PerVoice Service Architecture Tutorial

PerVoice

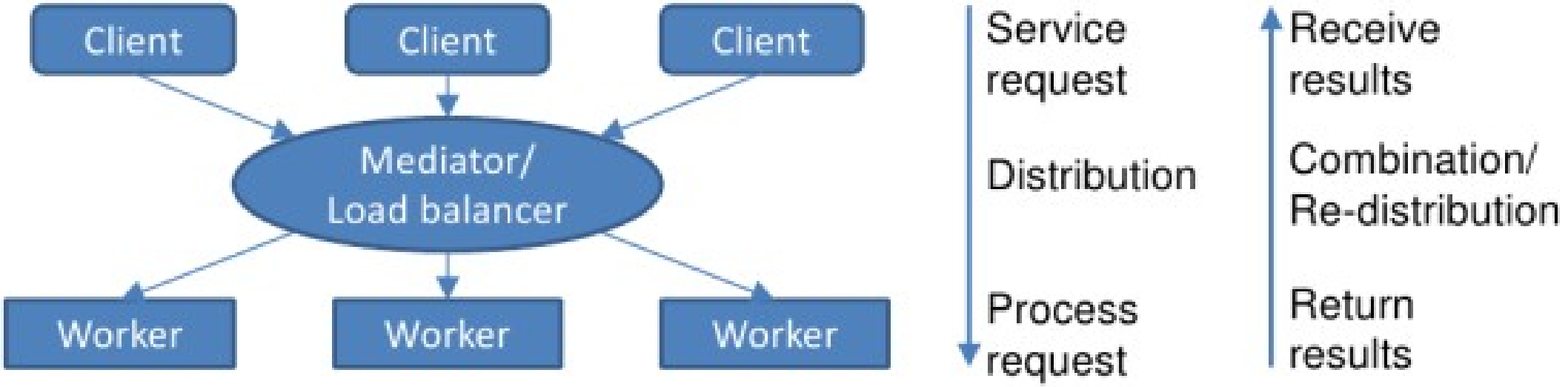
Tutorial overview

Today we will discuss about:

- PerVoice SA introduction and key features
- Terminology
- Flows, streams, queues and callbacks
- Client and Worker implementation example

