toolkit applications use, and by sticking to the commonly implemented Wayland interfaces it supports the main OSS UI toolkits out of the box, even running at the same time, with no custom code being required on the application side.

While applications targeting the next-generation Apertis application framework should work with any compliant Wayland compositor implementing the most common extensions, Apertis plans to provide a reference compositor that aims to be customizable for the different non-desktop use-cases targeted by Apertis.

The main requirement for the reference compositor is to be based on `libweston`, as this library is a valuable asset of reusable code for compositors originating from the Weston project.

A good starting point for the compositor reference implementation is to use the agl-compositor[4] project because it was purposely built as a reference implementation. Ease of coding was a design goal, and it is expected that both the client shell and the compositor itself are easy to understand and modify. The code base is small, trim, maintained and is currently evolving.

Additional features includes support to clients using XDG shell protocol, and an example of a compositor private extension that allows the client shell to provide additional roles to surfaces.

Another option for the reference compositor is the Maynard[5] project. Unfortunately the project is not currently maintained, and it's internal architecture is outdated: it builds Weston plugins out of tree which was the recommended way before libweston existed. The main issue of using Maynard is that because it is not maintained upstream, we would need to maintain it ourselves.

Audio management: PipeWire and WirePlumber

Applications should be able to play sounds and capture the user speech if they desire to do it, but the system need to guarantee that:

- applications cannot interfere with the audio streams of other applications;

[4]https://gerrit.automotivelinux.org/gerrit/admin/repos/src/agl-compositor
[5]https://gitlab.apertis.org/hmi/maynard

- access to the audio captured by microphones is granted only on explicit authorization by the user whenever possible;
- on a multi-zone setup like on some cars, sounds are emitted in the zone where the application is displayed;
- important messages can be emitted in clear, audible way even if other applications are already playing multimedia contents, by pausing the other streams whenever possible or mixing the streams at different volumes.

PipeWire is the current state-of-the-art solution for secure and efficient audio routing. Applications can use it natively, from GStreamer, or via the ALSA and PulseAudio compatibility layers, and it is designed to work well when combined with the Flatpak sandboxing capabilities.

Since PipeWire does not include any default policy engine, a separate component is in charge of setting up the connections between the PipeWire nodes to ensure that the system rules are enforced. The WirePlumber[6] project from AGL implements such policy service with goals and restrictions aligned to the ones for Apertis.

Session management: systemd

While not directly exposed to applications, session management is a fundamental part of the application framework with the purpose of:

- launching applications upon user request from the graphical launcher;
- running headless agents;
- activating session services needed by applications and agents;
- monitor the life-cycle of applications and services;
- enforce resource tracking on applications and services.

The systemd user session system provides the currently most advanced solution to the above problem space, with the Apertis legacy application framework already making use of it and other mainstream environment like GNOME being in the process of completely switching to systemd to manage their sessions.

Software distribution: hawkBit

For software distribution use-cases Apertis supports Eclipse hawkBit, a domain independent back-end framework for rolling out software updates to constrained edge devices as well as more powerful controllers and gateways connected to IP based networking infrastructure. This software distribution has to be enhanced to gain flatpak support.

With Flatpak, bundle repositories can be created and configured as needed, and a single system can fetch applications from multiple repositories at the same time.

[6]https://gitlab.freedesktop.org/gkiagia/wireplumber

Apertis will offer a reference instance where application can be shared and made available to all the Apertis users, to foster collaboration and to provide a rich set of readily available applications.

Downstreams and product teams can set up their own instance to publish applications intended for a more limited audience.

The Apertis reference store also builds on top of the Apertis GitLab code hosting services to define a reproducible Continuous Integration workflow to automatically build applications from source and publish them to the app store.

Once the quality assurance has validated a specific version of an application, an easy way is provided to the developer to publish the Apertis hawkBit instance.

To ensure a good quality of service, and to be certain that the service matches the expectations, Apertis core applications may themselves be shipped as Flatpak bundles over the Apertis hawkBit instance.

Evaluation

The next-generation application framework matches all the requirements that have driven the development of the legacy application framework.

In particular, in no way the next-generation application framework results in a loss of functionality or features: it instead builds on top of mature, proven technologies to expand what it is possible with the legacy framework, adapting to the evolving state-of-the-art application ecosystem on Linux.

