D5.3 – Instruments technical description and development scheduling

July 31st, 2016

Author/s: Marcus Knuth (LPUS)

Contributor/s: Paulo Pinto, Bernardo Patrão, Jorge Oliveira, Marco Neves (WFSW / Keystroke Recognition / Global Architecture), Patrick Snape, Epameinondas Antonatos, Stefanos Zafeiriou (Imperial / Face Recognition), Sushil Bhattacharjee, Sebastien Marcel, Andre Anjos (Idiap / Voice Recognition and Anti-Spoofing), Hugo Jair Escalante, Manuel Montes (INAOE / Forensic Analysis), David Gañán (UOC / Plagiarism Analysis), Joaquin Garcia-Alfaro, Ender Alvarez, Christophe Kiennert (Security Techniques)

Deliverable Lead Beneficiary: LPLUS.

This project has been co-funded by the HORIZON 2020 Programme of the European Union. This publication reflects the views only of the author, and the Commission cannot be held responsible for any use, which may be made of the information contained therein.

Deliverable number or supporting document title	D5.3 Instruments technical description and development scheduling
Type	Report
Dissemination level	Public
Nature	Report
Publication date	31-07-2016
Author(s)	Marcus Knuth (LPLUS)
Contributor(s)	Paulo Pinto, Bernardo Patrão, Jorge Oliveira, Marco Neves (WFSW / Keystroke Recognition / Global Architecture), Patrick Snape, Epameinondas Antonatos, Stefanos Zafeiriou (Imperial / Face Recognition), Sushil Bhattacharjee, Sebastien Marcel, Andre Anjos (Idiap / Voice Recognition and Anti-Spoofing), Hugo Jair Escalante, Manuel Montes (INAOE / Forensic Analysis), David Gañán (UOC / Plagiarism Analysis), Joaquin Garcia-Alfaro, Ender Alvarez, Christophe Kiennert (Security Techniques)
Reviewer(s)	Abdulkadir Karadeniz (AU)
Keywords	Portal, TEP, Instruments, VLE, database, institution, instructor, learner, SEN, interface.
Website	www.tesla-project.eu

CHANGE LOG

Version	Date	Description of change	Responsible
V0.1	29/06/2016	Initial skeleton	Marcus Knuth
V0.2	01/07/2016	Technical descriptions added	Marcus Knuth
V0.3	04/07/2016	Introduction and approach added	Marcus Knuth
V0.4	11/07/2016	Development Schedule added	Marcus Knuth
V0.5	14/07/2016	Refinements	Marcus Knuth
V0.6	15/07/2016	TEP specifications added	Marcus Knuth
V0.7	18/07/2016	Changes due to review	Marcus Knuth
V0.8	21/07/2016	Last Changes due to review	Marcus Knuth
V1.0	21/07/2016	Ready for review by Technical Board	Marcus Knuth
V1.1	22/07/2016	Docker Container sub-chapters added, other refinements based on David's comments.	Marcus Knuth
V1.2	22/07/2016	Changes due to comments from Xavier and Jorge	Marcus Knuth

V1.3	22/07/2016	Peer Review Version	Marcus Knuth
V1.4	28/07/2016	Additions and Corrections due to comments in peer review	Marcus Knuth

Table of Contents

1 Introduction 7
1.1 Objective 7
1.2 Approach 7
2 Instruments Specifications 9
2.1 Face Recognition and Verification (IMPERIAL) 9
2.1.1 Enrolment 9
2.1.2 Data Format 9
2.1.3 Data Storage 9
2.1.4 Technologies 9
2.2 Face Presentation Attack Detection (FPAD) 9
2.2.1 Enrolment 10
2.2.2 Data Format 10
2.2.3 Data Storage 10
2.2.4 Technologies 10
2.3 Automatic Speaker Verification (IDIAP) 10
2.3.1 Enrolment 10
2.3.2 Data Format 10
2.3.3 Data Storage 10
2.3.4 Technologies 11
2.4 Voice Presentation Attack Detection (VPAD) 11
2.4.1 Enrolment 11
2.4.2 Data Format 11
2.4.3 Data Storage 11
2.4.4 Technologies 11
2.5 Plagiarism Analysis Tool (UOC) 11
2.5.1 Enrolment 12
2.5.2 Data Format 12
2.5.3 Data Storage 12
2.5.4 Technologies 12
2.6 Authorship Verification (INAOE) 12
2.6.1 Enrolment 12
2.6.2 Data Format 12
2.6.3 Data Storage 12

2.6.4 Technologies 12
2.7 Keystroke Dynamics (WSFW) 13
2.7.1 Enrolment 13
2.7.2 Data Format 13
2.7.3 Data Storage 13
2.7.4 Technologies 13
2.8 Time Stamping Instrument (IMT) 13
2.8.1 Enrolment 13
2.8.2 Data Format 14
2.8.3 Data Storage 14
2.8.4 Technologies 14
3 Analysis Result 15
3.1 Programming Languages 15
3.2 Data 15
3.3 Invocation 15
3.4 Security 15
3.5 Scalability 15
3.6 Licences 15
3.7 Input and Return Values 15
3.8 Privacy 16
4 Global Architecture and Integration Concept 17
4.1 Global Architecture 17
4.2 Instrument Integration Concept 18
4.2.1 Instrument Environment 18
4.2.2 Instrument Communication 19
4.2.3 Docker container introduction 20
4.2.4 Docker Container Package (DCP) for instrument developers 22
4.3 TeSLA e-Assessment Portal (TEP) 23
4.3.1 Unified instrument routing and involvement 26
4.3.2 Configuration 27
5 Development Schedule 29
5.1 First Iteration M8 to M12 29
5.2 Second Iteration M13 to M18 30
5.3 Third Iteration M19 to M24 30
6 Conclusion 31

1 Introduction

The TeSLA system provides authentication and authorship verification to Virtual Learning Environments (VLEs). Therefore, it uses instruments developed by different organisations and companies.

The instruments have been developed without the scope of an integrated system like TeSLA so there have been no overall requirements in terms of programming languages, databases and there like. The challenge of integrating instruments from different organisations and companies is the mix of technologies that have been used.

WP5 focuses on the integration of instruments developed by different organisations and companies. The TeSLA system will use these instruments to verify authentication and authorship during local and remote examinations. For authentication the available instruments cover face detection and verification, voice verification and keystroke dynamics. Face presentation attack detection (FPAD) and voice presentation attack detection (VPAD) assist the face detection and verification instrument as well as the voice verification instrument in detecting cheating attempts. For the authorship verification a plagiarism detection instrument (PACPlagi) and an authorship recognition instrument are available. All of those instruments combined will allow a reasonable indication if the student and his document are authentic or not.

1.1 Objective

This document will summarize all technical aspects of the instruments to be integrated. It will show how the integration will be implemented based on the global architecture developed and presented by Watchful Software in D6.2.

Also this document will define a development roadmap and a schedule for the instruments integration.

1.2 Approach

Prior to this document all instruments partners have been asked to give detailed information about their instruments during T5.1. The collected information contains features, technologies and information concerning the use of the instruments and how their results have to be interpreted. The Information from the instrument partners was the objective of D5.1.

During T5.2 the information from D5.1 has been analysed and related with the outcome of the limitations given by WP3 (privacy and ethics) and WP4 (quality). The results of the analysis were the objective of D5.2.