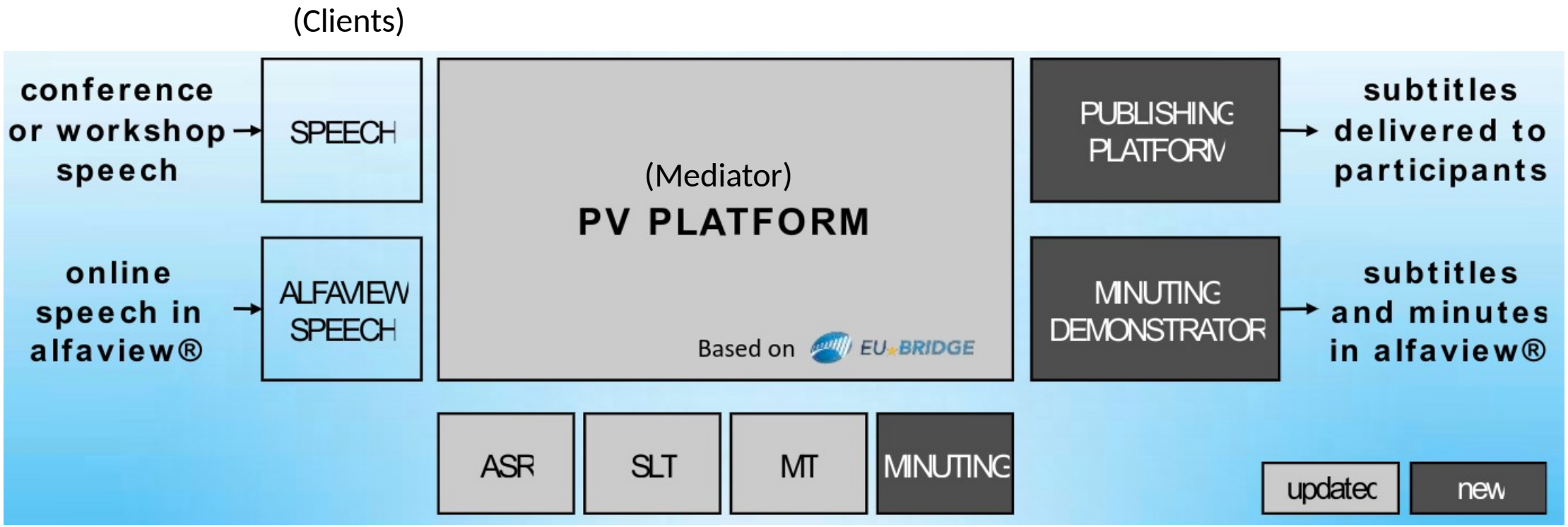
PerVoice

PerVoice SA overview



PerVoice

Placement of the PerVoice SA within the ELITR project



PerVoice SA key features:

Features:

- Distributed architecture
- Connection based communication using TCP/IP sockets
- Message protocol hidden by the API
- XML based message format
- Convenience functions for data handling
- Asynchronous sending and processing of packets using queues and
- Callback functions

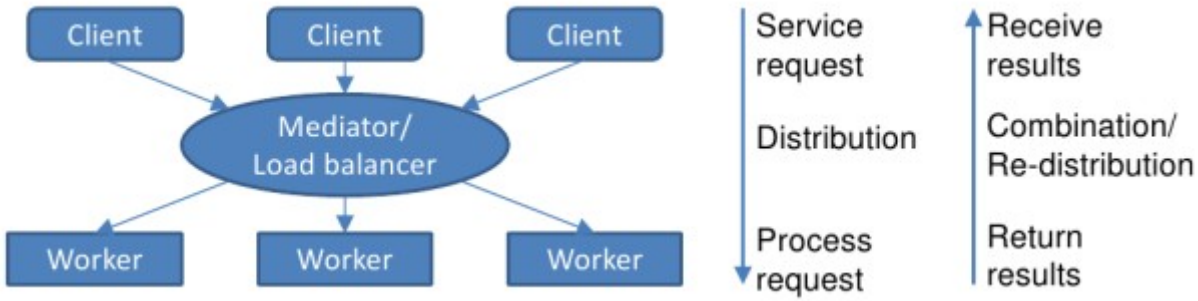
PerVoice

Client and Workers

Connection based communication with multiple service requests at the same time

Clients:
- Connect to the service architecture
- Send media streams (text, audio, image, video) to the service architecture
- May subscribe to one or multiple input streams of different type and language in order to get e.g. the translation results
- Examples: simple captioning (output: audio stream, input: automatically generated captions)

Workers:
- Register at the service architecture with one or multiple services that the worker is able to handle
- Accepts one incoming service request per connection
- Examples: ASR, SMT, MT, TTS …



PerVoice

Fingerprints

Fingerprints are used to specify the exact language and genere of a media stream

Structure of a fingerprint:

ll[-LL[-dddd]]

ll	language code (ISO369-1, 2 lowercase letters)
LL	optional country code (ISO3166, 2 uppercase letters)
dddd	optional additional flavor (n letters describing domain/ type/ version/ dialect etc.)

- Parts have to be separated by “-” (minus)
- If a language is spoken across several countries, the country code shall be the uppercased language code

Examples:

- en-EU European accented English
- de-AT German spoken in Austria
- en-US-weather US American English, weather domain

PerVoice

Input/ Output Types

- Stream types are used to specify the type of a media stream
- For a fully specified media stream, both the fingerprint and the stream type have to be given
- Some pre-defined stream types
  - **audio** audio containing speech
  - **image** image data, e.g. presentation slides
  - **text** properly formatted textual data, e.g. captions
  - **unseg-text** unsegmented textual data, e.g. ASR hypotheses

PerVoice

The mediation problem

- In order to accomplish a clients request, workers must be present that can convert a media stream specified by (input) fingerprint and type to the requested (output) fingerprint and type, e.g.
  - audio en-EU -> unseg-text en
- In case of mediation problems, backup strategies have been implemented:
  - Stream types must match
  - For fingerprints, the match is a score according to the criteria:
    - Exact match
    - Language code + country
    - Language code + domain
    - Language code
  - Example for en-EU-weather:
    - en-EU-weather > en-EU > en-US-weather > en-US

