Architecture of the Delco Client

Delco client is a one part of the complete Delco system. This section covers only the client side. With the client end users can start Delco services which can be basic download and channel service. Services are supplied to the client inside a Delco file. Subscribing to a channel service, basic download services belonging to the channel are started automatically as they arrive. Basic download can consists of one or multiple plain file(s), movie(s), music and so on. Streaming type of service is also supported which means that delivery of the data is optimized in a way that content can be consumed while still downloading it

Delco client combines different protocols when downloading data. It is designed to combine different kind of delivery mechanisms (peer-to-peer and multicast) where they can work simultaneously for the same content. Multicast reception is prioritized in order to preserve network resources better. Peer-to-peer can be efficiently used for a file repair and places where multicast is not available. Peer-to-peer comes also very handy for clients who don't have multicast reception available. Combined multicast and peer-to-peer clients can seed to the peer-to-peer network easily and serve non-multicast clients. Simultaneous operation provides better service to all clients regardless of their reception limitations.

1 Delco File

A Delco file is a description of the services to the client application. The Delco file can describe multiple *services*, but the default is to describe just one. A service can include multiple *contentItems/channels* each having a unique *ID*. A *contentItem* has also unambiguous fileURI so that files can be stored with the intended directory structure. The *ID* is required when reporting progress of the download. A *contentPointer* is used to map a delivery mechanism to the content. A single content can have multiple *contentPointers*, meaning that the content is for example available through multicast and P2P delivery. A *contentType* declares the type of the content behind the URI. It is primarily used with the channel service as the type may not be the same as the file extension. A *receptionReport* defines the report URI. *FluteFileRepair* defines the Associated Delivery Procedure Description (apdURI) for FLUTE file repair.

The DelCo file has the following content for different service types:

Download Service:

service(s)
Attributes:

- ID: optional (required with reporting)
- name: name to describe service
- stream: true/false

  contentItem(s)
  Attributes:

  - ID: required to be unique in the system

- fileURI: to identify the file, to describe the directory structure of the file, to find the right file from the FLUTE session
- continuation: true/false
- bitrate (optional, when streaming help clients to know bitrate in kbps)

contentPointer(s)
Attributes:
- type: flute | torrent | ed2k | magnet | http
- value: URI
- contentType: rss | delco | … : optional

fluteFileRepair (optional)
Attributes:
- apdURI

receptionReport (optional)
Attributes:
- serviceURI

Channel Service:

service(s)
Attributes:
- ID: optional (required with reporting)
- name: name to describe service
- stream: not used with the channel service

channel(s)
Attributes:
- ID: recommended to be unique in the system
- name: name to describe service

contentPointer(s)
Attributes:
- type: flute | torrent | ed2k | magnet | http
- value: URI
- contentType: rss | delco | … : optional