The global architecture takes account of outcomes from WP2 (educational framework model), WP3 (data privacy and ethics), WP4 (quality assurance) and the collected information regarding the instruments, their used technologies and how to use them. Watchful Software has been designing the global architecture during T6.2 and presented the results in D6.2

In Chapter 2 we will present the technical specifications of each instrument. The instruments are stand-alone products developed for a specific purpose and without any relation to the TeSLA project. Here we will understand what the instruments do, what kind of data they expect, what they will return and how to interpret the return values. This chapter will briefly describe the results from D5.1.

In Chapter 3 we will present the results of the analysis and group them by relevant topics as a base for the global architecture. This chapter will briefly describe the results from D5.2.

In Chapter 4 we will explain the details of the instruments integration based on the global architecture given in D6.2, the technical specifications of the instruments and the results of the analysis in chapter 3.

In Chapter 5 we will determine the tasks to be executed by the instruments developers based on the instrument integration concept presented in chapter 4 and present the schedule of the upcoming implementation phase for the instruments.

2 Instruments Specifications

In this chapter we will present the technical specifications of all instruments. We will explain their goals, what kind of data they collect, what kind of data they store, which technologies they use and if they require an enrolment phase. All instruments require an ID of a candidate to compare the input with. The instruments have no knowledge about private data of the candidate in terms of name, day of birth, etc. For the instruments the candidate is just an ID.

2.1 Face Recognition and Verification (IMPERIAL)

The face recognition and verification instrument analyses visual data such as images or videos and tries to recognize a face within the given data. It will compare the characteristics of a candidate shown on an image or video against characteristics of the candidate stored in a database or files system as image, video or model. A webcam and a browser extension is required to capture images or videos. Recognition works on images so if a video has been delivered as an input it will generate images from the video.

2.1.1 Enrolment

During the enrolment phase a model of the characteristics of a face in a series of images or generated images from a video will be build up. After that the images and videos are not required any longer and may be deleted, but it is also possible to keep them in the database or in the file system.

2.1.2 Data Format

Images or videos with a minimum resolution of 640 px x 480 px is required for face recognition. In case of a video its length should be at least 10 seconds in order to generate the required pictures. The face within the images or videos should cover at least an area of 150 px x 150 px. Slightly different angles of the face help making the face recognition more accurate. The preferred data format is PNG or JPG.

The result of the instrument will be a floating point between 0 and 1, where 1 represents a perfect match. That means the characteristics of a sample has been successfully compared to the stored model if the result is 1 and there haven't been found characteristics of the input compared to the model if the result is 0. The return value represents the probability of the candidate being the same person as stored in the model.

2.1.3 Data Storage

Received data for this instrument can be stored in a database or file system. Once a Model as has been created during the enrolment all received data can be deleted.

2.1.4 Technologies

The face recognition and verification instrument has been implement using Python but can be ported to C++ for performance issues if required. In Python the instrument does not work in real-time. The instrument can be run on Linux and MacOS platforms. Windows might be possible also but this has not been tested yet.

2.2 Face Presentation Attack Detection (FPAD)

This instrument tries to detect presentation attacks to the face recognition instrument. Such an attack can be executed using a photo or video of another person on an

electronic device such as a smartphone, a tablet or a computer-screen. A printed photograph of another person as well as 3D mask might also be used to attack the face recognition. 3D masks will not be in the scope of the TeSLA project. If the presentation attack is successful a candidate may authenticate as another candidate just by using an image and showing it to the camera. The FPAD assists the face recognition and verification instrument as it has to be a real person in front of the camera and not an image whose characteristics should be compared with the candidate's characteristics in the database.

2.2.1 Enrolment

There is no enrolment for this instrument. The accuracy can be increased by training the instrument using (anonymous) student data collected under realistic conditions.

2.2.2 Data Format

The FPAD expects exactly the same input as the face recognition and verification instrument so the same data can be passed to both instruments. For the FPAD to work properly the candidate should be close enough to the camera so that the minimum distance between the centers of the eyes is 50 pixels.

The FPAD will return a floating point number indicating the probability of a face presentation attack.

2.2.3 Data Storage

Received data for this instrument can be stored in a database or file system.

2.2.4 Technologies

The face presentation attack detection instrument is using the same technology as the face recognition and verification instrument. Python with the Bob package on Linux and MacOS.

2.3 Automatic Speaker Verification (IDIAP)

The automatic speaker verification instrument will be used to verify a candidate's identity by comparing the characteristics of the voice with a model that has been derived from an example of speech during enrolment. In order to record audio signals there has to be a microphone connected to the computer.

2.3.1 Enrolment

A set of speech samples together with the knowledge about the gender of each candidate is required

2.3.2 Data Format

Uncompressed recorded audio file with a minimum resolution of 16 kHz.

The instrument will return a floating number between 0 and 1 indicating the probability of the correct identity of the candidate.

2.3.3 Data Storage

Collected data for this instrument can be stored in a database or file system. The enrolment data will be stored in an uncompressed audio format. The collected data of a specific authentication attempt must be stored during the recognition process. After the attempt is finished the data can be deleted. Depending on the required accuracy that has

to be determined in the pilots the amount of data to store can become quite large. An audio sample in CD quality requires approximately 10 MB per minute.

2.3.4 Technologies

The automatic speaker verification instrument has been implement using Python. In Python the instrument does not work in real-time. The instrument can be run on Linux and MacOS platforms. Windows might be possible also but this has not been tested yet. Furthermore, this instrument uses the Bob package for Python.

2.4 Voice Presentation Attack Detection (VPAD)

The objective of this instrument is to detect attacks in the voice presentation. A candidate might try to authenticate by using a pre-recorded speech of another candidate. Like the FPAD assists the face recognition and verification instrument, the VPAD assists the automatic speaker verification in avoiding wrong authentication by simply playing an audio file in front of the microphone.

2.4.1 Enrolment

No enrolment is needed for this instrument to work. The instrument will need to be trained using speech samples from anonymous candidates under various conditions ranging from studio-quality to realistic scenarios.

2.4.2 Data Format

Like the automatic speaker verification, the VPAD expects uncompressed audio files with a minimum resolution of 16 kHz.

The FPAD instrument will return a floating point number between 0 und 1 indicating the probability of a presentation attack.

2.4.3 Data Storage

Data collected for this instrument can be stored in a database or file system. During a specific detection process the collected data must be stored and may be deleted afterwards.

2.4.4 Technologies

The VPAD instrument has been implemented using Python. The instrument can be run on Linux and MacOS platforms.

2.5 Plagiarism Analysis Tool (UOC)

The PACPlagi instrument has been developed to detect word-for-word copies in a given set of documents. It receives a document and compares it to all other documents in the set searching for word-for-word copies. At the moment this tool is not ready to be run in parallel because in a scenario where document A is already being compared to the set of documents and document B is also being compared to the set of documents both documents wouldn't be compared to each other. Therefor this instrument requires serialisation. In later versions this may change. Also this tool does not support the search for plagiarism in external databases, web pages or search engines – it only compares documents in its own database.

2.5.1 Enrolment

This instrument does not need enrolment or training. It uses all documents in the set in the database to find word-for-word copies in a received document.

2.5.2 Data Format

The instrument receives a text document in a common format (i.e. pdf, doc, txt) for the comparison to be executed.

The instrument will return an integer between 1 and 100, where 100 represents a perfect match between to documents which means they are equal. The instrument might update results for more than one document as it cannot know which document has been written first. If a word-for-word-copy has been found the probability for both documents will be adjusted accordingly.

