Multiuser transactional switching

Contents

Terminology . . . 2
Requirements . . . 3
Assumptions . . . 3
Use-case scenarios . . . 4
Technical considerations . . . 12
Approach . . . 12
Multiple users should be able to use the system, though not concurrently . . . 13
Switching users should not disturb some of the core functionality, such as music playing . . . 14
When the user starts the system they should find the same applications they had left open at shutdown, and in the same state . . . 14
When switching users, open applications must remain open . . . 14
Switching users shall be performed with a smooth transition, with no visual flickering . . . 16
User switching should not take more than 5 seconds . . . 16
User data is private to each user . . . 16
Removable devices are accessible to all users and all users can unmount/eject them . . . 17
Limiting customizability as a trade-off . . . 17
Recommendations summary . . . 18

This document describes one particular set of use-cases for how multiple users are expected to use the Apertis system, using the Multiuser Design document[1] as a base. It starts by describing the use cases that are believed to be important in the automotive context, followed by a technical analysis and recommendations.

The specific set of use cases on which this document focuses is a "transactional" temporary switch between users, which is a relatively unusual situation in mainstream computing, but has been identified as a situation that is more likely to arise in an automotive environment.

In order to balance the various requirements and priorities that might be present in OEM variants of Apertis, it is useful to consider trade-offs such as not allowing user switching in runtime, if implementing an ideal user experience for this feature would be too onerous or only possible with a sub-par experience. The amount of customization allowed would then be reduced to account for this design restriction, as discussed in Limiting customizability as a trade-off

Terminology

Please see the Multiuser Design document1 for the definitions used in this document for jargon terms such as *user*, *user ID/uid*, *trusted*, *system service*, *user*

[1] https://www.apertis.org/concepts/multiuser/

service, *multi-seat* and *fast user switching*.

Requirements

See the Multiuser Design document for general requirements applicable to all aspects of the multi-user design in Apertis. This document focuses on one set of use cases which has been identified as requiring detailed design: a "transactional" switch between users.

The driver is the primary user of the car, and hence the car's infotainment interface; but because the driver must be able to focus on driving, it is desirable that the front-seat passenger can "take over"a shared screen (for instance, in the typical design that places a touchscreen between the driver and front passenger) so that they can carry out a task on the driver's behalf (for instance, programming a navigation destination or finding a required piece of information).

The Apertis user interface is anticipated to be customizable, and the passenger' s preferences do not necessarily match the driver's. As a result, it is desirable that the passenger can temporarily switch to a set of preferences with which they are more familiar.

Depending on the specific use-case, it might be necessary for the passenger to access their own private data (as opposed to the driver's private data).

When switching users, it must be possible for open applications to remain open. Some use-cases benefit from this, and some do not.

Some of the requirements from the Multiuser Design document are particularly relevant to this design, and are re-stated here:

- Switching users shall be performed with a smooth transition, with no visual flickering.
- User switching should not take more than 5 seconds.

Assumptions

Some of the requirements in the Multiuser Design document are stated in terms of a class of possible sets of requirements, among which a concrete design must make a choice. In this document we have assumed the following requirements.

- User data is private to each user:
  - Settings
  - Address book
  - Browser history
  - Application icons
  - Arrangement of icons in the app launcher

  - Account data for web services
  - Playlists
- The following will be shared, if that makes the design simpler:
  - Applications (from the store)
  - Media library (music, videos)
  - Paired Bluetooth devices
- Removable devices are accessible to all users and all users can unmount/eject them

The idea is that application binaries, libraries and other supporting data, as well as media files, will be shared, but each user will have their own view of those. That means for instance that when an application is installed by a user for the first time its icon would appear only on the current user's launcher. When other users install the application no download would be necessary –it would just be a matter of making the icon appear on that user's launcher.

Out of scope for this document Some configurations are outside the scope of this particular proposal. They could be supported by a different concrete design within the general framework described by the Multiuser Design document.