PerVoice

Packet types

- **Data:** Packets containing data that is exchanged between client and worker. Available data types:
  - Audio
  - Text
  - Image
- **Done:** Status message that is sent either when the client has sent all data, or the worker has finished processing all data (after a “done” from the client has been received)
- **Error:** Status message that is sent whenever an error occurs such as a broken connection
- **Reset:** Status message that is sent, whenever the client or worker should be reset to its initial state
- **Flush:** Status message that is sent, whenever the client or worker should finalize processing pending data and flush corresponding results

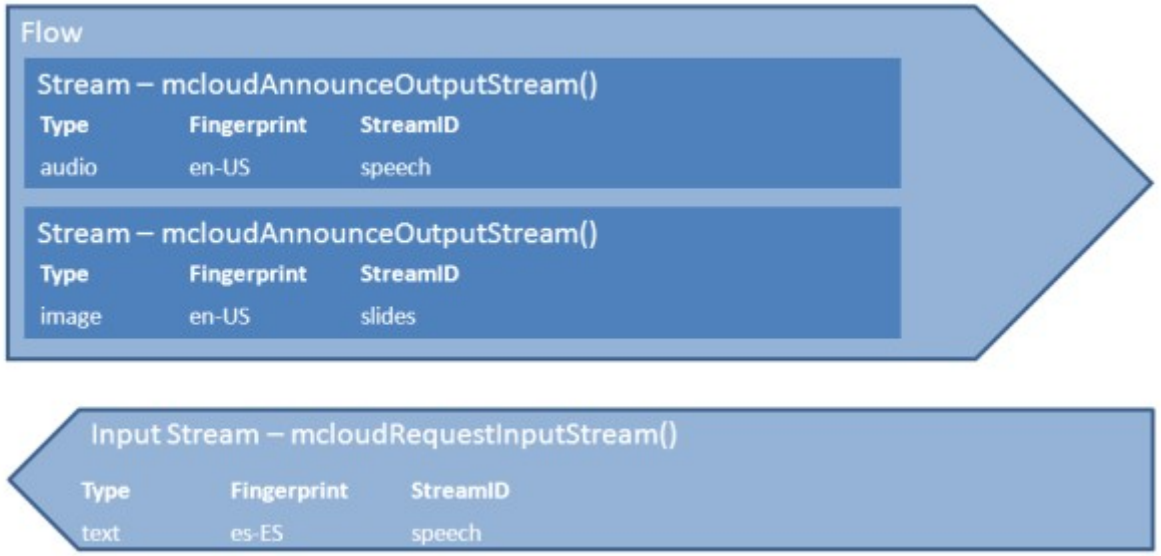
PerVoice

Client – Flows and Streams

- One flow per connection between client and mediator
- Multiple streams per flow (with different streamIDs)
- For each of the streams, service requests can be raised