2.5.3 Data Storage

All documents will be stored in the files system whereas all the results from the instruments will be stored in a database. The collected documents cannot be deleted after the comparison, because this instrument conducts a cumulative comparison.

2.5.4 Technologies

The PACPlagi has been implemented using Java and will run on Linux, Windows or MacOS environments. Currently an Oracle database is being used but the database can be replaced by any other database.

2.6 Authorship Verification (INAOE)

Authorship verification verifies that a document has been written by a specific author. To be able to do so it has to be trained with a set of text files written by the author.

2.6.1 Enrolment

This instrument has to be trained for every candidate whose documents will be verified. It needs a set of documents written by the candidate and builds up a model for the candidate.

2.6.2 Data Format

The instrument accepts unformatted text document as an input.

It will return a floating point number between 0 and 100, where 100 represents that the author of a given sample compared to the enrolment documents has been successfully verified.

2.6.3 Data Storage

A database or a file system can be used to store received text files. All files will be kept to constantly improve the recognition model.

2.6.4 Technologies

This instrument has been implemented using Python on Ubuntu Linux, NLTK and SCIKIT LEARN libraries.

2.7 Keystroke Dynamics (WSFW)

This instrument recognises patterns based on the timing information from pressed and released keys when a candidate is typing on a keyboard. Two key features can be extracted from a standard keyboard connected to a Personal Computer: the amount of time each key is held down (dwell time), and the elapsed time between the release of the first key and the depression of the second key (flight time). Based on these features a user profile will be created. On this basis, matching with an attempt sample can be conducted to predict authentication. This instrument collects every keyboard input from the candidates.

2.7.1 Enrolment

For the enrolment of this instrument at least 30 samples have to be collected, which should contain dwells and flights and which should be extracted from 125 consecutive pressed keys.

2.7.2 Data Format

The keyboard dynamic instrument will receive a list with formatted samples containing key-up and key-down events with a timestamp.

The result will be a floating point number between 0 and 1 representing the probability that the sample is from the same candidate who has presented input during enrolment.

2.7.3 Data Storage

Collected data will be stored in a Database. During the recognition process the received data has to be stored, afterwards the data can be deleted.

2.7.4 Technologies

The instrument has been implemented in Java und uses commercial software from Watchful Software as well as a commercial database which requires a license agreement. By the time of writing this document the implementation has been ported to Java and an open-source database.

2.8 Time Stamping Instrument (IMT)

The instrument limits to the client-side scope of the specifications in [RFC-3161, RFC-5816]. It limits to what an RFC 3161 enabled time-stamping client may support. Time-stamping server functionalities are defined in [RFC-3161, RFC-5816].

The instrument will be used as an interface requesting a Trusted Third Party, referred to as Time Stamping Authority (TSA) server to time-stamp a datum in order to establish evidence that such time datum existed before a particular time.

To perform the task, it must create a valid request. The request must be decoded by the TSA. If no errors are found by the TSA, the instrument shall obtain a timestamp token, as specified in Section 2.4 of [RFC-3161]. A hash of the datum to be time-stamped forms the request, among other control and optional fields. The algorithm used to create the hash should conform to the specifications in [RFC-5816].

2.8.1 Enrolment

No enrolment or training is required for this instrument.

2.8.2 Data Format

The input information is defined by the specification in [RFC-3161, RC-5816]]

2.8.3 Data Storage

Time-stamp tokens and request data will be stored on the server.

2.8.4 Technologies

The instrument is expected to be compliant to [RFC-3161, RFC-5816] and future related recommendations. The instrument needs to be implemented and can be developed using existing cryptographic libraries.

3 Analysis Result

Now that the technical specifications of all instruments have been described we will focus on different aspects of the implementation, environment, usage, etc. This will allow us to understand the arguments for the global architecture and the integration concept which will be explained later.

3.1 Programming Languages

The instruments use a wide range of programming languages including C++, C#, Java, and Python. Some of these languages compile to native code others compile to bytecode run by runtime environments and one is an interpreted high-level programming language with 3rd party tools to create an executable with integrated interpreter.

3.2 Data

To store information, instruments are using relational databases and file systems. Most tools use open-source Databases like MySQL and PostgreSQL. But also the features of commercial database manufacturers are used by instruments.

3.3 Invocation

Most instruments are prepared to be executed on a command line environment and don't support web services of their own. They expect program calls with parameters containing the required data and instructions to perform their operation. Also each instrument has its own set of parameter.

3.4 Security

As most instruments are called on a command line environment security is generally considered to be part of the environment they run on. Especially databases should be executed in a secure environment.

3.5 Scalability

Most instruments are not prepared for multi-thread use and can only serve one request at a time. Although the instruments are not prepared for multi-thread use by program code on their own, multiple instances can be run to process more than one request at a time. Only the plagiarism instrument cannot be run in parallel processes at the same time due to logical and functional reasons at this point of the project. This might change in later versions. For now, all request to the plagiarism instrument will have to be serialized in some way.

3.6 Licences

No commercial software or database system without an open-source license is used by the instruments excepted the Keystroke Dynamics tool which is being ported to open-source technologies.

3.7 Input and Return Values

All instruments need specific input data such as video, audio, a sequence of an input text and documents to process requests. For the instruments to work in the context of a VLE it is required to be able to capture images or video, audio and keystrokes. So in terms of hardware a webcam, a microphone and a keyboard will be required. Most of the

instruments will need a general or candidate specific training to work or at least to return best result possible.

All instruments return one single number per request. The face detection and verification instrument, the automatic speaker verification instrument, the keystroke dynamics instrument and the authorship verification instrument return a number indicating the probability of a match between a stored candidate profile or model and a candidate who has created the authentication request. In case of the presentation attack instruments the probability of an attack will be returned without the need of a candidate's model. In case of the plagiarism instrument the number indicates the percentage of similitude between the sent document with documents to be compared in the database. The plagiarism tool will also update the results of other documents that already have been compared to the documents in the database because they may have similar text and the tool cannot detect which document has been written first so it cannot estimate who copied from whom so it will raise the probability of both documents.

3.8 Privacy

All instruments are able to process requests without knowing details of the candidate - excepted for the voice recognition instrument that requires to know the gender. Indeed, they do save data related to a candidate but this data does not have to be connected to other private data of a candidate. So for the instruments to work only partial private data has to be presented to the instruments while the selection of a candidate's profile to compare with can be solved via a masked ID.

4 Global Architecture and Integration Concept

So far we have explained the instruments and we have also summarized aspects of their implementation. In order to understand the integration concept, the global architecture will be described briefly in this chapter. A complete description of the global architecture is the objective of D6.2 from Watchful Software.

The explanation of the global architecture is followed by a description and details of the integration concept for the instruments, that is fundamental to understand the tasks and schedule of the instrument integration.

4.1 Global Architecture

The global architecture has been developed by Watchful Software during T6.2 and it has been described by D6.2. It takes into account all requirements of the educational model (WP2), privacy and ethics (WP3) and quality (WP4) together with fundamental state-of-the-art software development principles. It takes into account modularity, performance, scalability, security, privacy, licenses, requirements of SEN learners and performance. This sub-chapter will briefly explain the concepts of the global architecture while concentrating on the subject of instrument integrations.

