Content hand-over Use-cases

Contents

Use-cases for content hand-over

This page collects the use-cases for "one-shot" content handover, as currently mediated by the Didcot service.

Some related use-cases which have been associated with the Didcot content-handover service during discussion of its API, but which we believe are sufficiently distinct that they should be examined separately, are collected on Content hand-over/Related[1].

Where use-cases describe a requirement for a particular UI/UX behaviour, this is intended to be shorthand for a requirement that the platform provides enough control and enough information that the behaviour described is implementable.

Terms

A *content-type* (sometimes also called a MIME type or Internet media type) is as defined in RFC:2046[2], for example `text/plain`. It is made up of a *top-level media type* such as `text`, and a *subtype* such as `plain`.

[1] https://jwd.pages.apertis.org/apertis-website/concepts/content_hand-over_related/
[2] http://www.rfc-editor.org/rfc/rfc2046.txt

A *URI scheme* is the part of a URI before the :, as defined in RFC:3986[3]. For example, `http`, `tel`, `mailto` and `skype` are URI schemes.

An *xyz: URI* is a URI whose scheme is `xyz`.

In use-cases, *the platform* refers to the components in the platform layer that provide the required functionality. We anticipate that the implementation for most or all of the functionality required here will be Didcot, a *user service* (as defined in the Multi-User concept document) which is trusted as part of the security boundary between app-bundles, but not trusted as part of the security boundary between users. However, the use cases have been written without this assumption, and in particular, parts of of the implementation that do not need to cross a security boundary should be implemented as library code.

Format of use cases

Prerequisites and assumptions

- Requirement
- Requirement
  - *Implementation notes*

Opening a URI by scheme

Geolocation handover

The user activates a geographic location in an application, for example a "show on map" option in a list of hotels.

- The hotel application asks the platform to resolve a URI representing the location, for example a `geo:` URI with latitude and longitude.
- The platform may launch an appropriate mapping or navigation application and direct it to the appropriate point.
- The platform may instead offer a menu of applications or actions applicable to a location. If so, it must be possible for third-party application bundles to add their own actions to this menu.
  - *Open question: do we need to support more than one action per application?*

No immediate side-effect from plain URIs

The user activates a `mailto:` URI in a web page.

- The platform may start an email application in a "compose a message" state, with the `To` address, subject, body etc. already filled in according to the URI.
- The email application must not send the resulting email until the user chooses to do so. The user must have the opportunity to cancel.

[3] http://www.rfc-editor.org/rfc/rfc3986.txt

- The platform may instead choose to offer a menu of reasonable things that can be done with an email address, such as composing a message or providing the email address to a system address book, either as a starting point for a new contact or to be attached to an existing contact.

The user activates a `tel:` URI in a web page.

- The platform should not start an audio call to the the number without giving the user the opportunity to proceed or cancel.
- The platform may instead choose to offer a menu of reasonable things that can be done with a telephone number, such as making an audio call, composing a SMS or providing the email address to a system address book, either as a starting point for a new contact or to be attached to an existing contact.
- The platform may offer actions from more than one application, such as making a conventional or Skype call.

The user activates a `geo:` URI in a web page. Suppose that, in this particular Apertis system, `geo:` URIs are associated with the navigation application.

- The navigation application may offer a menu of things to do with that location, such as zooming the map to that point or setting it as a navigation destination or waypoint.
- The navigation application may take one of those actions immediately; but if it does, it should offer a way to undo, to avoid malicious applications being able to interfere with navigation, for instance by overwriting the currently-set destination. ()

Specifying an action

The user activates a telephone number in an address book application. Suppose the application's user interface presents phone numbers in the same way as Android, where activating the number itself is expected to initiate a call, and a keyboard icon alongside the number is expected to initiate a SMS conversation.

- On activating the number itself, the address book application communicates to the platform that an audio call is the desired action.
  - *One possible implementation would be to avoid Didcot altogether for this use-case, and have the address book communicate directly with Telepathy Mission Control or with the in-call UI to initiate the call. We recommend this implementation.*
  - *Another possible implementation would be to pass a (URI, action) tuple to the platform, such as* `("tel:+443457484950", "call")`.
- Similar to No immediate side-effect from plain URIs, if an unprivileged application attempted to do the same thing as the address book, it should not be allowed to initiate the call without user confirmation, to avoid untrusted applications imposing a financial cost on the user.
  - *Using Telepathy would naturally provide this: only applications whose*

AppArmor profiles allowed communication with Telepathy Mission Control would be able to initiate calls.