PerVoice

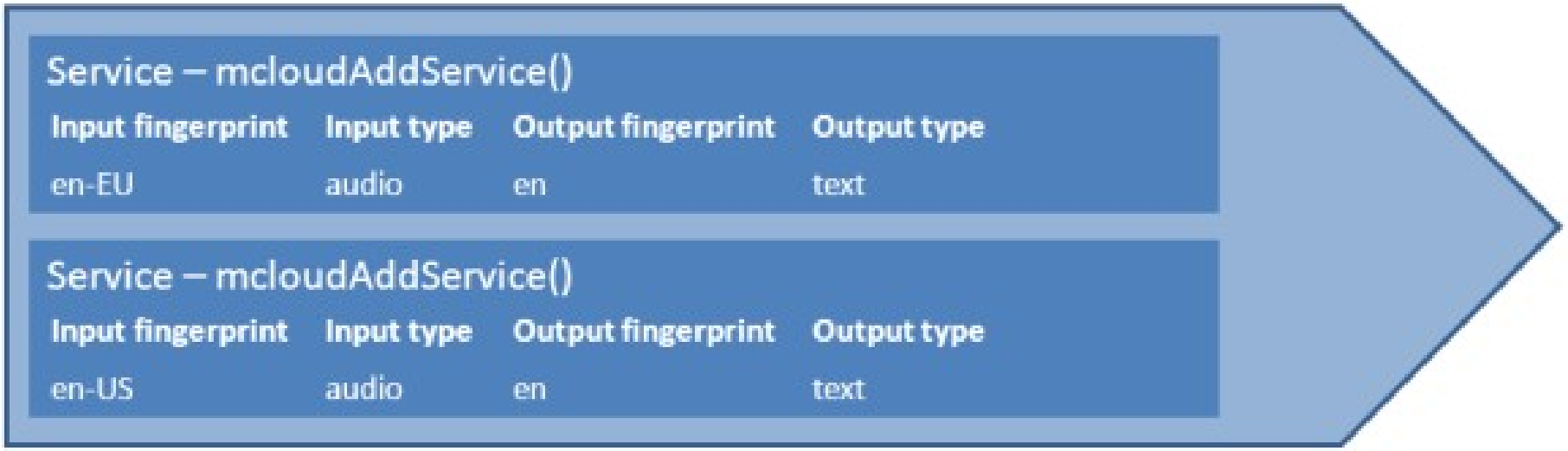
Client – Request for Input Stream



- Multiple requests for input streams are allowed
- Workers must be present in order to accomplish the request:
  - audio en-US -> text en-US and text en-US -> text es-ES