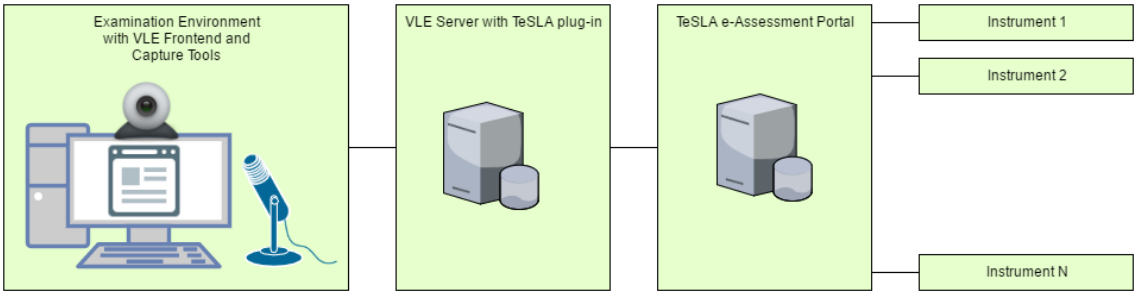


Figure 1: Overview TeSLA instrument integration

First of all, for the instruments to work data will have to be collected at the VLE side and it has to be send to TeSLA system. Therefor a plug-in for the VLE will be developed and installed. A set of capture tools belonging to the plug-in will be responsible for collecting video, audio, keystrokes and documents which will then be send to the TeSLA system by the TeSLA plug-in itself.

The TeSLA plug-in will send the data to the TeSLA e-Assessment Portal (TEP) utilizing a restful web service. Also it will replace the candidate's VLE ID through a global unique masked TeSLA ID. The data being send by the TeSLA plug-in contain no other private information as the data collected by the tools.

The TEP is a message broker and forwards requests with collected data to the instruments also using restful web services.

One or more instances of an instrument receive requests, queue it and later process it asynchronously and return the result of the evaluation back to the TEP where it will be stored for later use. Although this is an asynchronous process the instruments will return a status code directly after having received the sample to indicate problems with it. The status code will be returned to the plug-in immediately (synchronously) so the user interface can give hints of what is going wrong. That way the user can react to the given hint, i.e. move the face closer to the camera or make sure the microphone is close enough to the mouth.

Once a person of a valid role inside the VLE will ask for the results, the TeSLA plug-in will ask the TEP for the results and the latter one will return the stored results. Both is also performed using restful web services.

An organisation might use more than one VLE. In this case every VLE that intends to use TeSLA system will have its own plug-in. All TeSLA plug-ins within an organisation will communicate with the same and only TEP. The TEP will forward requests to the appropriate instrument. One or more instances of one instrument might be installed and connected to the TEP.

4.2 Instrument Integration Concept

Following the global architecture that has been described in the previous sub-chapter we will provide a detailed view on the integration of the instruments.

We start by explaining how the instruments will be run, how they will be made ready for multi-threaded use and how communication works.

4.2.1 Instrument Environment

The key to execute the instruments that have been developed using different technologies in one system is to give every instrument its own environment and connect them using web services. This will be realized using Docker container. Docker container is a light weight virtualization technology that will enable the instrument developers to install their instrument in an environment as if it was a dedicated server. Every instrument will have its own Docker container so that the installations will not affect each other. Docker container has been developed having the concept of composition of micro-services in mind.

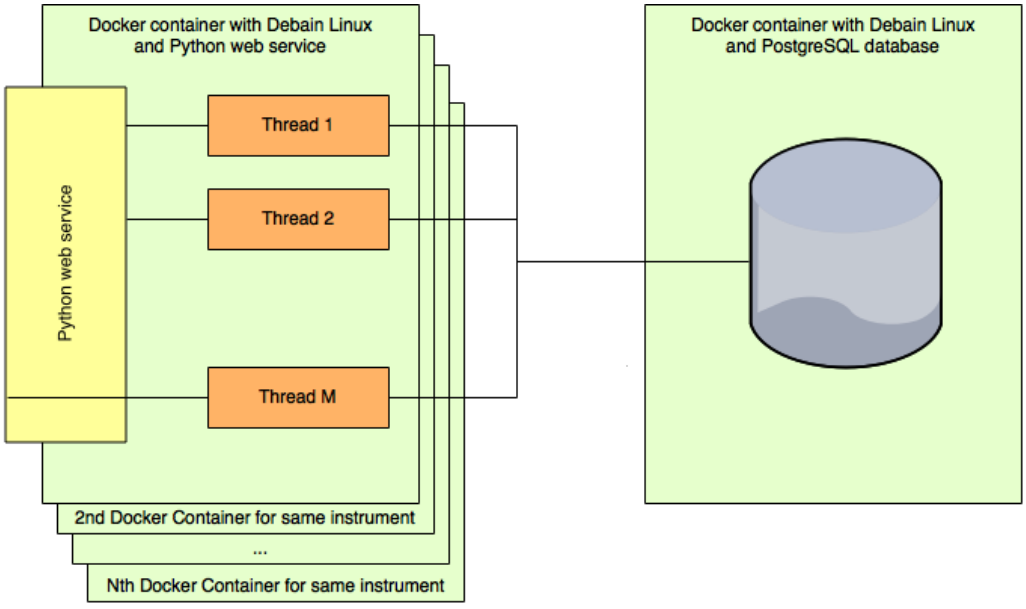


Figure 2: Docker container environment for instrument developers

As all instruments are able to run on Linux a free-to-use Debian Linux has been chosen to be installed as a base. Most instruments do not provide a web service interface so Python and a Python web service will be installed on the Debian Linux installation. Developers can install any libraries and tools required for their instrument installation without affecting other instruments.

The Python web service will receive the request and save it to a database for later processing. By saving incoming requests in a database first it is ensured that instruments can make use of serialization by running only one instance and one thread of their instrument. On the other hand, multiple instances with multiple threads of the instrument can be run if required that will pull requests out of the database and process it if the instrument allows this from a functional point of view. Instances of Docker containers have to be installed and started manually by an administrator and can be added to TeSLA system via the UI of the TeSLA e-Assessment Portal as explained later. In later versions of TeSLA system, a Docker Container management allowing TeSLA system to add/remove Instances on demand may be implemented. All instances and threads of an instrument will use the same database that will be hosted in a separate Docker container environment. The database for all instruments to be used will be an open-source, free-to-use PostgreSQL database installed on a Debian Linux Docker container. Instrument developers will have to make sure that storing results or model updates will work with more than one instance of the instrument by using database transactions. Also if temporary files are used, developers will have to make sure that threads being executed in the same Docker container instance will use distinct file names or folder names.

Instrument developers will receive a package containing two Docker container. One Docker container will be a Debian Linux with Python installed, having a demo web service with methods on how to receive and store request. It will also show how to use more than one thread at a time so even in one instance multiple requests can be handled simultaneously. The second Docker container will be Debian Linux with PostgreSQL database installed to be used by all instances of the running instrument.

Instrument developers will have to:

1. Study the Docker container package provided by Watchful Software; read the documentation
2. Install Docker container environment if not present
3. Install their instrument and all required libraries in the Docker container
4. Change or add database access code for the PostgreSQL database if required
5. Make sure database access from multiple threads/instances don't affect each other by using database transactions
6. Make sure temporary files from multiple threads in the same Docker container do not cause conflicts or wrong behavior
7. Add Code to the web service to store requests in the database
8. Make worker threads pull requests from the database and pass it to the instrument to be processed, store the result to the database and send it back to the TEP.