  - *Using a (URI, action) tuple would require that this access-control was implemented separately in the API provider, for example Didcot.*

- On activating the SMS icon, the address book application communicates to the platform that composing a SMS is the desired action.
  - *As for calls, one possible implementation would be to avoid Didcot altogether for this use-case, and have the address book communicate directly with Telepathy Mission Control or with the SMS composition UI to initiate the SMS conversation.*
  - *Another possible implementation would be to use a different (URI, action) tuple here, such as* `("tel:+443457484950", "sms")`.
  - *A third possible implementation would be to interpret* `tel:` *URIs as unambiguously meaning "initiate a call", and introduce parallel* `sms:` *URIs, as has been done in Apple's iOS. However, this is contrary to the principle that URIs are noun phrases, not verb phrases*[4]*, and should probably be avoided or deprecated.*

Are there any other use-cases for specifying a non-default action when handling a URI (link), other than telephony and instant messaging?

Requiring a variant-specific scheme handler

The user activates a `geo:` URI (RFC:5870[5]) representing a geographic location. The designer of this particular Apertis variant has chosen to disallow any application other than the built-in navigation app from handling `geo:` URIs. ()

- The platform should launch the navigation app and carry out some default action, perhaps setting the navigation destination to the specified point, zooming the map to the specified point, or presenting a menu of applicable actions.
- The platform must not launch a third-party app, even if that app is also able to handle `geo:` URIs (for instance a Wikipedia app might handle `geo:` URIs by looking up nearby points of interest).

Disallowing certain scheme handlers

All URI scheme handlers in a manifest file should be examined for relevance to the app during the app-store approval process. Inappropriate URI-scheme handling should be disallowed.

Suppose a third-party application attempts to register itself as the handler for `file:` URIs, thus hijacking all file operations.

- This should not be possible. Either the platform or the app-store approval process should prevent it.

[4] http://www.w3.org/DesignIssues/Architecture.html#Specific
[5] http://www.rfc-editor.org/rfc/rfc5870.txt

Suppose a third-party application attempts to register itself as the handler for `http:` and/or `https:` URIs.

- This should be allowed by the platform, but should be a "red flag" for app-store curators: only general-purpose web browsers such as Firefox and Chrome should be allowed to do this.

Interesting URI schemes

- IANA registry[6]
- `file:` is a special case and is described below (Opening a file by type). It should never have a registered handler.
- `http:` and `https:` could be handled by general-purpose web browsers, although we anticipate that OEMs would often want to restrict this.
- Particular namespaces within `http:` and `https:` could be handled by more specific applications (Handling_a_subset_of_a_scheme).
- Telephony and messaging: `tel:`, `skype:`, `sip:`, etc.
- Geolocation and navigation: `geo:`, `maps:`

Opening a file by type

We anticipate that in practice this mechanism will be used for all `file:` URIs, and for no other schemes.

Suppose a USB drive contains some files, which are displayed as icons or list entries in some built-in file manager or search application.

Activating a built-in content-type handler

Suppose the Apertis system has a built-in video player application. The user activates a MPEG video, for instance by tapping on its icon.

- The file manager should communicate to the platform that this particular file has been activated.
- The platform should discover that its content-type is `video/mpeg`, discover that the video player app can play videos in that format, launch the video player and instruct it to play the video.

Activating a third-party content-type handler

Suppose the Apertis system does not have anything built-in that can open Word documents, but the user has installed a third-party LibreOffice app-bundle which can. The user activates a Word document.

- The file manager should communicate to the platform that this particular file has been activated.

[6] https://www.iana.org/assignments/uri-schemes/uri-schemes.xhtml

- The platform should discover that its content-type is `application/msword`, launch LibreOffice and instruct it to open the document.

Choosing one of several content-type handlers

Suppose the Apertis system has a built-in image viewer supporting (among others) JPEG images, and additionally, a third-party image viewer also supports JPEG images. The user activates a JPEG image.