PerVoice

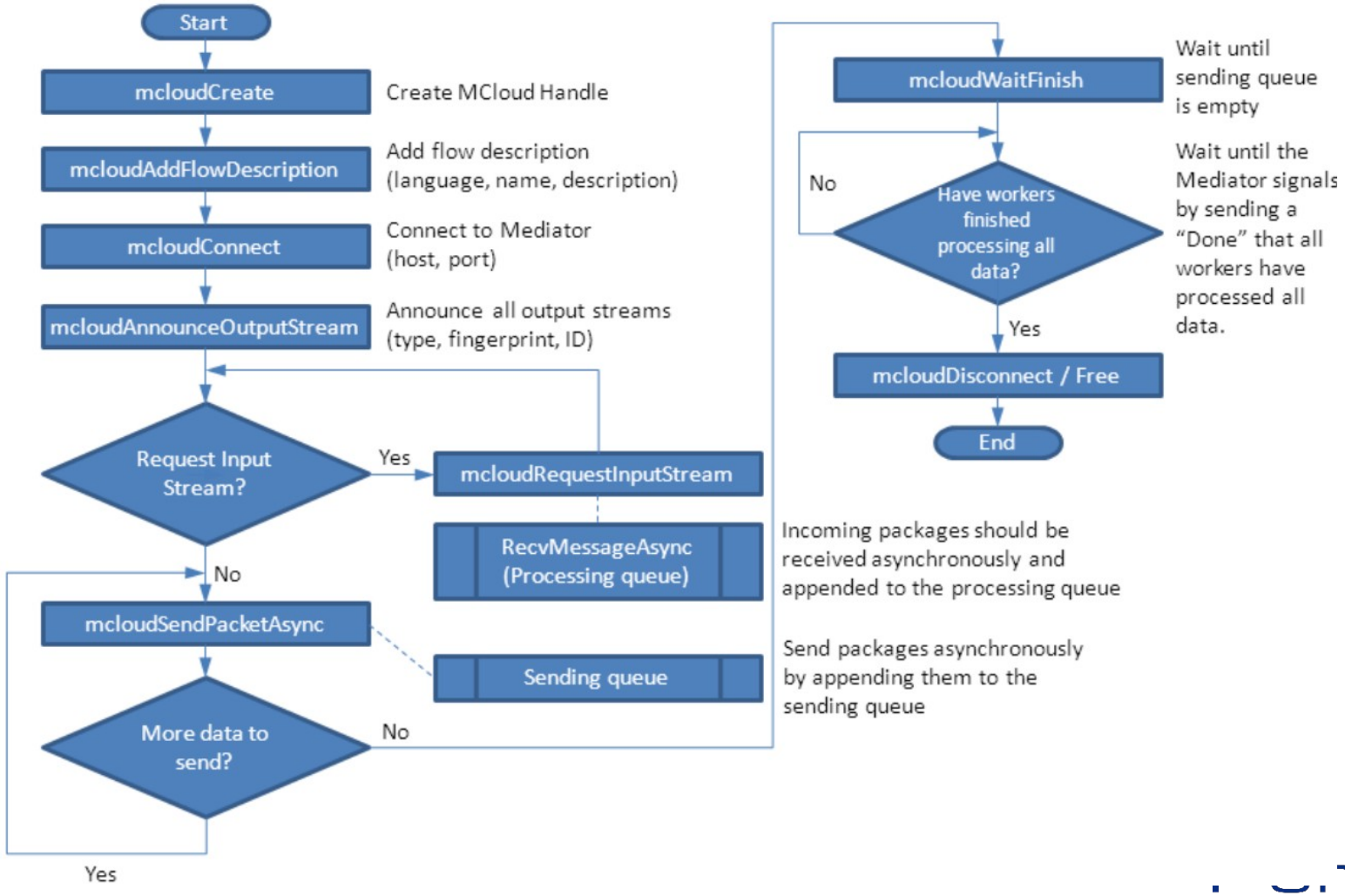
Worker – Service specification



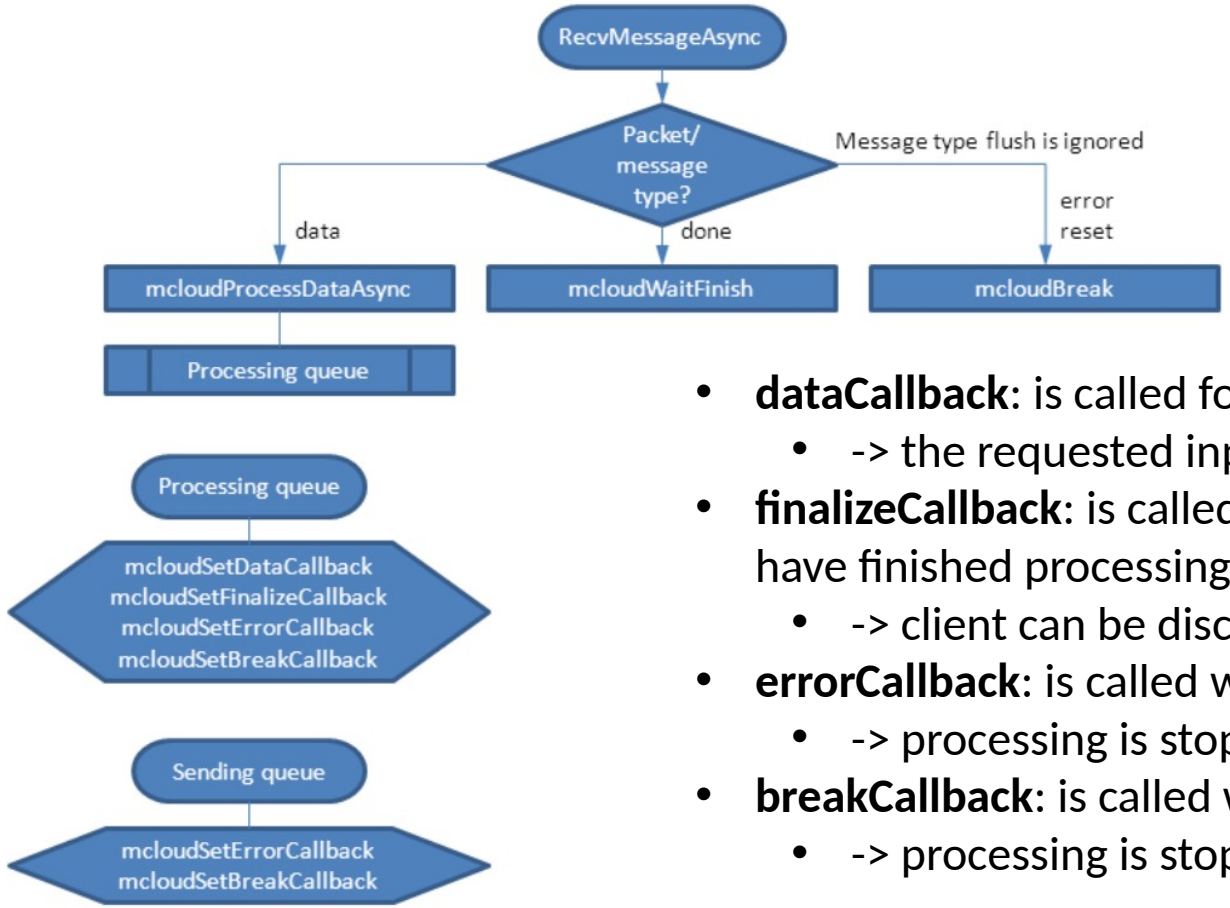
- A worker can offer multiple services
- Only one service can be active at the same time per connection