4.2.2 Instrument Communication

All data captured by the capture tools will be send to the TEP in a unified way to allow modularity. The request will be forwarded to an appropriate instance of the instrument. The communication will use restful web services and the communication to the instruments will be standardized.

An evaluation request for an instrument contains the following information:

API: POST /evaluation/new

1. TeSLA id (string): a unique and masked identifier of a candidate
2. Evaluation id (integer): the id of the request to be referred to later when sending the result back to TEP
3. Shared key (string): authentication data to verify an authorized source for the request (this will be most likely the TEP)

4. Sample Data (byte array). Containing the captured data to be processed

As soon as a request has been received the instrument should perform a sanity check and report back the quality of the sample together with an error code (integer) to indicate the problem. The reported status code will be transported back to the candidate's user interface so orders to adjust the capture of data can be given (e.g. "No face detected – please make sure you are sitting in front of the camera").

If enrolment is required a data request for an instrument contains the following information:

API: POST /enrolment/sample

1. TeSLA id (string): a unique and masked identifier of a candidate
2. Shared key (string): authentication data to verify an authorized source for the request (this will be most likely the TEP)
3. Sample Data (byte array): containing the captured data to be processed
4. Gender id (integer): 0 = not defined, 1 = male, 2 = female (only to be used by voice recognition, not defined otherwise)

As soon as this request has been received the instrument will process the sample data and it will return a status code like in the evaluation request above as well as a floating point number which will indicate the enrolment percentage (i.e. 0.5 = half of enrolment phase is done).

All instruments will communicate the result of an evaluation to the TEP using the same function of its web service. A call to this function will contain the following information:

1. Evaluation id (integer): same id as in the request
2. Evaluation result (float): the computed result of the instrument
3. Audit data (byte array): additional data for later analysis of the result if required
4. Shared key (string): authentication data to verify an authorized source for the request

The TEP will respond with a status code where 0 means everything went OK.

4.2.3 Docker container introduction

In this sub-chapter we will explain Docker containers in general.

Docker containers wrap a piece of software in a complete filesystem that contains everything needed to run in terms of code, runtime, system tools, system libraries – anything that can be installed on a server. This guarantees that the software will always run the same, regardless of its environment.

You can think of a Docker container as layers of file system. For example, one could create a layer of a Debian Linux base installation and then add a layer with all required components to run a webserver plus own code which will be hosted by the webserver.

Docker Container share hardware resources in an own secured Docker engine. Compared to other virtualization technologies Docker containers do not have to emulate a whole operating system. The following picture compares the concepts of virtual machines with the concept of Docker containers.

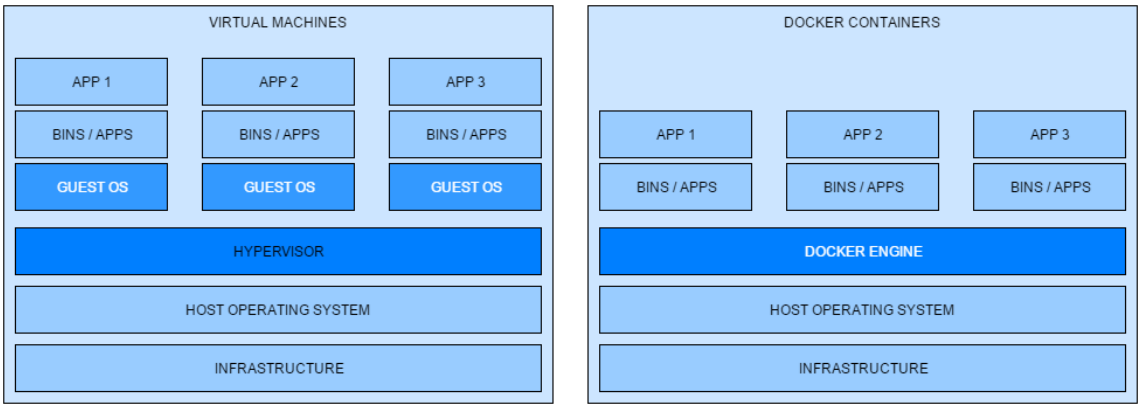


Figure 3: Docker Container vs Virtual Machines

A Docker Container is created by using a script which is called a Dockerfile and which has to be compiled to produce a Docker image that can be hosted on a Docker engine.

Inside a Dockerfile you can receive pre-made scripts from the Docker Hub – a global directory for Dockerfiles – execute commands and copy files into the image. There are more functions available – see Docker documentation on www.docker.com for details.

A simple Dockerfile would look like this:

```
# A basic apache server. To use either add or bind mount content under /var/www

FROM ubuntu:12.04

RUN apt-get update && apt-get install -y apache2 && apt-get clean && rm
-rf /var/lib/apt/lists/*

ENV APACHE_RUN_USER www-data

ENV APACHE_RUN_GROUP www-data

ENV APACHE_LOG_DIR /var/log/apache2

EXPOSE 80

CMD ["/usr/sbin/apache2", "-D", "FOREGROUND"]
```

This Dockefile tells the Docker compiler to receive Ubuntu Linux version 12.04 from the Docker hub. On the Ubuntu 12.04 an apt-get command will be run to install apache webserver. After that, environments variables for user, group and log file directory are set, port 80 is made a public port and a command for running the apache server is called.

After compiling this script with the Docker compiler command `'docker build'` the image will be built – and therefor the apache webserver in its own environment based on Ubuntu Linux can be run on the Docker engine by using the `'docker run'` command.

As mentioned before using Docker containers you can think of using layers of file system running in an own secured user space. Once a Docker image has been compiled it can be run and all changes to the file system will be stored in an own layer so that the layers below stay the same. The same image can be used to spawn more than one application from the same image.

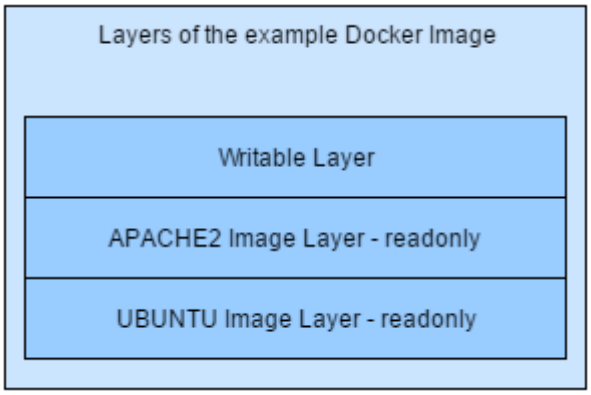


Figure 4: Docker Layers of the example Dockerfile after compilation

After having set up the Docker image with required software and libraries, you can copy your website into the Docker image by using a copy command in the Dockerfile. For the example above this could be the following command to be inserted right before the CMD command in the script:

```
ADD /mywebsite /var/www/mywebsite
```