- Multiple concurrent users are out-of-scope: to provide desired performance on optimized hardware, each user's applications will not in general remain active when another user is logged in. Instead, the previous user's processes will be instructed to save their state and exit so that they can be resumed later. An implementation may have both sessions run in parallel for a short time if necessary, in order to facilitate a smooth transfer, but this is intended to be merely a transitional state.
- Multi-seat (as defined by the Multiuser document) is out-of-scope: for the same reasons, if there are multiple screens, they will all be associated with the same user. In this document we do not aim to support separate concurrent logins on different screens (e.g. separate sessions for rear-seat passengers).

Use-case scenarios

This design includes all of the use-cases described in the Multiuser Design document, including those that require user switching and optional privacy between users.

When we approach the implementation stage for this design, it would benefit from input from a UX designer, with two main aims: first, confirm that the use cases and their suggested workflows make sense; and second, for use cases that benefit from "hinting"the user towards particular actions, recommend ways in which this can be done.

Passenger acts on behalf of driver; access to driver's private data Driver Diana and passenger Peter are on the way to visit Peter's friend Fred. Diana asks Peter to check Fred's exact address. Fred has shared the address on Facebook, in a post that is visible to Diana but not to Peter, or to both Diana and Peter.

Trivial case Assuming that the driver's display is situated between the driver and the front passenger as is conventional, Peter can use the shared display that is currently "logged in"as Diana. The system has no way to distinguish between input from Peter and input from Diana.

Comments: This use case is trivial to implement –indeed, it would be difficult to avoid implementing it –and it is equivalent to the behaviour of a single-user system. It is only mentioned here for comparison with the more complex use-cases below, where the system needs to be aware that the person using it has changed.

Switching for a transaction: access to non-driver's private data Driver Diana and passenger Peter are on the way to visit Peter's friend Fred. Diana asks Peter to check Fred's exact address. Fred has shared the address on Facebook, in a post that is visible to Peter but not to Diana.

Diana is the current user of the Apertis system. However, accessing this information requires Peter's private data (in this case, Facebook credentials).

Switching for a transaction Peter selects a menu option labelled "Switch user···"or similar, chooses his own name from a list of users, and authenticates in some way if required. This switches the current user of the Apertis system from Diana to Peter, so that he can view his Facebook page and find Fred's address. For the purposes of this particular use case, the initial state of Peter' s session is not significant (but see subsequent scenarios for situations where it does matter).

Transferring selected data Peter should be able to select Fred's address and set it as the satnav destination, without leaving Diana able to access his Facebook account in future.

Obviousness of current user context If Diana and Peter have selected different user interface themes, it should be obvious on whose behalf the Apertis system is acting: it should use Peter's theme if and only if it is working with Peter's data.

Core functionality not interrupted Certain core functions in the infotainment domain should not be interrupted by the user switch. For instance, if Diana was listening to locally stored media or to the radio, or using satnav to

navigate to the city where Fred lives, this should not be interrupted. In particular, it must be possible for a navigation-related notification (such as an imminent turning or a speed limit change) to appear during the animated transition from Diana to Peter.

Driver's settings retained for core functionality It is important that the driver is not distracted. While Peter is using the Apertis system, certain core functions should remain linked to the driver's user preferences, and should take precedence over what Peter is doing. For instance, if navigation is in the infotainment domain, it should continue to use Diana's preferences to determine how far in advance to warn Diana about a turning.

Alternative model, not recommended An alternative model that could be used for this transactional switching would be to use Peter's user interface preferences (theme, etc.), but with all applications still running as Diana, so that they have access to Diana's private data but not to Peter's. However, this model would not satisfy point **a** of this particular use case, because Diana's browser is either not logged in to Facebook, or logged in as Diana; and it is undesirable to require Peter to enter his Facebook password into Diana's browser. It also does not satisfy point **c**: we feel that using Peter's UI theme for Diana's browser would mislead Peter into believing that this browser is running on his behalf, not Diana's.

