



Multiuser transactional switching

Contents

2	Terminology	3
3	Requirements	3
4	Assumptions	3
5	Use-case scenarios	4
6	Technical considerations	12
7	Approach	13
8	Multiple users should be able to use the system, though not con-	
9	currently	13
10	Switching users should not disturb some of the core functionality,	
11	such as music playing	14
12	When the user starts the system they should find the same appli-	
13	cations they had left open at shutdown, and in the same	
14	state	14
15	When switching users, open applications must remain open . . .	15
16	Switching users shall be performed with a smooth transition, with	
17	no visual flickering	16
18	User switching should not take more than 5 seconds	16
19	User data is private to each user	17
20	Removable devices are accessible to all users and all users can	
21	unmount/eject them	17
22	Limiting customizability as a trade-off	18
23	Recommendations summary	18

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

28 The specific set of use cases on which this document focuses is a “transactional”
29 temporary switch between users, which is a relatively unusual situation in main-
30 stream computing, but has been identified as a situation that is more likely to
31 arise in an automotive environment.

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

¹https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application_security/multiuser/

38 Terminology

39 Please see the Multiuser Design document¹ for the definitions used in this doc-
40 ument for jargon terms such as *user*, *user ID/uid*, *trusted*, *system service*, *user*
41 *service*, *multi-seat* and *fast user switching*.

42 Requirements

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

47 The driver is the primary user of the car, and hence the car’s infotainment inter-
48 face; but because the driver must be able to focus on driving, it is desirable that
49 the front-seat passenger can “take over” a shared screen (for instance, in the typ-
50 ical design that places a touchscreen between the driver and front passenger) so
51 that they can carry out a task on the driver’s behalf (for instance, programming
52 a navigation destination or finding a required piece of information).

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

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

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

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

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

66 Assumptions

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

- 70 • User data is private to each user:
 - 71 – Settings
 - 72 – Address book
 - 73 – Browser history

- 74 – Application icons
- 75 – Arrangement of icons in the app launcher
- 76 – Account data for web services
- 77 – Playlists
- 78 • The following will be shared, if that makes the design simpler:
 - 79 – Applications (from the store)
 - 80 – Media library (music, videos)
 - 81 – Paired Bluetooth devices
- 82 • Removable devices are accessible to all users and all users can un-

83 mount/eject them

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

90 **Out of scope for this document** Some configurations are outside the scope
 91 of this particular proposal. They could be supported by a different concrete de-
 92 sign within the general framework described by the Multiuser Design document.

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

105 Use-case scenarios

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

109 When we approach the implementation stage for this design, it would benefit
 110 from input from a UX designer, with two main aims: first, confirm that the use

111 cases and their suggested workflows make sense; and second, for use cases that
112 benefit from “hinting” the user towards particular actions, recommend ways in
113 which this can be done.

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

119 **Trivial case** Assuming that the driver’s display is situated between the driver
120 and the front passenger as is conventional, Peter can use the shared display that
121 is currently “logged in” as Diana. The system has no way to distinguish between
122 input from Peter and input from Diana.

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

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

132 Diana is the current user of the Apertis system. However, accessing this infor-
133 mation requires Peter’s private data (in this case, Facebook credentials).

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

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

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

148 **Core functionality not interrupted** Certain core functions in the infotain-
149 ment domain should not be interrupted by the user switch. For instance, if
150 Diana was listening to locally stored media or to the radio, or using satnav to
151 navigate to the city where Fred lives, this should not be interrupted. In particu-
152 lar, it must be possible for a navigation-related notification (such as an imminent
153 turning or a speed limit change) to appear during the animated transition from
154 Diana to Peter.

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

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

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

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

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

184 **Diana’s “last-used” state is restored** The foreground application, the set
185 of background applications, and all of their states should be identical to how

186 they were at the beginning of **Switching for a transaction: access to non-driver's**
187 **private data**. It is as if the “transaction” had never happened.

188 *Comments:* Returning to the last-used state is important for a variant of this
189 use-case: if Diana accidentally initiates user-switching, then cancels the action,
190 this should not result in state being lost.

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

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

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

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

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

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

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

- 224 • If the state transfer is done by URL, suppose Diana is currently looking
225 at a page for which Peter does not have the necessary credentials, for
226 instance a private Google+ post from someone who is not Peter's friend.
227 In this case, the first thing Peter will see is a "permission denied" message,
228 which is not a friendly user experience.
- 229 • If the state transfer is done by URL, suppose Diana is currently looking at
230 a page whose URL is itself sensitive, for instance a Google Docs "shareable
231 URL" that contains its own authentication token. In this case, by retrieving
232 the URL from browser history, Peter now has perpetual access to edit that
233 document, which was not intended by Diana and could be characterized
234 as a security flaw. This could be mitigated by careful user interface design,
235 for instance choosing a verb with implications of "send" or "share".
- 236 • If the state transfer is done by content, suppose Diana is currently looking
237 at a page whose hidden content is sensitive, for instance one that contains
238 an authentication token to act on Diana's behalf in an embedded form.
239 In this case, by retrieving the content from browser cache, Peter now has
240 access to that authentication token, which once again was not intended
241 by Diana. Again, this could be mitigated by careful UI design.
- 242 • If the state is transferred back to Diana (point **b**), there is an equivalent
243 of each of those issues, with the roles reversed.

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

246 **Alternative model** The alternative model described in [Alternative model](#)
247 would avoid any privacy concerns, but does inherit the same issues as in and is
248 not recommended.

