



Multiuser transactional switching

1 Contents

2	Terminology	2
3	Requirements	2
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	15
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	17
18	User switching should not take more than 5 seconds	17
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	18
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](#)

38 Terminology

39 Please see the [Multiuser Design document](#)¹ 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*.

¹<https://jwd.pages.apertis.org/apertis-website/concepts/multiuser/>

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
45 of use cases which has been identified as requiring detailed design: a “transac-
46 tional” 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 that
55 the passenger can temporarily switch to a set of preferences with which they
56 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**

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

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

106 **Use-case scenarios**

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

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

115 **Passenger acts on behalf of driver; access to driver’s private data**

116 Driver Diana and passenger Peter are on the way to visit Peter’s friend Fred.
117 Diana asks Peter to check Fred’s exact address. Fred has shared the address on
118 Facebook, in a post that is visible to Diana but not to Peter, or to both Diana
119 and Peter.

120 **Trivial case**

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

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

130 **Switching for a transaction: access to non-driver’s private data**

131 Driver Diana and passenger Peter are on the way to visit Peter’s friend Fred.
132 Diana asks Peter to check Fred’s exact address. Fred has shared the address on
133 Facebook, in a post that is visible to Peter but not to Diana.

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

136 **Switching for a transaction**

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

143 **Transferring selected data**

144 Peter should be able to select Fred’s address and set it as the satnav destination,
145 without leaving Diana able to access his Facebook account in future.

146 **Obviousness of current user context**

147 If Diana and Peter have selected different user interface themes, it should be
148 obvious on whose behalf the Apertis system is acting: it should use Peter’s
149 theme if and only if it is working with Peter’s data.

150 **Core functionality not interrupted**

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

157 **Driver’s settings retained for core functionality**

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

163 **Alternative model, not recommended**

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

173 **Cancelling the transaction**

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

178 **Driver regains control of Apertis system**

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

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

186 **Diana’s “last-used” state is restored**

187 The foreground application, the set of background applications, and all of their
188 states should be identical to how they were at the beginning of **Switching for**
189 **a transaction: access to non-driver’s private data**. It is as if the “transaction”
190 had never happened.

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

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

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

205 **Switching user, maintaining state – web**

206 Driver Diana starts to look for information on a web page, then asks the passen-
207 ger Peter to take over so that she can concentrate on driving. Peter wishes to
208 authenticate as himself (as in **Switching for a transaction: access to non-driver’s**
209 **private data**) so that he can use his own display preferences, bookmarks, etc.

210 **State transfer**

211 Peter selects a menu option in Diana’s web browser labelled “Send to...” or simi-
212 lar, or uses a touchscreen gesture with the same effect. He chooses his own name
213 from a list of users, and authenticates as himself. After Peter authenticates, the
214 browser remains open in Peter’s session, and it displays the same web page that
215 Diana was looking at.

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

220 **Transfer back**

221 Peter finds the desired information and selects the “Send to...” option again.
222 The browser remains visible in Diana’s session, displaying the same web page
223 that Peter was looking at.

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

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

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

253 **Alternative model**

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

256 **Switching user, maintaining state – music**

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

261 **Peter’s playlists are available**

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

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

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

275 **Peter's HMI configuration is available**

276 To minimize frustration, Peter should be able to use his own configuration/"look
277 & feel" for the media player to find that song, not Diana's unfamiliar configu-
278 ration.

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

285 **Active app remains active**

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

289 **Non-private state is transferred**

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

292 **Non-private state can be transferred back to Diana**

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

300 **Alternative model:**

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

304 **Switching user, maintaining state – unknown app**

305 In a situation similar to **Switching for a transaction: access to non-driver’s**
306 **private data**, driver Diana starts to look for a particular item in an arbitrary
307 third-party app not specifically known to the system (e.g. a restaurant guide),
308 then asks passenger Peter to take over.

309 **Switching user**

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

313 **Active app remains active**

314 The restaurant guide should still be the active app after Peter has switched to
315 his own user context. Like **Switching user, maintaining state – music**, but unlike
316 the “user switching” scenario described in the Multiuser Design document, the
317 other apps that were running last time Peter used the car are *not* started.

318 **Non-private and transient state is transferred**

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

325 **Private state is not transferred**

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

331 **Explicitly returning state transfers it back**

332 If Peter “sends back” the state in which he was viewing the restaurant guide,
333 similar to scenario **Switching user, maintaining state – web**, then that state is
334 seen in Diana’s instance of the app.

335 **Cancelling the transaction restores previous state**

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

339 **Alternative model:**

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

343 **App declines to transfer state**

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

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

352 **State transfer does not occur**

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

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

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

367 **Switching user, maintaining state – missing app**

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

372 **Impossible state transfer is not offered**

373 Similar to the previous scenario, the system should not offer the ability to send
374 state to Peter.

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

381 **Alternative design**

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

388 **Alternative design**

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

392 **Alternative design**

393 a fourth possibility is to switch to Peter, with the initial state in Peter’s ses-
394 sion automatically opening the app installation procedure. If Peter chooses to
395 install the relevant app, the state transferred from Diana should be inserted
396 into the “newly installed” app. If Peter does not install the relevant app (for
397 example because he does not agree to an EULA or OS permissions request), the
398 transaction should be cancelled (as in [Cancelling the transaction](#)).