Cancelling the transaction Assume that the preconditions and events of use case Switching for a transaction: access to non-driver's private data have occurred. While trying to find Fred's address, Peter is distracted (perhaps by a call on a phone not connected with the car) and does not continue to interact with the HMI.

Driver regains control of Apertis system Because some functions of the Apertis system are driver-focused, it must be easy for Diana to revert to her preferred configuration. If Peter's use of the situation is viewed as a "transaction" , then Diana reclaiming the system can be viewed as "aborting"or "rolling back" the transaction.

This could occur either via a timer (when Peter stops interacting with the HMI for some arbitrary length of time, control returns to Diana) or via explicit action from either Peter or Diana (a menu option or touchscreen gesture).

Diana's "last-used"state is restored The foreground application, the set of background applications, and all of their states should be identical to how they were at the beginning of Switching for a transaction: access to non-driver' s private data. It is as if the "transaction"had never happened.

Comments: Returning to the last-used state is important for a variant of this use-case: if Diana accidentally initiates user-switching, then cancels the action,

this should not result in state being lost.

Automatically switching via a timer could lead to undesired results, and should be deployed with care: for instance, if Peter has left a photo of Fred's house displayed on the screen to help Diana to identify where to park, but has not explicitly used some "send to user···"action to "complete the transaction"by explicitly sending that content back to Diana, then it is undesirable for the system to switch back to Diana's context if that would mean not displaying that photo.

As a result, we recommend that user-switching should be via an explicit action, not via a timer. One possible compromise would be for a timer to trigger a notification that effectively asks "are you still there?", offering actions "switch back to Diana"and "stay as Peter".

Switching user, maintaining state –web Driver Diana starts to look for information on a web page, then asks the passenger Peter to take over so that she can concentrate on driving. Peter wishes to authenticate as himself (as in Switching for a transaction: access to non-driver's private data) so that he can use his own display preferences, bookmarks, etc.

State transfer Peter selects a menu option in Diana's web browser labelled "Send to···"or similar, or uses a touchscreen gesture with the same effect. He chooses his own name from a list of users, and authenticates as himself. After Peter authenticates, the browser remains open in Peter's session, and it displays the same web page that Diana was looking at.

Comments: The user interface design for this requires some care to set up the appropriate privacy expectations: if the action was phrased more like "switch user"rather than "send to", this would risk users unintentionally sharing private state, leading to a loss of confidence in the system.

Transfer back Peter finds the desired information and selects the "Send to ···"option again. The browser remains visible in Diana's session, displaying the same web page that Peter was looking at.

Comments: This use-case has privacy concerns due to the unclear security model that has evolved over time for the Web, and must be handled carefully. To fulfill the use case, the state that is transferred must include the web page's URL and/or its content. In either case this can lead to a poor UX or a security vulnerability if mishandled, even taking into account that Peter can already see the contents of Diana's screen:

- If the state transfer is done by URL, suppose Diana is currently looking at a page for which Peter does not have the necessary credentials, for instance a private Google+ post from someone who is not Peter's friend. In this case, the first thing Peter will see is a "permission denied"message, which is not a friendly user experience.

- If the state transfer is done by URL, suppose Diana is currently looking at a page whose URL is itself sensitive, for instance a Google Docs "shareable URL"that contains its own authentication token. In this case, by retrieving the URL from browser history, Peter now has perpetual access to edit that document, which was not intended by Diana and could be characterized as a security flaw. This could be mitigated by careful user interface design, for instance choosing a verb with implications of "send"or "share".
- If the state transfer is done by content, suppose Diana is currently looking at a page whose hidden content is sensitive, for instance one that contains an authentication token to act on Diana's behalf in an embedded form. In this case, by retrieving the content from browser cache, Peter now has access to that authentication token, which once again was not intended by Diana. Again, this could be mitigated by careful UI design.
- If the state is transferred back to Diana (point **b**), there is an equivalent of each of those issues, with the roles reversed.

As a result of the issues described, Collabora recommends being careful to set privacy expectations via UI design.