2 User Environment

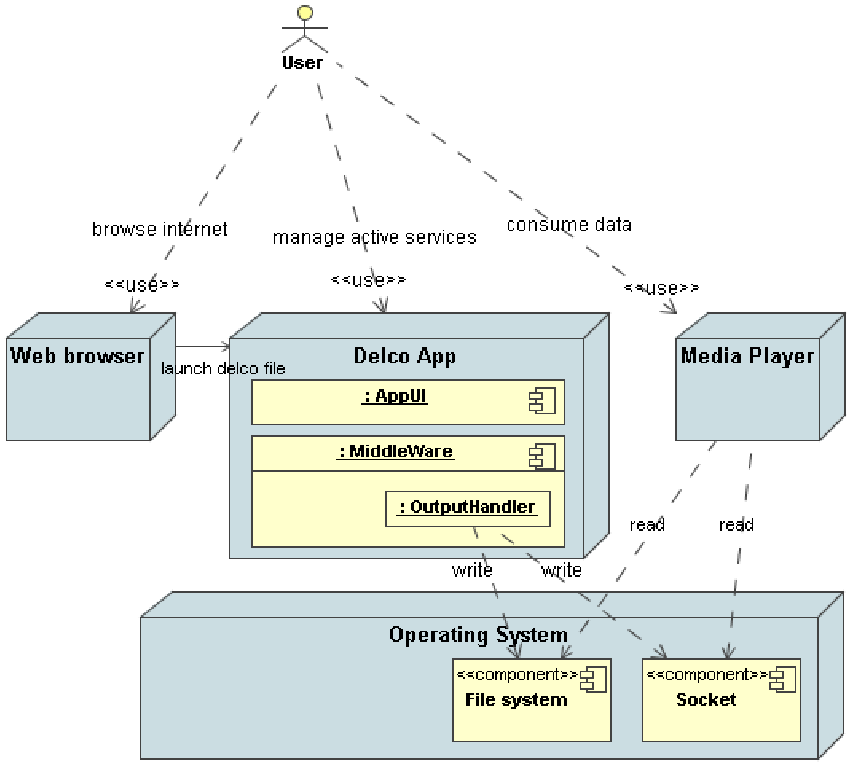

Figure 1: User environment

Here in Figure 1 we can see the typical case of the end user environment. There are web browser, Delco App (Delco client software) and media player on the top of operating system. Typical use case is that first user browses for a web site providing some Delco files. User (or Browser) starts an interesting file with the client. Services included in the file are launched according to the service type and other metadata. At this point user could choose which services to download presented in the file but with our client all services are downloaded. Choice can be however made at both web and delco file level. For an integrated device that don't have good browser capabilities it would be good that client software provides an UI to choose the service from the list of services that resides in the Delco file provided by some central HTTP server. With the help of UI user can observe and manage active services. After some content is available it can be used e.g. with media player. Media player is not integrated into the Delco application but that is possible. In our case Delco application and the media player communicates through an operating system.

While optimizing download procedure for streaming type of service data can be delivered to the media player through a file system or a socket (with UDP). File system is preferred because media players can often handle better data flow through it as currently we don't support RTP. Using a socket means that data is pushed to the socket as fast as consecutive data blocks arrive which can cause too fast bitrates (slow as well, but it is problem also with file system streaming) to the media player. Server releasing services are responsible that content is really streamable when streaming option is used with the service.

3 Software Architecture

This section describes the software architecture of the client. In Figure 2 we can see the layered architecture and how it settles on the top of operating system and system libraries. Application consists of user interface, Delco Middleware, GenericDownload, download libraries and other third party libraries (Boost, wxWidgets). UI is meant to be easily replaceable to support more challenging requirements from usage point of view (for example choosing service from the list of services provided by the Delco file). That also means that wxWidgets (API for creating GUI applications) can be changed easily.

Core components are Delco Middleware and GenericDownload. They are meant to remain intact when customizing application (UI and download libraries) and platform. They depend on C++ Boost library for threading and file system operations among others in order to be independent of the operating system and its system libraries. However if desired platform doesn't support Boost, core components must be modified accordingly.