- The file manager should communicate to the platform that this particular file has been activated.
- The platform should either:
  - launch the built-in image viewer, or
  - launch the third-party image viewer, or
  - present the user with a menu from which they can choose one or the other.
- The platform must provide enough configurability that if the user is prompted with a menu, there can be a "remember this in future" option.
- It should be possible for the platform to provide a configuration option, for instance in a Settings app, with which the user can select a preferred handler for JPEG images.

Requiring a certain content-type handler

Suppose instead that this particular Apertis system is an OEM variant, which has a built-in image viewer supporting JPEG and PNG images, and that the OEM does not ever want third-party image viewers to be used for these formats (). The user activates a JPEG image.

- The file manager should communicate to the platform that this particular file has been activated.
- The platform should identify the image's format, discover that the built-in image viewer has been configured as preferred, and launch it.
- It must not launch a third-party viewer or offer a menu.

Ensuring that the relevant file is accessible

The user activates an attachment in an email application; suppose it is a PDF. We assume that the email application must write the attachment into a file, if it is not already represented as a separate file, and pass its `file:` URI to the platform, so that a suitable application (e.g. a PDF viewer) can be launched.

- The file must be writeable by the initiating app-bundle (in this case email).
- The file must be made readable by the receiving app-bundle (in this case the PDF viewer).
- The original file must not be writeable by the receiving app-bundle, but if a new copy is made, that copy may be writeable by the receiving app-bundle.

- The file is potentially confidential, so it must not be made readable to unrelated app-bundles (so writing it to our equivalent of Android's /sdcard is not a valid solution).
- *Possible implementations:*
  - *One possible implementation would be for a privileged component to copy the file content into a directory accessible only by the receiving app-bundle, for instance using a btrfs "reflink" to avoid duplicating the content.*
  - *Another possible implementation would be to pass an open file descriptor across D-Bus from the initiator, through the platform component that chooses the recipient, to the recipient.*
  - *Writing into a directory that is common to all applications is probably not a viable implementation in Apertis, because we use AppArmor profiles to provide the security boundary between app-bundles, and Unix filesystem permissions can discriminate between uids but not between AppArmor profiles.*

Interesting media types

- IANA registry[7]
- `text/vcard`: contact information
- `audio/*`, `video/*`: media
- We can define a media type for Apertis packages
- `application/vnd.apple.mpegurl`: playlist or reference to a media stream
- *Does TPEG have one or more registered media types? If not, can we register an application/vnd.xyz type for it with IANA?*

Registering new media types

The platform discovers files' content types via a combination of file extensions and "magic numbers". The current implementation of this is the the freedesktop.org Shared MIME Info specification[8].

- (Possible requirement) Apps should be able to register new content types with extension and/or "magic number" matches in their app-bundle manifest files. For example, a reader for a new document format "mydoc" could associate `*.mydoc` files with the `application/vnd.example.mydoc` content-type.
  - *Is this required?*
  - *This could be implemented by including freedesktop.org MIME registrations in the package's content, or by taking input in a simplified format as part of the manifest, and generating corresponding files in `/var/lib/apertis_extensions/mime`, either prior to upload to the app store or during installation.*

[7] https://www.iana.org/assignments/media-types/media-types.xhtml
[8] http://standards.freedesktop.org/shared-mime-info-spec/shared-mime-info-spec-latest.html

  - *Declining to support this use-case would mean that files on disk are never automatically detected as having a content type not described by the platform, but custom media type handlers could potentially still be used for files announced as having that media type by a Web server.*
- If apps can do this, to avoid type handler hijacking, the app review process should include inspecting the desired content types (if any). The app store curator should normally reject anything that would override a standard content-type defined by the platform, and should resolve conflicts between store apps.
  - *This process could be made easier via tools.*

Non-default actions on files

Do we need actions other than the implied "open" for files, or does that only make sense for URIs?

Possible future additions

Handling a subset of a scheme

A preinstalled or third-party YouTube app-bundle has been installed. The user activates a web link to a YouTube video, such as https://www.youtube.com/watch?v=dQw4w9WgXcQ.

- It should be possible for a YouTube application to "take over" this subset of the `https:` URI namespace and play the desired video within that application.
- It should be possible for the user to prevent this from happening, and view the web page in an ordinary web browser.
- This should be a "red flag" for app-store curators: an app-bundle should not be able to take over URIs within the www.youtube.com[9] origin unless it was published or approved by YouTube.

Is this functionality required?

Is there any scheme other than http(s) where this would make sense?

[9] http://www.youtube.com