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

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

257 If playlists are considered to be private, Peter must authenticate and switch
258 to his own user context, as in scenarios [Switching for a transaction: access to](#)
259 [non-driver's private data](#) and [Switching user, maintaining state -web](#), to locate
260 his own playlist.

261 If playlists are not considered to be private, Peter may either switch to his own
262 user context, or locate the playlist while remaining in Diana's configuration as

263 in **Passenger acts on behalf of driver access to drivers private data** (for instance,
264 the music player could show an unobtrusive “Peter’s playlists” folder alongside
265 Diana’s own playlists).

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

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

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

278 **Non-private state is transferred** If Peter does switch to his own user con-
279 text, the state in which Diana was viewing the media library browser (e.g. cur-
280 rently viewed album) is preserved.

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

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

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

297 **Switching user** Suppose Peter knows that the desired restaurant is saved in
298 his favourites, or believes that it would be easier to find in his user interface

299 configuration. He should be able to authenticate and use his own configuration
300 to find it.

301 **Active app remains active** The restaurant guide should still be the active
302 app after Peter has switched to his own user context. Like **Switching user,**
303 **maintaining state -music**, but unlike the “user switching” scenario described in
304 the Multiuser Design document, the other apps that were running last time
305 Peter used the car are *not* started.

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

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

319 **Explicitly returning state transfers it back** If Peter “sends back” the state
320 in which he was viewing the restaurant guide, similar to scenario **Switching user,**
321 **maintaining state -web**, then that state is seen in Diana’s instance of the app.

322 **Cancelling the transaction restores previous state** If Peter merely can-
323 cels the transaction and lets the system return to Diana, similar to **Cancelling**
324 **the transaction**, then the state in which the app was saved before point **a** is
325 restored.

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

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

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

338 **State transfer does not occur** In this particular app, there is no state that
339 would be appropriate to transfer to Bob. The user switch should not occur:
340 for example, this could be implemented by displaying a notification instead of
341 starting the switching process, or by putting an explanatory message where the
342 list of possible users would normally appear. If the UX design is such that apps
343 normally have a “send to...” menu option or button, it could appear disabled, or
344 not be present at all.

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

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

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

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

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

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

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

376 **Alternative design** a fourth possibility is to switch to Peter, with the initial
377 state in Peter’s session automatically opening the app installation procedure. If
378 Peter chooses to install the relevant app, the state transferred from Diana should
379 be inserted into the “newly installed”app. If Peter does not install the relevant
380 app (for example because he does not agree to an EULA or OS permissions
381 request), the transaction should be cancelled (as in [Cancelling the transaction](#)).

382 *Comments:* as with the previous scenario, this scenario cannot occur when
383 cancelling a transaction as in [Cancelling the transaction](#), because that action
384 does not transfer state in any case.

385 **Technical considerations**

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

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

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

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

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

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

410 Approach

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

414 Multiple users should be able to use the system, though not concu- 415 rently

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

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

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

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

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

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

447 Other system resources would also probably need to be regulated, such as muting
448 applications of the first user so that a game running on their session would not
449 interfere with a different game running on the second user’s session. Bandwidth

450 regulation may become more complex, as well, to ensure no application from
451 the first user interferes with streaming being performed inside the second user'
452 s session, and there are implications that need to be considered if the first user'
453 s phone is being used for Internet connection and has a metered data plan that
454 B will now be using.

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

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

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

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

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

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

484 **When the user starts the system they should find the same applica-**
485 **tions they had left open at shutdown, and in the same state**

486 This topic is discussed in the Multiuser and Applications design documents.
487 The only aspect directly relevant to this particular document is that the same
488 "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 **Appli-**

²https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application_customization/preferences-and-persistence/

³https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application_security/multiuser/

523 cations⁴ design documents), but placing state in memory or in an OS-supplied
524 temporary directory instead of in the per-(user, app) data directory. We recom-
525 mend that similar data formats and API conventions should be used, so that in
526 trivial cases where there is no private state, the application’s implementations
527 of “save state” and “send state” can call into the same common code. However, it
528 should be presented as a separate, parallel API call, to encourage application au-
529 thors to think about the amount of state transfer between users that is desired.
530 Similarly, the “restore state” and “receive state” operations should be distinct, but
531 follow similar enough conventions that they can share an implementation if that
532 is what the application author wants.

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

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

546 **Switching users shall be performed with a smooth transition, with no** 547 **visual flickering**

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

551 **User switching should not take more than 5 seconds**

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

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

⁴<https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application/applications/>

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

564 **User data is private to each user**

565 By using the traditional “one UNIX uid per user” approach, each user will have its
566 own home directory protected by the usual mechanisms, such as file ownership,
567 user and group permissions, in addition to the AppArmor restrictions described
568 in the [Security Design document](#)⁵. Note that usage of the UNIX home directory
569 concept, in which a single directory has all of a given user’s files, is not in the
570 plans for Apertis. Instead, each application will store its data in a directory
571 named after the UNIX user account, and owned by the appropriate uid, but
572 inside the application directory.

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

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

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

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

587 **Removable devices are accessible to all users and all users can un-** 588 **mount/eject them**

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

⁵https://apertis-website-0b3586.pages.apertis.org/concepts/archive/application_security/security/

594 Limiting customizability as a trade-off

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

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

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

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

616 Recommendations summary

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

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

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

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

644 Ways of having a smooth visual transition when switching users are discussed
645 in the main Multiuser Design document. Collabora recommends the use of
646 multiple Wayland compositors, with the first user's *session compositor* hand-
647 ing over control of the graphics device to a *system compositor* to perform the
648 switch, which in turn hands over the graphics device to the second user's session
649 compositor.

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