PerVoice

Client – Example Program Flow



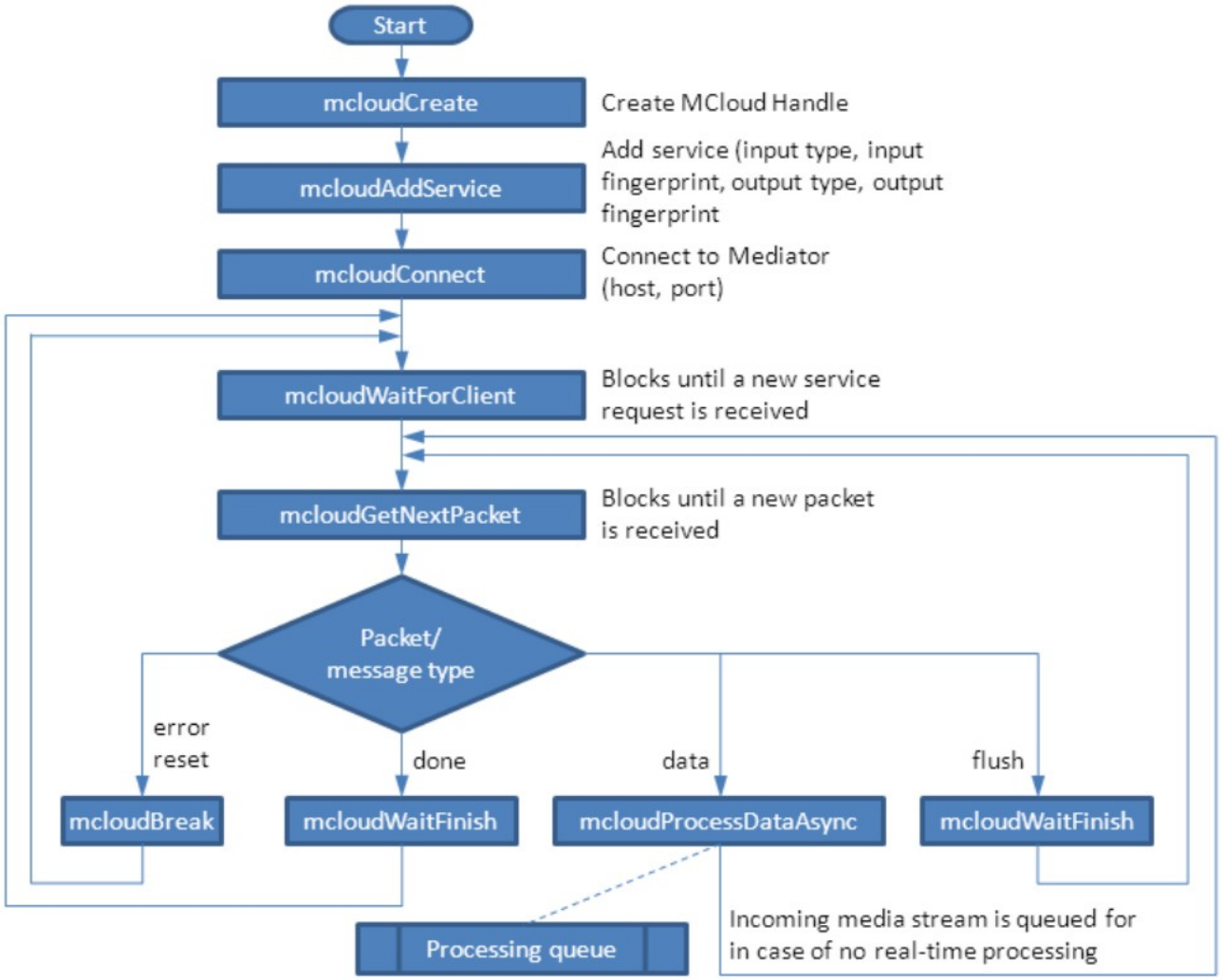
Client – Queues and Callbacks



- **dataCallback**: is called for each packet in the queue sequentially
  - -> the requested input stream results are handled
- **finalizeCallback**: is called when the Mediator signals that all workers have finished processing the data
  - -> client can be disconnected
- **errorCallback**: is called when an error occurs during internal processing
  - -> processing is stopped by resetting queues
- **breakCallback**: is called when mcloudBreak is called
  - -> processing is stopped by resetting queues

