transLectures

Transcription and Translation of Video Lectures

D5.3.2: Second report on integration

UPVLC, XEROX, JSI-K4A, RWTH, EML and DDS

Distribution: Public

transLectures
Transcription and Translation of Video Lectures

ICT Project 287755 Deliverable D5.3.2

SEVENTH FRAMEWORK PROGRAMME

Project funded by the European Community under the Seventh Framework Programme for Research and Technological Development.

Information Society Technologies

Project ref no.	ICT-287755
Project acronym	transLectures
Project full title	Transcription and Translation of Video Lectures
Instrument	STREP
Thematic Priority	ICT-2011.4.2 Language Technologies
Start date / duration	01 November 2011 / 36 Months

Distribution	Public
Contractual date of delivery	April 30, 2014
Actual date of delivery	April 30, 2014
Date of last update	April 30, 2014
Deliverable number	D5.3.2
Deliverable title	Second report on integration
Type	Report
Status & version	v1.0
Number of pages	45
Contributing WPs	all
WP / Task responsible	JSI-K4A
Other contributors	UPVLC, XEROX, RWTH and EML
Internal reviewer	Jorge Civera, Alfons Juan
Author(s)	UPVLC, XEROX, JSI-K4A, RWTH, EML and DDS
EC project officer	Susan Fraser

The partners in transLectures are:

Universitat Politècnica de València (UPVLC)
XEROX Research Center Europe (XEROX)
Josef Stefan Institute (JSI) and its third party Knowledge for All Foundation (K4A)
RWTH Aachen University (RWTH)
European Media Laboratory GmbH (EML)
Deluxe Media Europe (DDS)

For copies of reports, updates on project activities and other transLectures related information, contact:

The transLectures Project Coordinator
Alfons Juan, Universitat Politècnica de València
Camí de Vera s/n, 46018 València, Spain
ajuan@dsic.upv.es
Phone +34 699-307-095, Fax +34 963-877-359

Copies of reports and other material can also be accessed via the project's homepage:
http://www.translectures.eu

Executive Summary

This document contains details about WP5 task 5.3 **Integration into the case studies** with regards to the work done in the last 12 months of the task. Task 5.3 itself will continue until the end of the project.

1. Introduction

The objective of task 5.3 is to incrementally integrate the models and tools developed in WP3 (Massive Adaptation) and WP4 (Intelligent interaction with users) into VideoLectures.NET and poliMedia. This will be achieved through the following means:

- VideoLectures.NET and poliMedia workflows and players will be modified to start using the results from these WPs and WP6 (Evaluation).
- Real-time serving of transcriptions/translations via the transLectures Web Service will be implemented.
- Some optimizations will be done to maximize scalability and response effectiveness.

This document describes the work done in the last 12 months of the task and is divided into 3 sections:

- Development of the transLectures Platform (TLP)
- Integration of the transLectures Platform into VideoLectures.NET and poliMedia
- Relationship with other work packages

2. Development of the transLectures Platform (TLP)

During the M18-M30 period most of the efforts have been pushed to the development and integration of transLectures Platform into the case studies. The transLectures Platform (TLP) includes all components developed in order to integrate transLectures transcription and translation technologies into remote repositories. The components are the transLectures Database, Web Service, Ingest Service and Player. The source code is being continuously and steadily improved, and so will be until the end of the project.

2.1. Database

The transLectures database is at the core of the transLectures Platform. It is a PostgreSQL relational database which stores all the data required by the Web Service. Specifically, the transLectures database stores the following entities:

- **Lectures**
  All information related to each lecture is stored in the Database, such as external ID, language, duration, title, keywords and category.
- **Speakers**
  Minimal information about speakers is stored into the database: name, gender, age, accent, etc. This information could be exploited by the ASR system through adapting the underlying models to the speech peculiarities of an specific speaker. In addition, we have extended the information we store about speakers with their e-mail addresses, attending to the 2nd recommendation formulated by the project reviewers at the end of the 2nd year: *Include the authors of the transcriptions and translation in the workflow in order to have vital feedback on the quality.*
- **Captions**
  All captions generated by the ASR and MT systems are kept in the Database and retrieved by the Web Service. Captions are stored in DFXP format (see Annex A).
- **Media**
  By default, a local copy of the remote repository is maintained in the transLectures server and the Database keeps organized and accessible all uploaded media files.
- **Uploads**
  Every time an Ingest operation is requested by the remote repository via the Web Service, a new upload entry is stored in the database. Then the Ingest Service takes over the upload and processes the data.

2.2. Web Service

The transLectures Web Service is the interface for exchanging information and data between the remote repository and the transLectures system. It also enables the subtitle visualization and editing capabilities of the transLectures Player. This Web Service is implemented as a Python Web Server Gateway Interface (WSGI), and defines a set of HTTP interfaces related to caption delivery and lecture upload:

- ***ingest***
  POST request which allows the client to upload directly from the recording studio audio/video files and other related material such as slides and textual resources, bundled in a Media Package file (see Annex C).