The application framework is compliant with the current requirements of the Apertis platform for system services[7], user services[8], and graphical programs[9]. It relies heavily on the freedesktop.org specifications that specify where applications can store their data with different guarantees, how their metadata is to be encoded, and how they can best integrate with the system.

Flatpak uses `libostree` to implement robust application updates and rollbacks, efficiently using network bandwidth and local storage. Updates are signed and the alternative signing mechanisms developed by Apertis for its system updates can be used to avoid the GPL-3 issues related to the use of GnuPG.

The requirement of having a security boundary between applications is addressed by the use of the control group and namespacing kernel subsystems. The use of AppArmor can be introduced to add another layer of defense to the already strong secuirty provisions Flatpak offers. Flatpak also let applications to be installed per-user, increasing the separation on multi-user systems.

Application data and settings are stored inside the application sandbox, ensuring that they are stored securely, that they can be managed easily for rollback

[7]https://martyn.pages.apertis.org/apertis-website/glossary/#system-service
[8]https://martyn.pages.apertis.org/apertis-website/glossary/#user-service
[9]https://martyn.pages.apertis.org/apertis-website/glossary/#graphical-program

purposes, and that applications are free to chose any mechanisms to manage them.

App bundle contents

The Flatpak application bundle contents[10] is a well-defined application layout that largely matches the approach used by the legacy application framework, improving over it in particular with the introduction of "frameworks" as a way to decouple the application from the base OS and yet retain efficiency in term of deploying updates affecting multiple applications and in term of storage consumption.

With the use of Flatpak frameworks any language runtime can be used easily by applications even if the base OS does not ship it.

Data Management

Flatpak applications can use the XDG Base Directory Specification[11] to find the appropriate places to store persistent private data that can't be accessed by other applications, and temporary cache files that can be deleted by the system to reclaim space.

Policies for storage space reclaiming and rollback need to be defined and are to be implemented in dedicated components.

Sandboxing and security

With the use of the control group and namespacing kernel subsystems, Flatpak offers a state-of-the-art approach for containing applications to limit what they can access on the system and to isolate them from each other.

The integrity of the application data is guaranteed by the namespaced application filesystem being mounted read-only, and thus being unmodifiable by the application itself, and by using namespaces to limit the amount of data each application can access.

Applications can not see the other installed and running applications and neither can modify them. They also can't communicate between each other without user consent.

App pemissions

The Flatpak pemissions[12] system allows to declare in advance any needed permissions to access sensitive resources like user data or special devices, to be reviewed by app store curators.

Additional runtime permissions to access data outside of what the application normally need to use can be granted via explicit user actions, usually via dedicated Flatpak portals.

[10] https://github.com/flatpak/flatpak/wiki/Filesystem
[11] https://specifications.freedesktop.org/basedir-spec/basedir-spec-latest.html
[12] http://docs.flatpak.org/en/latest/sandbox-permissions.html

Integration with Flatpak portals to transparently grant applications privileged access on explicit user actions is already available in the main application toolkits like Qt, GTK, etc.

App launching

Each installed Flatpak application automatically exports its `.desktop` entry point, in a way that any compliant application launcher can automatically list and start the installed Flatpak applications.

The applications themselves have to use the Desktop Entry Specification[13] to provide the required metadata and entry points.

It is possible for applications to explicitely specify that they should not be listed in the launcher, to avoid headless agents polluting the menu.

Document launching

Applications and entry points can specify the media types they handle using the MIME type handling provisions[14] from the Desktop Entry Specification[15]. The application framework is responsible of making the selected document visible to the associate application and run the application if it wasn't previously running, or queue the queue the file opening on busy systems.

URI launching

With the special `x-scheme-handler` MIME type the same mechanism used for *Document launching* can be used to handle specific URI schemes. In case the URI scheme is a `file`, treat it as launching a local document.

Content selection

Flatpak provides portals to let users explicitly grant access to any of their files without any upfront special permissions being granted to the application. Integration with the file selection portals is already available in the most widespread OSS application toolkits.

Data sharing

Flatpak applications can be granted special permissions to access D-Bus services or filesystem subtrees that can be used to share data across a set of applications. Flatpak also let applications to be activated on-demand via D-Bus, which can be particularly useful for headless agents.