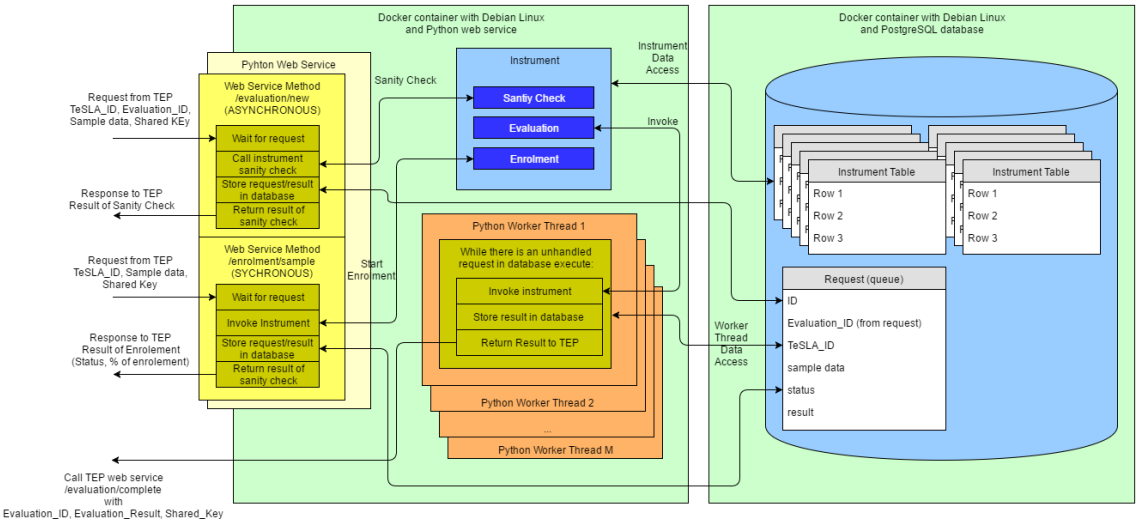
4.2.4 Docker Container Package (DCP) for instrument developers

Now that we have given a brief introduction to Docker containers, the Docker Container Package (DCP) to be used by the instrument developers provided by Watchful Software and how it is working will be explained.

The Package consist of the following elements:

1. A Dockerfile for the instrument with
    a. Debian Linux
    b. Python language
    c. PostgreSQL-Client libraries
    d. An example script of database settings and database access
    e. Python demo web service with
        i. Method /enrolment/sample
        ii. Method /sample/save
        iii. Method for invoking the worker thread
        iv. Method for calling the TEP web service to return results
2. A Dockerfile for the instruments database with
    a. Debian Linux
    b. PostgreSQL Database
3. A database structure for the request queue
4. A dummy TEP web service script to be used for testing purposes during development
5. A file upload mini website in HTML/Javascript for testing purposes
6. A complete documentation on how to use the package for the implementation of the integration

The following picture shows how the instrument integration works in detail:



The Python web service provides two methods.

Method ‘evaluation/new’ will call the instruments sanity check, store the request in the request table of the database and will return the result of the sanity check to TEP. One or more Python worker threads will constantly look into the request table if a not processed request is present. If so they will load it, invoke the instruments evaluation routines, store the result in the database and send the result to TEP. Sanity check should be as fast as possible as the web service method will wait for its results. Evaluation works asynchronously and can therefor take a long amount of time without blocking the system.

Method ‘enrolment/sample’ will call the instruments enrolment routines, store the request and the result to the database and will respond to TEP with the results from the check. This is a synchronous operation which means that the web service method will wait for the end of the enrolment routine.

The instrument stores its data in own tables which have to be added to the schema script already provided with the package. Instrument developers will eventually also have to write/change data access code if they didn’t use PostgreSQL before.

Instrument developers will retrieve a complete working example with all Python scripts for worker threads, web service methods and data access in the package.

4.3 TeSLA e-Assessment Portal (TEP)

In the scope of the instrument integration the TeSLA e-Assessment Portal (TEP) component is connected to all the instruments and the TeSLA plugin. It acts as message broker and it is the service for the VLE.

It will have the following features:

- Unified Instrument routing and involvement (requests, responses)
- Configuration (license(s), instruments, SEN learners)
- License verification and authentication
- Provide a filter mechanism for the instruments to be used by SEN learners
- Send statistics to the TeSLA Portal (not in the scope of this documents)

All users accessing the TEP user interface will have one of the following roles:

1. Institution Super User
    a. Access to all views
    b. Add license and instruments and monitor the system

2. Institution User
    a. Monitor the system (system view)

The TEP UI will consist of the following views:

1. License View
    a. General information and configuration of instruments
    b. License management (upload) and validity of license
2. System View
    a. Instrument usage metrics per instrument given by the number of requests
    b. System load of the TEP component and response times for requests
    c. Errors lists
3. System Management View
    a. Instruments status (active/inactive)
    b. Add or remove instruments

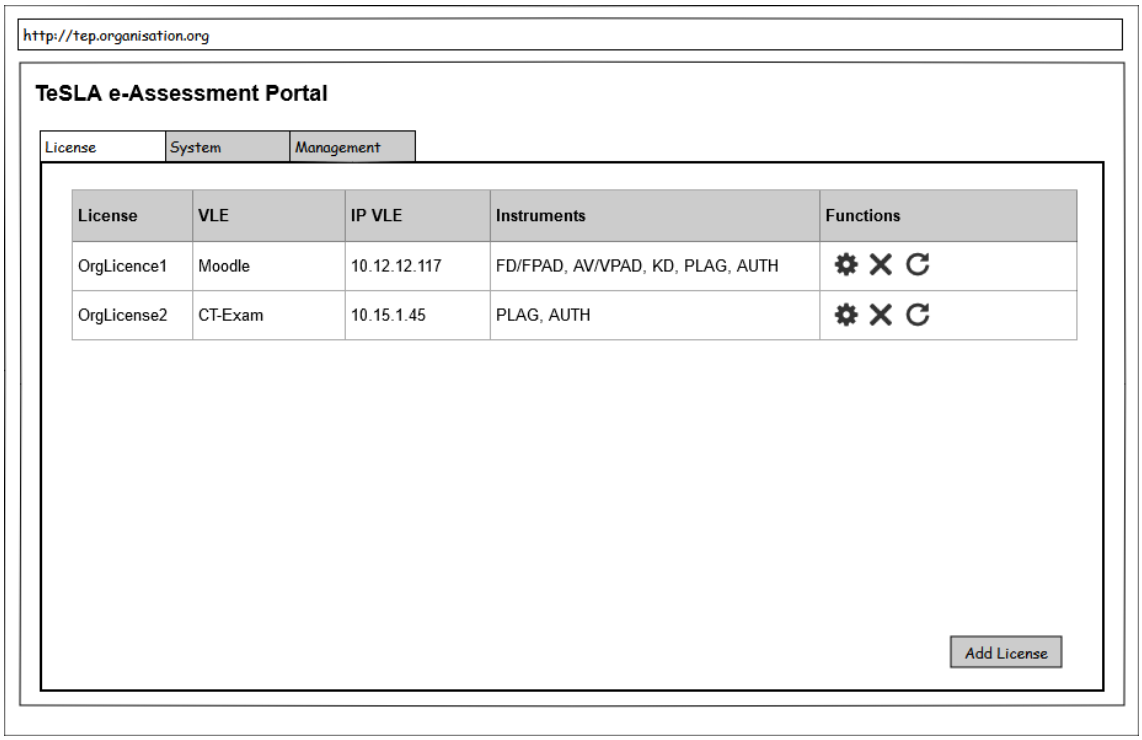


Figure 5: License View

In the license view all licenses which have been added are shown together with the instruments that are allowed to be used with this license. Functions for editing licenses details, deleting licenses and updating licenses are available for each licence as well as a function to add a license are available. Each VLE is required to have an own license with an own set of instruments that can be used.

Figure 6:System View

The system view shows the system load, response times, instrument usage in a graphical format covering average, maximum, minimum values as well as a list of errors below the graphs.

