Going through PerVoice Service Architecture Sample Code

Version 1.0, Gennaio 2019

Table of Contents

1. PerVoice Service Architecture short introduction

The PerVoice Service Architecture represents an efficient Distributed Architecture, which is characterized by message protocol hidden by the API, XML message format, communications based on TCP/IP sockets connections based and an asynchronous sending and processing of packets using queues and callback functions.

The PerVoice Service Architecture enables a connection-based communication with multiple service re-quests at the same time. A client connects to the mediator and the mediator connects the output media stream of the client with one or multiple workers in order to accomplish a specific service request.

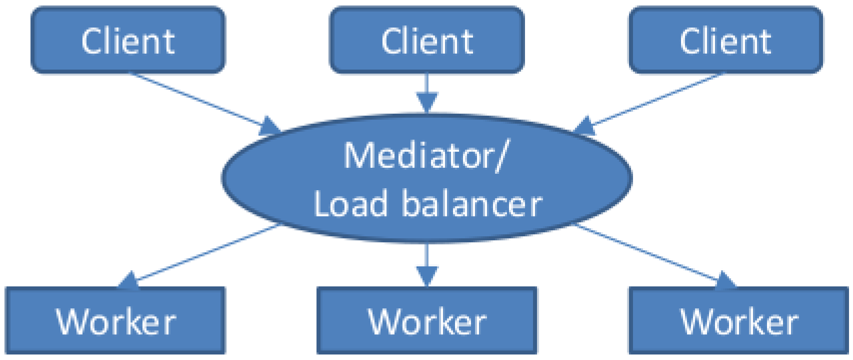

Figure 1. High level architecture

Main components are:

- **Mediator:** SCALA/AKKA server component of the PerVoice Service Architecture. The goal of the SCALA Mediator is the brokerage between client requests and workers. The system creates a pipeline of service able to satisfy a client request base of data type and content "signature". Therefore, the SCALA Mediator is a highly concurrent system able to serve multiple clients requests that involve multiple services at same time.
- **Client:** client component of the PerVoice Service Architecture. The Client instantiate service request to achieve specific tasks specified through input/output fingerprint. It connects to the PerVoice Service Architecture and sends media streams (text, audio, image, video) to it.
- **Worker:** service component of the PerVoice Service Architecture. The goal of the Worker is to accomplish atomic operations. It first registers at the PerVoice Service Architecture with one or multiple services that the worker is able to handle. Then it accepts incoming service requests and provides processed data. Each worker accepts only one incoming service request per connection, but, in order to accomplish each request, requests are queued internally for later processing.

Several clients can connect to a mediator which distributes the requests and load amongst several connected workers.

Please refer to doc/deliverable_D4.1.pdf for detailed description.

2. Directory structure

Let’s start checking the provided data structure

```
.
|-- audio
|   `-- talk1313.mp4-16kHz.wav
|-- doc
|   |-- deliverable_D4.1.pdf
|   |-- EBClient_Manual.pdf
|   |-- images
|   |-- MCloud_building_instructions.pdf
|   |-- MCloud.xsd
|   `-- refman.pdf
|-- include
|   |-- FingerPrints.h
|   |-- MCloud.h
|   |-- MCloudPThread.h
|   |-- Platform.h
|   `-- S2STime.h
|-- Linux
|   |-- lib32
|   |-- lib64
|   |-- makefile.linux
|   |-- README
|   `-- src
|-- pv-platform-sample-connector-v1.0.pdf
|-- README.md
`-- Windows
    |-- lib32
    |-- lib64
    |-- makefile.win
    |-- README.txt
    `-- src
```

Repo is organized to provide easy client/worker working examples. Then, under Linux and Windows directories you’ll find sources, libs and makefiles to build examples.

Please be sure to have libxml in your CPATH

```
export CPATH=/usr/include/libxml2/:$CPATH
export TARGET_PLATFORM=x86_64
export LD_LIBRARY_PATH=/path/to/Linux/lib64/
```

To build source code

```
make -f makefile.linux all
```

For more details have a look at doc/MCloud_building_instructions.pdf

For both Windows and Linux OS you will find README file, explaining how to run tests.

Please remember that the provided access data are temporary data and may change.

3. Worker example

You will find a couple of worker examples, but start from the backendASR2. It's just and example of ASR Worker integration, but it hasn't any ASR logic inside. It just returns always the same message.

Inside the code you will find comments explaining the methods and the logic. For a better comprehension of phases and messages, please have a look at doc/deliverable_D4.1.pdf chapter "1.3.5 Worker ? Example Program Flow"

4. Client example

In order to use a client for test, have a look at Detailed documentation in available in doc/EBClient_Manual.pdf

5. MCloud library

The library is already compiled and available in the following files:

```
./Linux/lib32/libMCloud.so.1.0
./Linux/lib64/libMCloud.so.1.0
./Windows/lib32/MCloud.exp
./Windows/lib32/MCloud.dll
./Windows/lib64/MCloud.exp
./Windows/lib64/MCloud.dll
```

Header files are the ones in the include directory:

```
FingerPrints.h
MCloud.h
MCloudPThread.h
Platform.h
S2STime.h
```

Objects are described by doc/MCloud.xsd

Complete library documentation is in link:doc/refman.pdf