Alternative model The alternative model described in Alternative model would avoid any privacy concerns, but does inherit the same issues as in and is not recommended.

Switching user, maintaining state –music In a situation similar to the scenarios above, driver Diana starts to look for a particular song in the media library, then asks passenger Peter to take over so that she can concentrate on driving. Assume that Peter knows the desired song is in one of his playlists.

Peter's playlists are available Peter should be able to use his own playlists to find the song. There are two ways this could work, depending whether playlists are considered to be private or merely user-specific (see the Requirements section of the Multiuser Design document).

If playlists are considered to be private, Peter must authenticate and switch to his own user context, as in scenarios Switching for a transaction: access to non-driver's private data and Switching user, maintaining state –web, to locate his own playlist.

If playlists are not considered to be private, Peter may either switch to his own user context, or locate the playlist while remaining in Diana's configuration as in Passenger acts on behalf of driver access to drivers private data (for instance, the music player could show an unobtrusive "Peter's playlists"folder alongside Diana's own playlists).

Peter's HMI configuration is available To minimize frustration, Peter should be able to use his own configuration/"look & feel"for the media player to find that song, not Diana's unfamiliar configuration.

In practice, whether Peter will actually switch users in order to do this seems likely to depend on which he finds more irritating –using an unfamiliar user interface, or authenticating to switch user? –and on whether he intends to do other things "as himself"after finding the song. Remaining in Diana's configuration is covered by Passenger acts on behalf of driver access to drivers private data, so we assume here that he does switch.

Active app remains active The media player should still be the active app after Peter has switched to his own user context. The other apps that were running last time Peter used the car are not started.

Non-private state is transferred If Peter does switch to his own user context, the state in which Diana was viewing the media library browser (e.g. currently viewed album) is preserved.

Non-private state can be transferred back to Diana Peter finds the appropriate playlist, queues the song for playing and stops using the Apertis system. If he opts to use a similar "send···"option to return control of the Apertis system (as in scenario Switching user, maintaining state –web), the state in which Peter was viewing the media library browser is preserved, i.e. the playlist remains displayed. If he merely switches back ("cancelling"the transaction as in scenario Cancelling the transaction), the media player returns to the state that was saved as part of Diana's session during point **a**.

Alternative model: The alternative model described in Alternative model would naturally satisfy points **b**, **c**, **d** and **e**, but would not satisfy point **a** unless playlists are not considered to be private.

Switching user, maintaining state –unknown app In a situation similar to Switching for a transaction: access to non-driver's private data, driver Diana starts to look for a particular item in an arbitrary third-party app not specifically known to the system (e.g. a restaurant guide), then asks passenger Peter to take over.

Switching user Suppose Peter knows that the desired restaurant is saved in his favourites, or believes that it would be easier to find in his user interface configuration. He should be able to authenticate and use his own configuration to find it.

Active app remains active The restaurant guide should still be the active app after Peter has switched to his own user context. Like Switching user,

maintaining state –music, but unlike the "user switching"scenario described in the Multiuser Design document, the other apps that were running last time Peter used the car are *not* started.

Non-private and transient state is transferred Suppose Diana has got part way through finding the desired restaurant, and has narrowed down search results to the correct city. Peter should not be required to to repeat that process: the first thing he sees after login should be the same search results. If the user interface is designed to set the expectation that state will be transferred, using words such as "send"or "share", then the amount of state that can be transferred without violating that expectation is greater.

Private state is not transferred Because third-party apps could do anything, and the level of privacy of the data they deal with will vary greatly, it should also be possible for the app developer to avoid transferring all of its state between users. For instance, if Diana is logged-in to the restaurant guide app so that she can submit reviews, her login credentials must not be transferred to Peter.

Explicitly returning state transfers it back If Peter "sends back"the state in which he was viewing the restaurant guide, similar to scenario Switching user, maintaining state –web, then that state is seen in Diana's instance of the app.

Cancelling the transaction restores previous state If Peter merely cancels the transaction and lets the system return to Diana, similar to Cancelling the transaction, then the state in which the app was saved before point **a** is restored.