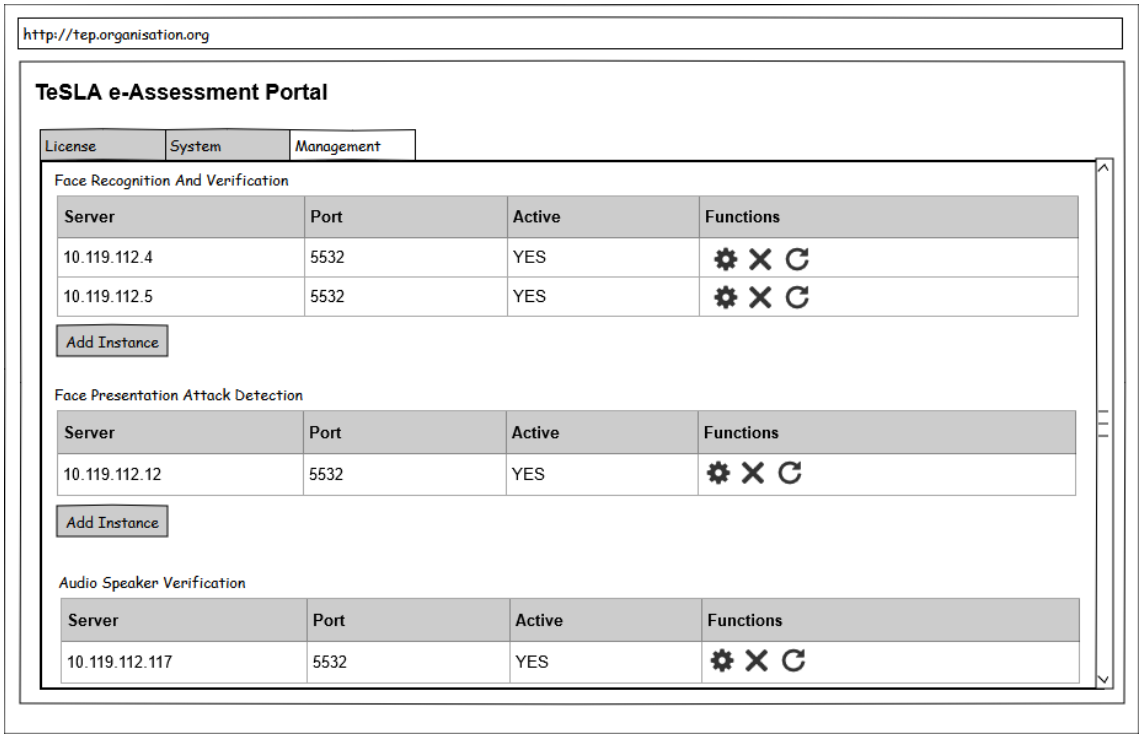


Figure 7: Instrument Management View

The management view will allow to see the status of all instrument instances that have been activated together with their status where 'active' means the instance is responding to requests. The number of instrument instances is not limited by the license and more instances can be added at any time. Server IP and server port are essential details for

the TEP to be able to connect to the instrument instances. Functions for editing instance details, delete instance and refresh instance details are provided for each of the instances. If no instance of an instrument is given or all instances of the instrument are not active no requests can be answered.

4.3.1 Unified instrument routing and involvement

Once an enrolment or evaluation request has been received from the TeSLA plug-in, TEP will look up available instrument instances in its database and forward the request to one of the instances using a round-robin approach. Later this might change to a balancing mechanism based on the load of the instrument instances. The appropriate instrument will be chosen by the instrument id in the request message.

The plug-in and the TEP share a list of instruments associated with instrument IDs. Messages for all instruments are unified as explained in the previous chapter and contain a global unique TeSLA ID representing a candidate. TEP will never gain control over private data as private data is never being send to the TEP. A candidate will be represented by TeSLA ID only and every VLE connected to the TEP is supposed to deliver the same TeSLA ID so that enrolment and training need to be done only once for all VLEs. This is only valid if the user ID presented by all VLEs is the same. If there are different user IDs for one candidate in different VLEs than enrolment will be needed for each VLE as the TeSLA system only refers to TeSLA IDs that will be derived from the user ID.

Communication between TEP and instruments has already been described in a previous chapter. Communication between the TeSLA plug-in and the TEP concerning enrolment and evaluation requests will be:

- Enrol sample: send data to the instrument to be used during enrolment phase
  - Request from plug-in:
    - TeSLA id (string): masked ID of the candidate
    - Instrument id (integer): instrument to deliver the data
    - Shared key (string): shared secret to secure communication
    - Sample data (byte array): data to be used for enrolment
    - Gender id (integer): 0 = NotDefined, 1 = male, 2 = female (only to be used by voice recognition, NotDefined otherwise)
  - Response from TEP:
    - Status code (integer): “0” = okay
- Request enrolment status: the percentage of enrolment completion and the current phase of enrolment for an instrument
  - Request from plug-in:
    - TeSLA id (string): masked ID of the candidate
    - Instrument id (integer): instrument to deliver the data
    - Shared key (string): shared secret to secure communication
  - Response from TEP:
    - Enrolment phase id (integer): id of the enrolment phase
    - Enrolment completion (float): percentage of completion
- Request learner evaluation: process the data by an instrument
  - Request from plug-in:
    - TeSLA id (string): the masked ID of the candidate
    - VLE id (integer): the ID of the VLE that sent the request
    - Instrument id (integer): the instrument to deliver the data
    - Shared key (string): shared secret to secure communication
    - Sample data (byte array): the data to be used for evaluation
    - Activity type (integer): the type of activity the evaluation will be for

        - Activity id (integer): the id of the activity the evaluation will be for
    - Response from TEP:
        - Status code (integer): “0” = okay
- Request TeSLA evaluation sample: retrieve the sample data of an evaluation from the instrument
    - Request from plug-in:
        - Evaluation id (integer): id of the evaluation to use for the sample data to return
        - Shared key (string): shared secret to secure communication
    - Response from TEP:
        - Data type (integer): type of data (i.e. 0 = audio, 1 = mp4-video)
        - Sample data (byte array): sample data which has been used for the evaluation
- Request TeSLA evaluation retrieve all results from all instruments used in an activity
    - Request from plug-in:
        - TeSLA id (string): the masked ID of the candidate
        - VLE id (integer): the ID of the VLE that sent the request
        - Shared key (string): shared secret to secure communication
        - Activity type (integer): the type of activity the evaluation will be for
        - Activity id (integer): the id of the activity the evaluation will be for
    - Response from TEP:
        - Special need (boolean): indicates if a learner has special needs
        - Special need code (integer): special needs code
        - For every instrument being used in the activity (array):
            - Evaluation id (integer): id of the evaluation
            - Instrument id (integer): id of the instrument that performed the evaluation
            - Evaluation result (float): result of evaluation as estimated by the instrument

4.3.2 Configuration

The TEP provides more functions to the plug-in so that by the help of the tools for learners and instructors information about agreement information and acceptance, state of enrolment, available instruments, activity setup, instrument and enrolment phase descriptions as well as SEN details can be retrieved and updated.