Life cycle management

Each Flatpak sandbox automatically contains all the application processes in a secure and efficient way. The system user session management can add an-

[13] https://standards.freedesktop.org/desktop-entry-spec/latest/
[14] https://standards.freedesktop.org/desktop-entry-spec/latest/ar01s10.html
[15] https://standards.freedesktop.org/desktop-entry-spec/latest/

other layer of control, tracking both application and system services with a homogeneous approach.

The compositor can track to which process and thus to which application or service each window belongs to.

Last used context

Applications can store their last status in their private data area and have it available on the next launch, enabling the implementation of the simplest approach purely on the application side with no specific involvement of the application framework.

More advanced use cases that may require a deeper involvement of the application framework needs to be evaluated.

Installation management

Flatpak allows applications to be installed system-wide or per-user, and provides extensive tooling to retrieve contents from remote stores, list local applications, and fetch updates.

The use of OSTree to store application contents makes rolling them back simple and efficient. Data is not usually rolled back when rolling back an application: if use-cases require data rollback it needs to be implemented in dedicated components.

Flatpak also provides both efficient online and offline installation mechanisms.

Conditional access

Flatpak lets applications to be installed either system-wide, making them available to every user, or per-user where only user that have explicitly installed an application can access it.

However, the latter means that storage is not de-duplicated. Advanced setups may be defined to leverage the de-duplication capabilities of OSTree without automatically sharing installed applications with every user of the system.

UI customization

One of the key values for Apertis is to be aligned with upstream, so the best UI customization strategy is to rely on the upstream theming infrastructure offered by toolkits like GTK.

Flatpak can inject system themes in the containerized runtimes[16] to apply a global theme without changing anything in the applications.

[16] https://blog.tingping.se/2017/05/11/flatpak-theming.html

Focus on the development user experience

Flatpak provides extensive tooling to give developers a working environment that is easy to setup and use: the framework provides the necessary tools and libraries for developers to create their application and is highly extensible.

A key part of delivering the best developer experience is by promoting a default Integrated Development Environment (IDE). As such, GNOME Builder is the current best option for providing an efficient environment bringing Flatpak as a first-class component, integrating with many languages, providing support for Git versioning system, and available on any Linux distribution as it is itself distributed as a Flatpak.

As the framework is composed of a set of different tools interacting with each other, it is also possible for the developer to use a classic developer workflow and use the command line to build and install an application. Guaranteeing the same result independently of the machine it is built on and thus allowing fully reproducible builds. The framework itself is built upon existing technology, it will benefit from the broadly available documentation and support of highly heterogeneous build configuration that each application requires.

Installing a flatpak application from Flathub only requires a single command, here is an example with Goodvides, an internet radio player application:

```
flatpak install flathub io.gitlab.Goodvibes
```

The application can then be run by clicking on the desktop icon or simply with:

```
flatpak run io.gitlab.Goodvibes
```

Each application can be defined using a standard manifest[17] that describes all the dependencies, their source and how to build them. If a dependency is not in the Apertis framework Runtime, it can be added by the developer itself in the definition file. The libraries aren't shared with the base system, allowing the developer to ship the version of the dependency that matches the needs of the software and not needing to wait for it to be available in the system itself. A set of tools is even available for the developer to build a runtime using the same dependencies that are available on its machine.

To illustrate the comprehensive coverage of flatpak regarding the developer experience, here are the few steps to build the Goodvides application that we previously mentioned:

[17] http://docs.flatpak.org/en/latest/manifests.html

1. Getting the manifest describing the dependencies from the original package

```
flatpak run --command=cat io.gitlab.Goodvibes /app/manifest.json > io.gitlab.Goodvibes.json
```

2. Build the flatpak locally, allowing to install the dependencies from flathub if required

```
flatpak-builder          --install-deps-from=flathub          build-
dir io.gitlab.Goodvibes.yaml
```

That's it, the flatpak is now built

3. For testing the result, you can directly use

```
flatpak-builder --run build-dir io.gitlab.Goodvibes.yaml goodvibes
```

Legacy Apertis application framework

Both the new and the legacy Apertis application frameworks are meant to be available at the same time in Apertis as alternative options, the legacy framework being shipped on the reference images in the short term. As the new application framework matures, components will get swapped on the reference images, leaving the legacy components available in the archive.

See thecanterbury legacy application framework[18] for more details.