Alternative model: The alternative model described in Alternative model would naturally satisfy all these points except the first half of the first, unless favourite restaurants are not considered to be private.

App declines to transfer state Suppose a current user Alice (who could either be the driver or passenger, there is no distinction in this use case) is using the Apertis system under her own user context. She is using a third-party app whose designer does not consider it to be appropriate to transfer any state to another user under any circumstances, for example a saved-password manager or an online banking app.

Suppose Alice attempts to transfer state to another user Bob, as in Switching for a transaction: access to non-driver's private data, Switching user, maintaining state –web etc.

State transfer does not occur In this particular app, there is no state that would be appropriate to transfer to Bob. The user switch should not occur:

for example, this could be implemented by displaying a notification instead of starting the switching process, or by putting an explanatory message where the list of possible users would normally appear. If the UX design is such that apps normally have a "send to···"menu option or button, it could appear disabled, or not be present at all.

However, if sending to another user is done via a touch gesture, there is no direct equivalent of a disabled option. In particular, touch gestures should always have visual feedback, whether successful or not (similar to the way scrolling is often made to "bounce"at the end of the scrollable range). This is so that the user can distinguish between an unrecognized gesture, and a recognized gesture that did not result in an action in this specific case.

Comment: this does not arise when cancelling a transaction as in Cancelling the transaction, because that action does not transfer state in any case.

Switching user, maintaining state –missing app Similar to Switching user, maintaining state –unknown app, driver Diana starts a restaurant guide app, then asks Peter to take over. This time, suppose that Peter has not installed the restaurant guide, so the system will not be able to reproduce the current state for Peter.

Impossible state transfer is not offered Similar to the previous scenario, the system should not offer the ability to send state to Peter.

One possible implementation would be to avoid displaying a "send to user···" control in applications that are not installed for any other users, and to avoid listing Peter in the menu of possible users if he does not have the application. This has the disadvantage that in a system with three or more users, it could become non-obvious why some applications display that control and some do not, and why some users do not always appear in the menus.

Alternative design another design that was considered is to run the app anyway, on the basis that it is in fact already installed on the system. However, this undermines the abstraction that each user has their own collection of apps. It also does not address the issue that the app might require accepting a EULA, approving a request for special OS permissions (access to GPS, etc.) or similar actions, which Peter has not done.

Alternative design a third design that was considered is to present a choice between "just switch user to Peter"(which would restore his last-used state) and "don't switch". However, presenting the driver with a distracting prompt/question is undesired.

Alternative design a fourth possibility is to switch to Peter, with the initial state in Peter's session automatically opening the app installation procedure. If

Peter chooses to install the relevant app, the state transferred from Diana should be inserted into the "newly installed"app. If Peter does not install the relevant app (for example because he does not agree to an EULA or OS permissions request), the transaction should be cancelled (as in Cancelling the transaction).

Comments: as with the previous scenario, this scenario cannot occur when cancelling a transaction as in Cancelling the transaction, because that action does not transfer state in any case.

Technical considerations

The use case scenarios described above impact on several design decisions which may lead to technical challenges.

The most important of all is the implication that most of the state, including applications, remains the same after a user switch.

One possible approach is that the application remains running through a user switch and simply loads the private data of the new user.

As noted in the more general Multiuser design document, implementing such a feature would require doing away with the separation of privileges provided by using one UNIX uid (user account ID) and one X or Wayland session per user, pushing all of the burden of authorization and tracking states for each user onto the applications. The complexity for application authors could be alleviated by providing some common high level APIs, but even in that case it would all be new, untested code. It would also put too much trust on the applications themselves, which would each be treated as a security boundary in this model; it is highly likely that some applications would mishandle user checks, allowing data to leak from one user session to the other.

For these reasons, Collabora does not recommend this approach; instead, as in the more general Multiuser design document, we recommend that each user is represented by a separate UNIX uid, with all state transfer between users mediated by system services.