- Request learner agreement information: retrieve the status of a learner’s agreement and a list of enabled instruments for the activity requested
    - Request from plug-in:
        - TeSLA id (string): the masked ID of the candidate
        - VLE id (integer): the ID of the VLE that sent the request
        - Shared key (string): shared secret to secure communication
        - Activity type (integer): the type of activity the evaluation will be for
        - Activity id (integer): the id of the activity the evaluation will be for
    - Response from TEP:
        - Agreement Status (boolean): indicates if learner has accepted the agreement
        - For every enabled instrument for this activity (array):
            - Instrument id (integer): instrument
- Update learner agreement: store that learner has accepted agreement
    - Request from plug-in:
        - TeSLA id (string): the masked ID of the candidate
        - Shared key (string): shared secret to secure communication
    - Response from TEP:
        - Status code (integer): “0” = okay

- Request TeSLA options: retrieve a list of instruments that can be used by the VLE
  - Request from plug-in:
    - VLE id (integer): the ID of the VLE that sent the request
    - Shared key (string): shared secret to secure communication
  - Response from TEP:
    - For every enabled instrument for this VLE (array):
      - Instrument id (integer): id of the instrument that can be used
      - Instrument locked (boolean): indicates if instructor is allowed to change the configuration
    - For every available action if learner does not accept the agreement (array):
      - Alternative Code (integer): ID of the alternative
      - Alternative Blocked (boolean): indicates if the instructor is allowed to change the configuration
- Setup TeSLA activity: set the list of enabled instruments for an activity
  - Request from plug-in:
    - VLE id (integer): the ID of the VLE that sent the request
    - Shared key (string): shared secret to secure communication
    - Activity type (integer): the type of activity the instruments will be set for
    - Activity id (integer): the id of the activity the instruments will be set for
    - For every enabled instrument for this activity (array):
      - Instrument id (integer): the instrument to be used
      - Required (boolean): indicates if the instrument has to be used in the activity
      - Alternative Code (integer): alternative to be used instead of the instrument if learner declined the agreement
  - Response from TEP:
- Request TeSLA activity status: retrieve whether an activity has TeSLA enabled or not
  - Request from plug-in:
    - VLE id (integer): the ID of the VLE that sent the request
    - Shared key (string): shared secret to secure communication
    - Activity type (integer): the type of activity
    - Activity id (integer): the id of the activity
  - Response from TEP:
    - Enabled (boolean): true if TeSLA is enabled for this activity
- Request instrument description: retrieve the description of an instrument
  - Request from plug-in:
    - Instrument id (integer): the instrument for which the description is being requested
    - Shared key (string): shared secret to secure communication
  - Response from TEP:
    - Instrument description (string): description of the instrument
- Request SEN information: retrieve SEN information
  - Request from plug-in:
    - SEN id (integer): id
    - Shared key (string): shared secret to secure communication
  - Response from TEP:
    - SEN description (string): description
- Request enrolment phase description: retrieve description of the enrolment phase
  - Request from plug-in:
    - Phase id (integer): the ID of the phase to retrieve the description of
    - Shared key (string): shared secret to secure communication
  - Response from TEP:
    - Phase description (string): description of the phase

5 Development Schedule

According to the project plan and the given timetables a schedule for the implementation of the instruments integration has been developed.

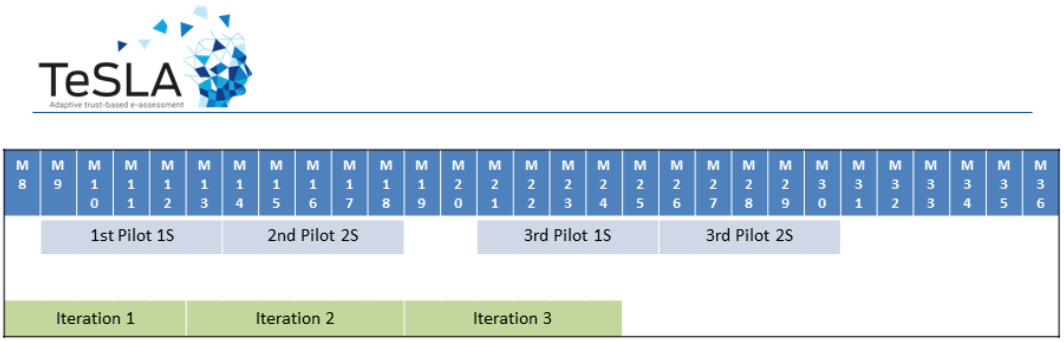
TeSLA project will provide required tools for the integration in terms of a central place to store results (i.e. GIT). Details of the development infrastructure will be send out to the instrument developers at the beginning of the implementation.

The implementation of the instruments integration will use an iterative approach and like the schedule of the pilots the implementation will have three phases:

1. Initial implementation
2. Bug fixes and performance enhancements 1
3. Bug fixes and performance enhancements 2

These phases will be synchronous to the following pilots:

1. First pilot M9-M13 – without TeSLA system but with a prototype to collect realistic data for instrument calibration
2. Second pilot M14-M18 – using initial feature complete TeSLA system implementation. Collect more realistic data in realistic scenarios. Find bugs and performance issues.
3. Third pilot 1S M21-M25 – using bug fixed and enhanced second version of TeSLA.
4. Third pilot 2S M26-M30 – using bug fixed and enhanced third version of TeSLA



- Iteration 1:
  - Initial implementation and integration for 2nd Pilot
  - First Unit Tests
- Iteration 2 and 3:
  - Bug fixing
  - Trouble shooting
  - Enhancement performance
  - Implementing tests (Unit Test, Mass operation tests)

5.1 First Iteration M8 to M12

In the first iteration instrument developers will receive the Docker Container Package containing one Docker container for their instrument and another Docker Container for

their database. They will get familiar with the Docker Container Package eventually supported by LPLUS and Watchful Software.

All instrument developers should be able to integrate their solution into the Docker Container in parallel while TEP, TeSLA plug-in and data collectors will be implemented also.

At the end of M12 all instruments should be integrated and connected to the TEP which will be connected to the TeSLA plug-in. In M13 the iteration will end with a test installation prior to the 2nd pilot to ensure its functionality and eventually react on critical bugs, performance issues or other show stoppers. The goal is to provide a working TeSLA system even if it is not feature complete or fully functional.

This iteration concentrates on the functionality and integration of the instruments. It does not cover performance enhancements, scalability or mass tests in general which is up to iterations two and three.

5.2 Second Iteration M13 to M18

During the second iteration more realistic data from the 2nd pilot becomes available so that the instrument developers are able to adjust their instruments in terms of accuracy and quality. Also measurements of the performance and usability will become available. Probably the requirements will slightly change after the 2nd pilot.

Reported bugs from the 2nd pilot may be fixed and if necessary the pilot installation may be updated depending on the priority of the bugs.

At the end of M17 all development and enhancements should be finished for the 3rd pilot so that the TeSLA system can be installed and tested properly in M18 before the pilot starts.

Instrument developers should have developed most important tests at this stage (if not present or if not using Test Driven Development).

5.3 Third Iteration M19 to M24

The final iteration will concentrate on performance, scalability and the implementation of tests if not finished.

6 Conclusion

TeSLA System will provide instruments to prove authentication and authorship. The integration of the instruments of all partners will be executed in parallel and by the use of micro services hosted in light weight virtual machines every instrument partner will be able to use an environment as required. No dependencies of one instrument will conflict with the dependencies of another.

All instruments will be installed in Debian Linux free-to-use operating system and they will make use of a free-to-use, open-source database.

Using Docker containers for instrument integration will make instrument integration easy, scalable and secure.