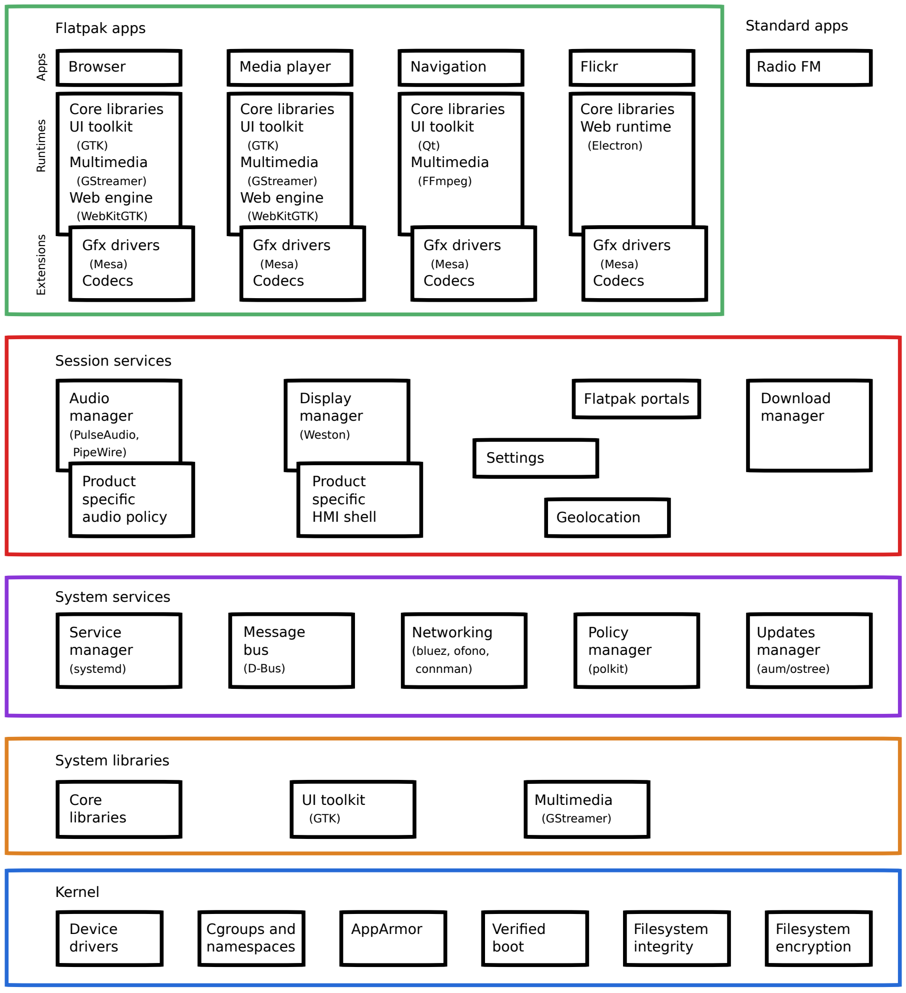
High level implementation plan for the next-generation Apertis application framework

The transition to the new infrastructure can follow a process to keep the legacy framework fully available during the whole process and ensure that it still continue to work afterwards. Both frameworks will be in the Apertis repositories as mutually exclusive options to be chosen by product teams based on their needs.

The new Apertis application framework integrates with the existing QA and testing platform for Apertis.

[18]https://martyn.pages.apertis.org/apertis-website/architecture/canterbury-legacy-application-framework/

The implementation will be held whithin a few different axis that can be developed in parallel and in the order that might make more sense at the time of the implementation.



Flatpak on the Apertis images

The goal here is to ensure that all the Flatpak tools and services are working on the reference Apertis images.

1. Ensure that all the Flatpak tools are installed by default on the reference Apertis images:
    - target images have the tools needed to install, update, run, and remove Flatpak applications
    - SDK images also ship the tools needed to create Flatpak bundles

2. Test that a simple test application like GNOME Calculator can be installed on the reference Apertis images, that it gets displayed normally and that the user interaction is also working.
3. Test a more complex application like Goodvibes, ensure that the audio playback is working.
4. Test more complex applications requiring GL rendering (for instance, OpenArena), ensure that the open-source graphical rendering stack works. Testing the proprietary graphical stack is out of scope as it does not provide same levels of functionality and support when compared to the open source stack.
5. Taking the needs of the product teams into consideration, go through the list of official portals and ensure that they are functional.

The Apertis Flatpak application runtime

The goals here are to create a reference Flatpak runtime for Apertis applications and move all the applications to Flatpak.