Furthermore, we believe it is important to consider a number of trade-offs regarding the desired functionality and the technical viability of the solution. The recommendations below try to strike a balance between ease of use, complexity for the application developer, stability and security.

Approach

This chapter goes over each of the requirements presenting the trade-offs Collabora feels are necessary and proposing technical solutions to approximate as much as possible the desired user experience.

Multiple users should be able to use the system, though not concurrently

The general approach Collabora recommends is adopting the usual approach with one UNIX uid per user, similar to the approach used for desktop and laptop systems.

In most GNU/Linux distributions a user switch is performed by running a second instance of the X server and starting a second session with the appropriate uid, after which subsequent switching between the same users is simply a switch between those two X servers (the so-called “fast user switching”model, described in more detail in the Multiuser design document).

A similar approach could be adopted for Apertis, but it would most certainly lead to memory pressure very quickly. For this reason Collabora believes the best way to implement user switching is by closing down the whole session of the current user, saving applications’ states while doing so, and only then starting the session for the other user. This is equivalent to the procedure used in most GNU/Linux distributions for a “log out”operation followed by a new login, but with the addition of a “save state”step before closing each application; it is also very similar to switching between user accounts on Android devices.

Potential for concurrent users as a future enhancement One option that can be considered is to provide additional hardware resources in systems shipping for premium segment cars, such as doubling the available RAM, for instance. This would help with memory pressure and make the approach involving two X servers an achievable goal, with the caveats discussed below.

CPU usage, for instance, could become a problem and degrade the performance experienced by the second user if the programs running on the first user’s session are kept running. One possible solution for this is a freeze/thaw approach in which the first user’s applications remain present in memory, but their execution is paused until the first user’s session is resumed.

If the first user’s session is not frozen completely, then services running outside the user sessions, such as the media player (see Switching users should not disturb some of the core functionality such as music playing below), would need to deal with the fact that there are now potentially two controlling user interfaces and handle multiple connections gracefully.

Other system resources would also probably need to be regulated, such as muting applications of the first user so that a game running on their session would not interfere with a different game running on the second user’s session. Bandwidth regulation may become more complex, as well, to ensure no application from the first user interferes with streaming being performed inside the second user’ s session, and there are implications that need to be considered if the first user’ s phone is being used for Internet connection and has a metered data plan that B will now be using.

Switching users should not disturb some of the core functionality, such as music playing

This kind of cross-session functionality points to two probable design decisions. The first is that at least some of the data used by the various users should be shared; for example, music should probably be stored in a shared repository rather than in a user's private storage area.

The second is that the application that performs the actual playing must persist. There are two major options here, from which a concrete recommendation has not yet been chosen; depending on other requirements, the various "core" components do not necessarily all need to take the same solution.

- It could be a *system service* (as defined by the Multiuser Design document) rather than a regular user-level application. That means it will be executed outside the user session, and be controlled by the user interface via D-Bus or a similar inter-process communication mechanism; this naturally results in its state, such as the list of tracks currently queued, being shared between all users. The relevant user interfaces in each user's session could all communicate with the same system service.
- It could be a *user service* running on behalf of the driver, which is flagged not to be terminated during user-switching, and communicates with the users via notifications.

Other "core"services that need to span across multiple user sessions, such as navigation (if present in the Apertis domain), could follow a similar design. For example, the oFono service used for telephony is already a system service, so taking the "system service"option for that is a natural approach.

If the system service needs to distinguish between users and act on behalf of a specific user in response to their requests, it is important to note that this results in it being part of the TCB (trusted computing base) responsible for enforcing separation between users. Such services should be checked to ensure that they do not violate the system's intended security model.

When the user starts the system they should find the same applications they had left open at shutdown, and in the same state

This topic is discussed in the Multiuser and Applications design documents. The only aspect directly relevant to this particular document is that the same "save state"step that would be done during shutdown should be performed when switching away from a user, so that the saved state can be reloaded for use cases such as scenario Cancelling the transaction.