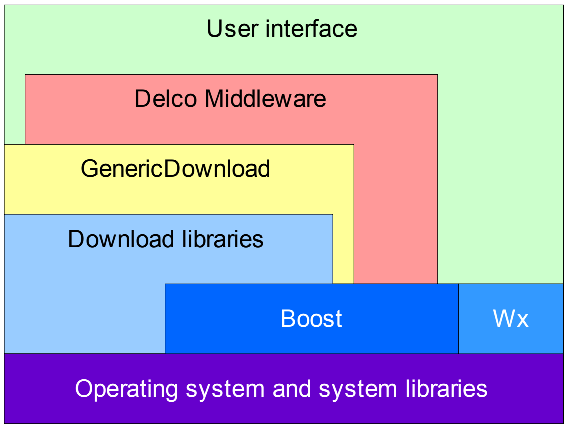

Figure 2: Layered architecture of Delco application

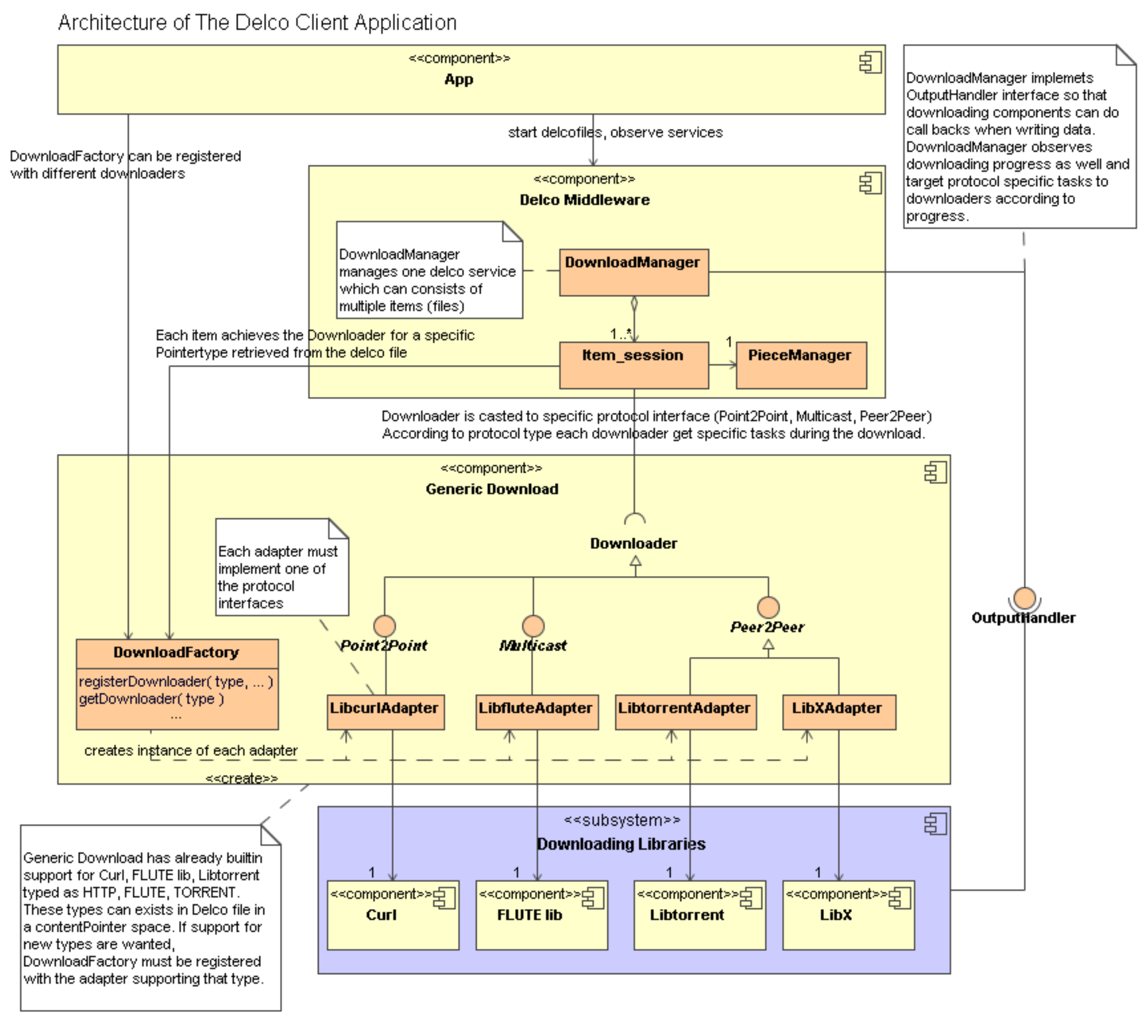
Architecture of The Delco Client Application
<<component>>
App
start delcofiles, observe services
DownloadFactory can be registered with different downloaders
DownloadManager implemets OutputHandler interface so that downloading components can do call backs when writing data. DownloadManager observes downloading progress as well and target protocol specific tasks to downloaders according to progress.
<<component>>
Delco Middleware
DownloadManager manages one delco service which can consists of multiple items (files)
DownloadManager
1..*
1
Item_session
PieceManager
Each item achieves the Downloader for a specific Pointertype retrieved from the delco file
Downloader is casted to specific protocol interface (Point2Point, Multicast, Peer2Peer) According to protocol type each downloader get specific tasks during the download.
<<component>>
Generic Download
Each adapter must implement one of the protocol interfaces
Downloader
Point2Point
Multicast
Peer2Peer
OutputHandler
DownloadFactory
registerDownloader(type, ...)
getDownloader(type)
...
LibcurlAdapter
LibfluteAdapter
LibtorrentAdapter
LibXAdapter
creates instance of each adapter
<<create>>
Generic Download has already builtin support for Curl, FLUTE lib, Libtorrent typed as HTTP, FLUTE, TORRENT. These types can exists in Delco file in a contentPointer space. If support for new types are wanted, DownloadFactory must be registered with the adapter supporting that type.
<<subsystem>>
Downloading Libraries
1
1
1
1
<<component>>
Curl
<<component>>
FLUTE lib
<<component>>
Libtorrent
<<component>>
LibX

Figure 3: Detailed architecture of core components

In Figure 3 we can see more detailed architecture of the core components. Middleware is the engine which controls all services and their downloading process. Each service will be managed by its own DownloadManager and each file (or content) in the service is tracked by the Item_session. PieceManager blocks the content into smaller pieces. While receiving data PieceManager makes best of it to utilize and write the data to disk (or socket) even when received chunk is not aligned with the blocking.

GenericDownload is a component which includes generic downloading and control interfaces for multicast, peer-to-peer and point-to-point types of data reception. It also includes adapters for mapping methods in the interfaces to specialized library methods. Adapters can be registered to DownloadFactory and bring into use whenever service requires. Each adapter is registered with a certain type after which the type can be used in a contentPointer section of the Delco file. Unsupported types in the Delco file are ignored. GenericDownload has already built in support for libtorrent, mad-fcl FLUTE and cURL typed as “torrent”, “flute” and “http”. However application can register other as well if adapter exists and library meets the placed conditions (e.g usage of the OutputHandler callback interface). Middleware has access to download components only through GenericDownload which provides great customization within the download libraries.