To avoid bottlenecks, the Flatpak bundles produced in the steps described here can be tested on any non-Apertis platform supporting Flatpak.

1. Setup Flatdeb[19] to automate the creation of Flatpak runtimes and Flatpak applications from `.deb` packages using the GitLab Continuous Integration pipelines.
2. Create a basic Apertis reference runtime aimed at headless agents and without legacy component like Mildehall, built with Flatdeb[20], similar to the FreeDesktop.org SDK.
3. Create a guide for product teams to create their own applications and runtimes using the Apertis tools.
4. Create a basic Flatpak runtime to run Mildenhall applications
5. Convert the sample-apps to Flatpak using the Mildenhall runtime, starting from the simplest ones to the ones requiring the most interaction with the system. Ensure that each porting process is documented.
6. Coalesce the documentation in a comprehensive guide to convert legacy applications.
7. Convert more complex Mildenhall legacy applications like Frampton.
8. Create a legacy-free Apertis reference runtime for GUI applications.
9. Investigate more modern alternatives to the Mildenhall legacy demo applications and base them on the legacy-free Apertis reference runtimes.

Implement a new reference graphical shell/compositor

This section is about deploying a new graphical shell based on modern components and avoiding deprecated libraries like Clutter.

[19]https://gitlab.collabora.com/smcv/flatdeb
[20]https://gitlab.collabora.com/smcv/flatdeb

1. Begin with a new minimal shell based on the Weston Wayland compositor and make it available on the reference images, to be enabled optionally.
2. Ensure that legacy Mildenhall applications work properly under the new compositor.
3. Progressively add features like notifications and an application drawer to discover and launch applications.
4. Switch the default compositor from the legacy Mildenhall-Compositor to the new one.
5. Iteratively improve the look and feel of the shell.
6. Document how the shell can be customized or replaced by product teams while fully re-using the Weston core compositor implementation.

Switch to PipeWire for audio management

The steps described here are about making audio management more secure and flexible on Apertis.

1. Update the Apertis audio management[21] design document to describe the different approach using PipeWire[22] instead of PulseAudio.
2. Start the work using a basic policy with WirePlumber[23] from AGL.
3. Ensure that audio capture is functional using a simple audio player application.
4. Ensure that video capture is functional using a simple camera viewer application.
5. Ensure that audio playback is functional without PulseAudio, but still default to PulseAudio for audio playback.
6. Ensure compatibility with applications using the PulseAudio client libraries to provide a smooth migration.
7. Switch the default for audio playback to PipeWire.
8. Progressively refine policies and introduce stream priority handling.
9. Provide a guide for product teams about customizing the audio management policies.

AppArmor support

This section focuses on using AppArmor as an additional level of security to constrain applications.

1. Add a basic AppArmor profile setup to Flatpak to ensure each application runs with its dedicated profile.
2. Progressively make the application profile more strict.
3. Customize the AppArmor profile based on the application permissions described in its manifest.

[21] https://martyn.pages.apertis.org/apertis-website/concepts/audio-management/
[22] https://pipewire.org
[23] https://gitlab.freedesktop.org/gkiagia/wireplumber

The app-store

For the user-driven use-case it is key to demonstrate a full workflow that includes an application store.

The store and the deployment management service are kept separate:

- the store is the front-end for the user and is the commercial layer of the system (payments, etc.);
- the deployment management service manages the actual installation of the software on the device based on the state of the store, but also dealing with updates that do not go through the store.

1. Improve the reliability of the Apertis hawkBit instance.
2. Plug the Apertis hawkBit instance authentication system to the Apertis user database.
3. Extend the application building pipelines to push Apertis apps to hawkBit.
4. Extend the hawkBit agent to manage Flatpak applications.
5. Create and deploy a simple front-end store for applications, extending an existing e-commerce platform or adopting hawkBit-based solutions like the Kuksa Appstore[24].
6. Ensure that the whole app-store workflow is documented and functional to handle user-driven installations and updates via hawkBit.
7. Extend the hawkBit agent and other tools to handle the conditional access[25] use cases.
8. Provide a guide for product teams about deploying their own app-store.

[24] https://github.com/eclipse/kuksa.cloud/tree/master/kuksa-appstore
[25] https://martyn.pages.apertis.org/apertis-website/concepts/conditional_access/