When switching users, open applications must remain open

This requirement exists to enable use cases in which the driver asks a passenger who is also a user of the system to perform some task (Switching user,

maintaining state –web, Switching user, maintaining state –music, Switching user, maintaining state –unknown app and Switching user, maintaining state –missing app are examples of this category). This passenger would log in, but at least part of the state of the driver's session would remain.

We see three possible ways to satisfy these use cases:

- Transfer the state of all apps from the driver's session to the new user's session
- Transfer the state of the single foreground app from the driver to the new user
- Do not transfer any state

In some use cases such as Cancelling the transaction and App declines to transfer state, having the same applications open after a switch is not a desirable user experience. It is not necessarily true that the user would like to use, for instance, the browser if the previous user had it open. It's also not clear that the currently open browser tabs are necessarily interesting to the new user, particularly if they will "overwrite"the new user's saved browser tabs from their last session.

Finally, the privacy implications of implicitly transferring state are considerable, with significant potential for "over-sharing"; this could cause users to lose confidence in the system and avoid using it for personal data, reducing its usefulness.

Our recommendation is that each application should have a way to indicate to the operating system whether it is able and willing to send (partial or full) state to another user's instance of the same application. If it is, the HMI can display a "send to other user"option for that application; if the application is such that state transfer is unsafe or never useful, or if it simply does not support state transfer, then that option would appear disabled (greyed-out) or not appear at all.

The actual state transfer would be similar to the state saving mechanism that is already needed for save/restore functionality (as discussed in the Preferences and Persistence design document[2], and more briefly in the Multiuser[3] and Applications[4] design documents), but placing state in memory or in an OS-supplied temporary directory instead of in the per-(user, app) data directory. We recommend that similar data formats and API conventions should be used, so that in trivial cases where there is no private state, the application's implementations of "save state"and "send state"can call into the same common code. However, it should be presented as a separate, parallel API call, to encourage application authors to think about the amount of state transfer between users that is desired. Similarly, the "restore state"and "receive state"operations should be distinct, but

[2] https://www.apertis.org/concepts/preferences-and-persistence/
[3] https://www.apertis.org/concepts/multiuser/
[4] https://www.apertis.org/concepts/applications/

follow similar enough conventions that they can share an implementation if that is what the application author wants.

Optionally transferring the state of a single foreground app, with vendors encouraged to design their HMIs to set appropriate privacy expectations for this action, seems a reasonable compromise between the convenience of transferring state when it is desired, and the disruption and privacy concerns of transferring state when it is not desired.

We do not recommend the alternative model in which the superficial appearance of the passenger's preferences is applied to processes that continue to run with access to the driver's personal data (as outlined in Alternative model), since that approach seems likely to lead to users'privacy expectations not matching the reality. This applies to both the driver's privacy (it is not entirely obvious that the passenger can still access the driver's private data) and the passenger's privacy (for instance, it is not at all obvious that the passenger should not enter passwords into what appears to be "their"browser).

Switching users shall be performed with a smooth transition, with no visual flickering

This topic is discussed in the Multiuser Design document. We recommend the approach involving the first user's session handing off to a separate system-level compositor, which in turn hands off to the second user's session.

User switching should not take more than 5 seconds

This requirement puts pressure into how long the user session may take for closing down. An application that spends a lot of time writing state or doing some other processing, like an email client synchronizing its state with a slow IMAP server, may increase the amount of time required for completing the switch significantly. This means care must be taken in application development to not allow this.

Other than that, Collabora believes the system components for user switching should be pretty fast and that the 5 seconds goal is achievable.

Note that in a premium car system, depending on the additional amount of memory available, the applications would not necessarily really be closed down, so this requirement could more easily be achieved by simply freezing the existing session or not touching it at all.

User data is private to each user

By using the traditional "one UNIX uid per user"approach, each user will have its own home directory protected by the usual mechanisms, such as file ownership, user and group permissions, in addition to the AppArmor restrictions described

in the Security Design document[5]. Note that usage of the UNIX home directory concept, in which a single directory has all of a given user's files, is not in the plans for Apertis. Instead, each application will store its data in a directory named after the UNIX user account, and owned by the appropriate uid, but inside the application directory.