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

402 **Technical considerations**

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

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

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

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

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

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

427 **Approach**

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

431 **Multiple users should be able to use the system, though not concu- 432 rently**

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

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

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

449 **Potential for concurrent users as a future enhancement**

450 One option that can be considered is to provide additional hardware resources
451 in systems shipping for premium segment cars, such as doubling the available
452 RAM, for instance. This would help with memory pressure and make the ap-
453 proach involving two X servers an achievable goal, with the caveats discussed
454 below.

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

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

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

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

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

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

- 483 • It could be a *system service* (as defined by the Multiuser Design document)
484 rather than a regular user-level application. That means it will be executed
485 outside the user session, and be controlled by the user interface via D-
486 Bus or a similar inter-process communication mechanism; this naturally
487 results in its state, such as the list of tracks currently queued, being shared
488 between all users. The relevant user interfaces in each user's session could
489 all communicate with the same system service.

- 490 • It could be a *user service* running on behalf of the driver, which is flagged
491 not to be terminated during user-switching, and communicates with the
492 users via notifications.

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

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

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

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

509 **When switching users, open applications must remain open**

510 This requirement exists to enable use cases in which the driver asks a passen-
511 ger who is also a user of the system to perform some task ([Switching user,](#)
512 [maintaining state – web](#), [Switching user, maintaining state – music](#), [Switching](#)
513 [user, maintaining state – unknown app](#) and [Switching user, maintaining state](#)
514 [– missing app](#) are examples of this category). This passenger would log in, but
515 at least part of the state of the driver’s session would remain.

516 We see three possible ways to satisfy these use cases:

- 517 • Transfer the state of all apps from the driver’s session to the new user’s
518 session
- 519 • Transfer the state of the single foreground app from the driver to the new
520 user
- 521 • Do not transfer any state

522 In some use cases such as [Cancelling the transaction](#) and [App declines to transfer](#)
523 [state](#), having the same applications open after a switch is not a desirable user
524 experience. It is not necessarily true that the user would like to use, for instance,
525 the browser if the previous user had it open. It’s also not clear that the currently
526 open browser tabs are necessarily interesting to the new user, particularly if they
527 will “overwrite” the new user’s saved browser tabs from their last session.

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

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

538 The actual state transfer would be similar to the state saving mechanism that
539 is already needed for save/restore functionality (as discussed in the Preferences
540 and [Persistence design document](#)², and more briefly in the [Multiuser](#)³ and [Appli-
541 cations](#)⁴ design documents), but placing state in memory or in an OS-supplied
542 temporary directory instead of in the per-(user, app) data directory. We recom-
543 mend that similar data formats and API conventions should be used, so that in
544 trivial cases where there is no private state, the application’s implementations of
545 “save state” and “send state” can call into the same common code. However, it
546 should be presented as a separate, parallel API call, to encourage application au-
547 thors to think about the amount of state transfer between users that is desired.
548 Similarly, the “restore state” and “receive state” operations should be distinct,
549 but follow similar enough conventions that they can share an implementation
550 if that is what the application author wants.

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

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

²<https://jwd.pages.apertis.org/apertis-website/concepts/preferences-and-persistence/>

³<https://jwd.pages.apertis.org/apertis-website/concepts/multiuser/>

⁴<https://jwd.pages.apertis.org/apertis-website/concepts/applications/>

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

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

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

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

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

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

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

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

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

592 **However, some data will be shared**

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

599 A custom view can still be provided for each user. The icons for applications
600 may appear only if the user explicitly installs the application, which in this

⁵<https://jwd.pages.apertis.org/apertis-website/concepts/security/>

601 case would not cause a new download, just the addition of the icon to the user's
602 launcher. The same can go for other kinds of user interface aids such as playlists,
603 providing the user with a way of picking the songs or videos they are interested
604 in from the shared library.

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

606 **Removable devices are accessible to all users and all users can un-** 607 **mount/eject them**

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

613 **Limiting customizability as a trade-off**

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

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

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

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

635 **Recommendations summary**

636 As discussed in [Multiple users should be able to use the system though not](#)
637 [concurrently](#), Collabora recommends having one UNIX user account ID (uid)
638 per user. The first user to be registered in a new system must be able to perform

639 administration tasks such as system updates, application installation, creation
640 of new users and setting up permissions, as discussed in the main Multiuser
641 Design document.

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

649 Services that need to stay running after a user switch should have their back-
650 ground functionality split from their UIs, as discussed in section [Switching users](#)
651 [should not disturb some of the core functionality such as music playing](#); they
652 can either run as a different UNIX user account ID – a “system service” – or
653 be a specially flagged “user service” that is not terminated with the rest of the
654 session.

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

664 Ways of having a smooth visual transition when switching users are discussed
665 in the main Multiuser Design document. Collabora recommends the use of
666 multiple Wayland compositors, with the first user’s *session compositor* hand-
667 ling over control of the graphics device to a *system compositor* to perform the
668 switch, which in turn hands over the graphics device to the second user’s session
669 compositor.

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