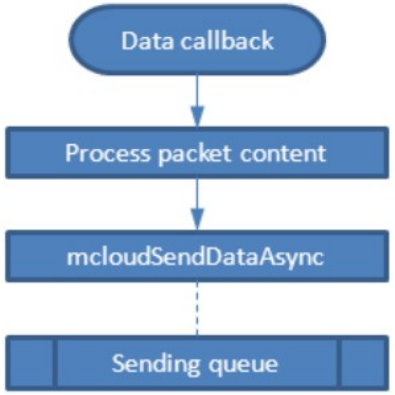
PerVoice

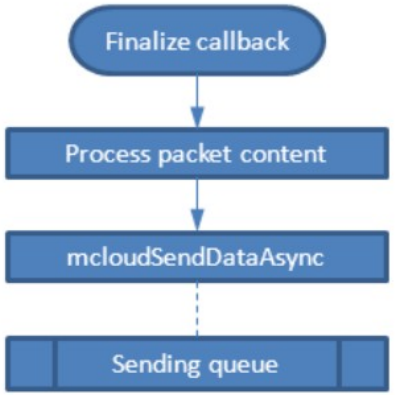
Worker – Example Program Flow

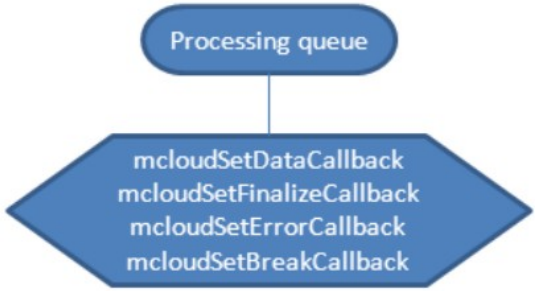


PerVoice

Worker – Queues and Callbacks







- dataCallback: is called for each packet in the queue sequentially
  - -> the packet content has to be processed and available results have to be sent asynchronously
- finalizeCallback: is called when the Mediator signals that the client has finished sending the data
  - -> process all pending packages
- errorCallback: is called when an error occurs during internal processing
  - -> processing is stopped by resetting queues
- breakCallback: is called when mcloudBreak is called
  - -> processing is stopped by resetting queues

PerVoice

Convenience Functions

- **mcloudPacket object**

```
MCloudType     packetType;        char          *start;
MCloudType     dataType;          char          *stop;
char           *sessionID;        char          *statusDescription;
char           *streamID;         char          *xmlString;
char           *fingerPrint;      xmlDoc        *doc;
char           *creator;
```

- **Prepare a packet for sending:**
  - `mcloudPacketInitFromAudio (startTime, stopTime, fingerPrint, sampleA, sampleN)`
  - `mcloudPacketInitFromImage (startTime, stopTime, fingerPrint, width, height, format, buffer, byteN)`
  - `mcloudPacketInitFromText (startTime, stopTime, fingerPrint, text)`
  - `mcloudPacketInitFromWordTokenA (startTime, stopTime, fingerPrint, tokenA, tokenN)`
- **Extract content from a packet:**
  - `mcloudPacketGetAudio`
  - `mcloudPacketGetImage`
  - `mcloudPacketGetText`
  - `mcloudPacketGetWordTokenA`

PerVoice

Convenience Functions

- **Set and get MCloud attributes**
  - `mcloudGetAttr`
  - `mcloudSetAttr`
- **For sample rate conversion, audio encoding/ decoding and compression**
- **Attributes**
  - `MCloudA_sAudioCodec, /**< Audio codec type [pcm, adpcm, speex, fcm] (get/set) (default: pcm) */`
  - `MCloudA_iSampleRate, /**< input/ output sample rate in Hz (get/set) (default: 16000) */`
  - `MCloudA_iSampleSize, /**< input/ output sample size in bits (get/set) (default: 16) */`
  - `MCloudA_iChannelN /**< input/ output number of channels (get/set) (default: 1) */`