More information about this can be found in the Applications design.

However, some data will be shared The requirements state that optionally some data can be shared if it makes the problem more tractable. Collabora believe it's a good idea to make installed applications and data such as the music library be shared. Making installed applications per-user makes application management much more complex, including possibly having to waste space by having two separate versions of the same application available.

A custom view can still be provided for each user. The icons for applications may appear only if the user explicitly installs the application, which in this case would not cause a new download, just the addition of the icon to the user's launcher. The same can go for other kinds of user interface aids such as playlists, providing the user with a way of picking the songs or videos they are interested in from the shared library.

More information about this can be found in the Applications design.

Removable devices are accessible to all users and all users can unmount/eject them

This requirement can be fully satisfied by the proposed approach. The mounting and unmounting of devices is a privileged operation that is mediated by system services already, so making it so that any user can mount and unmount devices no matter who mounted them in the first place is simply a matter of setting that up as the policy.

Limiting customizability as a trade-off

If the Apertis ends up being designed with no user switching or even no multi-user capabilities, then it might be desirable to consider limiting the customizability of the system, so as to not burden drivers who seldom use the system that is customized by the main driver.

As a general principle, the easier it is made to switch between users, the more customizability can be offered without it becoming a problem. One special case is that the mechanism to switch users should remain obvious and in a consistent location in all configurations and themes. Similarly, the user interface for driver-focused tasks, such as the icon to open satnav functionality, should remain consistent between configurations.

[5] https://www.apertis.org/concepts/security/

If user-switching is absent or limited, Collabora believes that any customization that allows relocation of items and interface controls should be avoided. That means any configuration for the positions or visibility of menu items, application launchers, core user interface elements such as the status bar, the back button, and so on should not be allowed.

Appearance customization, such as colour scheme, should not cause trouble for a casual user of the system trying to find their way. The same goes for features that allow organization of user data such as the creation of custom playlists or photo albums. However, configuration of fonts and font sizes can cause the core UI elements to change layout in ways that might be confusing, so allowing configuration for those needs to be considered carefully.

Recommendations summary

As discussed in Multiple users should be able to use the system though not concurrently, Collabora recommends having one UNIX user account ID (uid) per user. The first user to be registered in a new system must be able to perform administration tasks such as system updates, application installation, creation of new users and setting up permissions, as discussed in the main Multiuser Design document.

At a conceptual level, user switching should be done by closing down the user session and starting the new user session, to avoid memory pressure. However, implementors should consider allowing the old session to run in parallel for a short time while applications are given a chance to save and exit. Running two user sessions in parallel for an extended period of time, to enable "fast user switching", can be considered for premium cars with greater computing resources available.

Services that need to stay running after a user switch should have their background functionality split from their UIs, as discussed in section Switching users should not disturb some of the core functionality such as music playing; they can either run as a different UNIX user account ID –a "system service"–or be a specially flagged "user service"that is not terminated with the rest of the session.

Collabora recommends against trying to have a login mode that moves the entire session state from the current user to the user that is logging in, as described in When switching users open applications must remain open. To satisfy use cases in which the current state of one user's application is sent to another user' s instance of the same application, it would be sufficient to have that single application save and restore state, using a mode which omits private data from the state where necessary. It is not necessarily possible or desirable to implement this for every application, and care must be taken to set appropriate privacy expectations.

Ways of having a smooth visual transition when switching users are discussed in the main Multiuser Design document. Collabora recommends the use of

multiple Wayland compositors, with the first user's *session compositor* handing over control of the graphics device to a *system compositor* to perform the switch, which in turn hands over the graphics device to the second user's session compositor.

Collabora recommends in [this section][However, some data will be shared] that data for applications and media files be shared among users to avoid duplication, with custom views allowing per-user customization.