PerVoice

Word Tokens

- All text packets are containing a normal text string
- In order to be able to pass additional information, e.g. from the ASR, an array of word tokens have to be used
- In this case, the text string is generated by the concatenation of the written forms of the word token array

```
struct MCloudWordToken
  int           index;          /* token index */
  char         *internal;       /* internal form of the token */
  char         *written;        /* written form of the token (can be NULL) */
  char         *spoken;         /* spoken form of the token (optional) */
  float         confidence;     /* confidence value in the interval [0,1] */
  unsigned int startTime;       /* start time [ms] relative to the start of
                                   the stream */
  unsigned int stopTime;        /* end tim [ms] relative to the start of the
                                   stream */
  int           isFiller;       /* set to 1, if token is a filler token and
                                   not a regular word */
```

PerVoice

Time Stamps

- Time stamps in the header are absolute time stamps within the media stream
- The difference between the absolute time stamp of a packet and the absolute time stamp of the beginning of the media stream, gives the position of the packet in the media stream
- The time stamps in the packet should exactly correspond to the content of the packet
- Time stamps in the word token array are relative time stamps within the media stream
- The beginning of the media stream is set to 0

PerVoice

We provided you...

https://github.com/ELITR/pv-platform-sample-connector.git

- C-library for Linux in binary form
- Header files
- Documentation
- C source code examples

Connection data:

URL: mediator.pervoice.com

NO_SSL_WORKER_PORT: 60021
NO_SSL_CLIENT_PORT: 4445
SSL_WORKER_PORT: 60419
SSL_CLIENT_PORT: 4444

PerVoice

ASR Worker – Implementation

```
mcloudCreate ()
mcloudAddService (en-GB, audio, en-GB, text)
mcloudSetDataCallback (userData) // set other callbacks if required

while (1) {
  mcloudConnect ()
  mcloudWaitForClient ()
  while (1) {
    proceed = 1
    while (proceed && p=mcloudGetNextPacket ()) {
      switch (p->type) {
        data:  mcloudProcessDataAsync (p); break
        flush: mcloudWaitFinish (); mcloudFlush (); break
        done:  mcloudWaitFinish (); proceed = 0; break
        error:
        reset: mcloudBreak (); proceed = 0; break
      }
    }
  }
}
```

PerVoice

ASR Worker – Implementation

```
// data callback function
dataCallback (p, userData) {
  switch (p->type) {
    audio: mcloudPacketGetAudio (p, &sampleA, &sampleN)
           asr_Decode
           asr_GetResult
           // copy result into token array
           p = mcloudPacketInitFromWordTokenA (startTime, stopTime,
                                               tokenA, tokenN);
           mcloudSendPacketAsync (p);
           break
  }
}
```

PerVoice

Client – Implementation

```
mcloudCreate ()
mcloudAddFlowDescription (English, Weather, \Weather Forecast")
mcloudConnect ()
mcloudAnnounceOutputStream (audio, en-GB, speech)
userData->proceed = 1
mcloudSetDataCallback (userData) // set other callbacks if required
if (wantInputStream) {
  mcloudRequestInputStream (text, en-GB, speech)
  recvMessagesAsync (userData); // install thread for receiving data
}
while (mediaDataIsAvailable) {
  p = mcloudPacketInitFromAudio (startTime, stopTime, fingerPrint,
                                 buffer, bufferSize, isFinal)
  mcloudSendPacketAsync (p)

}
mcloudWaitFinish ()
while (userData->proceed) sleep (500)
mcloudDisconnect ()
```

PerVoice

Client – Implementation

```
// receiving messages asynchronously
recvMessagesAsyncMain (userData) {
  p = userData->packet
  while (userData->proceed && p=mcloudGetNextPacket ()) {
      switch (p->type) {
        data:  mcloudProcessDataAsync (p); break
        flush: break
        done:  mcloudWaitFinish (); userData->proceed = 0; break
        error:
        reset: mcloudBreak (); userData->proceed = 0; break
      }
    }
  }
}

dataCallback (p, userData) {
  mcloudPacketGetWordTokenA (p, &tokenA, &tokenN)
  // so something with packet content such as storing the transcript
}
```

PerVoice

Thank you!

- For support please contact:

support@pervoice.it