- ***status***
  GET request to check the status of a video lecture uploaded through the */ingest* interface.
- ***lecturedata***
  GET request that provides metadata and media files location to the transLectures Player.
- ***langs***
  GET request that provides the client with a list of captions and languages available for a given lecture.
- ***dfxp***
  GET request that returns captions in DFXP format (see Annex A) for an specific lecture and language.
- ***mod***
  POST request to send and apply the changes made by a user when editing a transcription or translation.

Detailed and updated documentation of every interface of the transLectures Web Service API can be found in Annex B.

During this reporting period, the Web Service's code was reimplemented and logically split into two different layers for the sake of portability and genericity. The upper layer implements the Web Service API, whilst the lower layer deals with local implementation specifics of getting and storing the data.

The Web Service API was extended with a new interface */lecturedata*, which was needed in order to establish full independence of the transLectures Player on the basis of a distributed architecture.

Also, we have modified the code to allow lecture ingestion forwarding to external service providers (ESP, for example EML's transcription Web service) via external plug-ins that can be registered during the Web Service's load. These plug-ins allow slight modifications of the API of the */ingest* and */status* interfaces according to the ESP's requirements.

Finally, DFXP format was modified by adding a new *status* attribute at the document level. This new attribute shows the supervision status of the whole DFXP file (no supervision, partially supervised, fully supervised). Detailed documentation of the DFXP format can be found in Annex A.

2.3. Ingest Service

After the prototype of the Ingest Service was successfully tested, the UPVLC has developed a new implementation of the Ingest Service, which handles and properly processes Media Package files that have been uploaded through the */ingest* interface of the Web Service. It is implemented as a Python module that has to be executed periodically (typically every minute) to check whether new lectures have been uploaded for the processing and in addition to assess whether uploads, that are being processed, are making progress or have failed. The *uploads* table of the internal Database is used in this case to keep track of the upload process, which is queried by the */status* interface.

The main features of the Ingest Service are the following:

- **Double-layered**
  The code is split into two separated layers for the benefits of portability.

  - Upper layer: also called *Core*, implements the main logic of the Ingest Service, which handles all possible workflows, that can be followed by a Media Package.
  - Lower layer: takes care of local installation dependencies related with data storage and job scheduling. It is separated into two parallel layers:
    * *Scheduler layer*: implements an API for launching and scheduling transcription and translation processes, among others.
    * *Storage layer*: implements an API to allow access to the stored data.

- **Modular design**
  Ingest Service functionalities can be modified/replaced/extended with ease via external modules (i.e. *Mailer* module, *Media Conversion* module, etc.). This allows easy ASR/MT systems integration into the service.

- **Configurable**
  The Ingest Service permits the definition of which file formats will be allowed for each type of file (media, slides, etc.), or which ASR/MT modules will be offering the service, along many other options.

A Media Package file is a non-compressed ZIP file, which contains all media and attachments uploaded to the transLectures Platform. It should contain at least a "manifest.json" file, which declares which files have been included in the package and have to be taken in consideration in the transcription and translation processes, in addition to the lecture's metadata and other information.

Further and detailed documentation of the Media Package file format can be found in Annex C.

2.4. transLectures Player

The transLectures player has been completely redesigned in order to meet the integration requirements of the remote repositories, especially focusing on user's identification. The user interface has been updated into a more modern looking tool and many small changes have been introduced for reducing the user effort on correcting transcription and translation errors to a minimum. The player core has also gone through a major update for taking advantages of the latest HTML5 features.

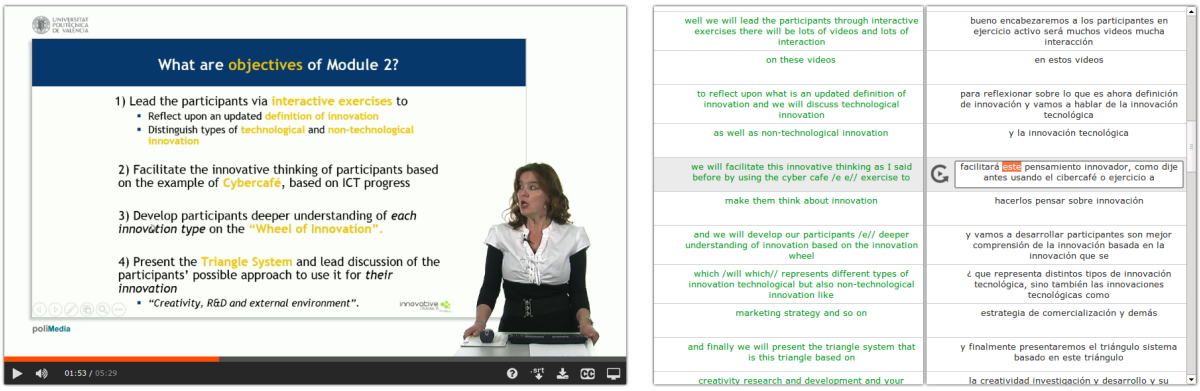

Figure 1: transLectures player for translation editing

More specifically, the main changes of the transLectures player are:

- **Core changes**:
  - Use of new HTML5 *ContentEditable* attribute.
  - Authentication system implemented. Repositories can decide whether or not a specific user is able to perform modifications on current transcriptions and translations.
  - Use of new */lecturedata* transLectures Web Service interface.
  - Loading and saving transcription changes are now asynchronous.
  - Support of Media Fragments URI for starting video editing at a specified time.
  - Full alert system developed to inform users of possible errors.
- **Interface changes**:
  - New player *flat* design.
  - Automatically seeking of video to the beginning of next transcription segment if the time difference is bigger than 0.5 seconds.
  - Improved player focus managing.
  - New *text-only* layout, specially useful for editing translations on small screens.
  - Updated keyboard shortcuts to provide a more natural interaction with users.

3. Integration of the transLectures Platform into VideoLectures.NET and poliMedia

3.1. VideoLectures.NET

Repository Update

All transcriptions and translations in VideoLectures.NET repository were updated to include the new *status* attribute. This new attribute shows the supervision status of the whole DFXP file:

- completely automatic (no supervision)
- mixed (partially supervised)
- completely human (fully supervised)

Web Player

All lectures, which were transcribed and translated via transLectures project, are now available on the translectures.videolectures.net site, which is an identical copy of the original Web site, except for the player. The new player fully supports the transLectures Platform and by communicating with the transLectures Web service it can check and display the transcriptions and translations from the transLectures repository to users of VideoLectures.NET.

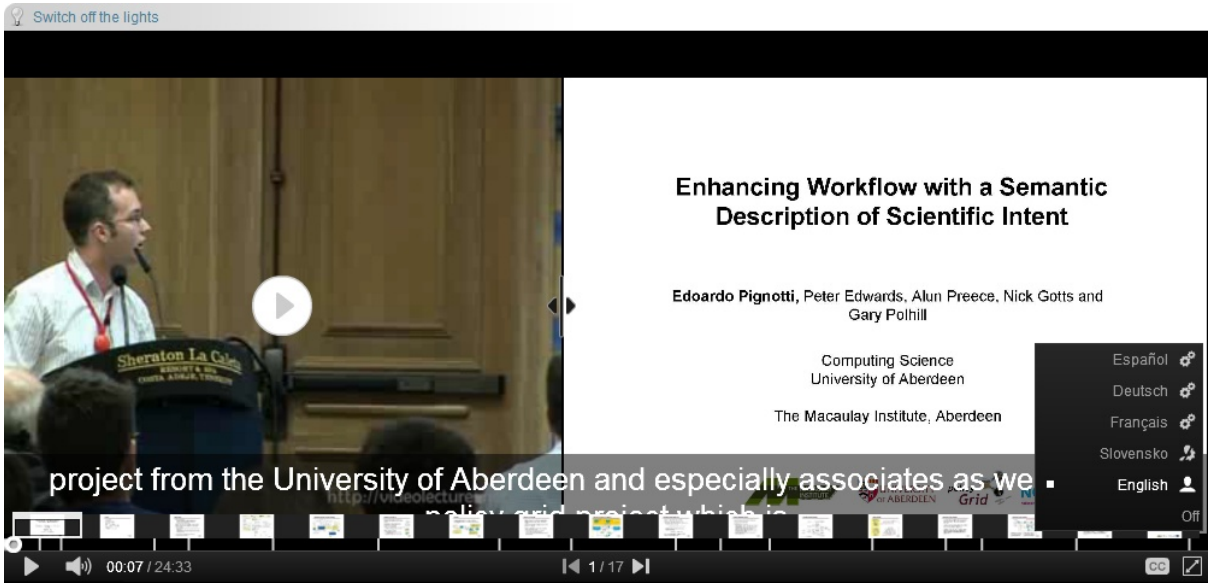

Figure 2: VideoLectures.NET Web player with CC button

Subtitles displayed by the new player are now broken down to several lines if the subtitle is too long to fit on the player's display area.

Following the recommendations by the project reviewers at the end of 2nd year (... *identify/highlight those lectures which have been transcribed and translated (together with an indication of the quality), so that this is immediately recognizable for end users*) the following enhancements have been made to the player:

- **Display of supervision status**:
  According to the supervision status stored in DFXP files player now displays one of the following icons beside the language in the CC pop-up window:

Figure 3: Completely human

Figure 4: Mixed (partially supervised)

Figure 5: Completely automatic

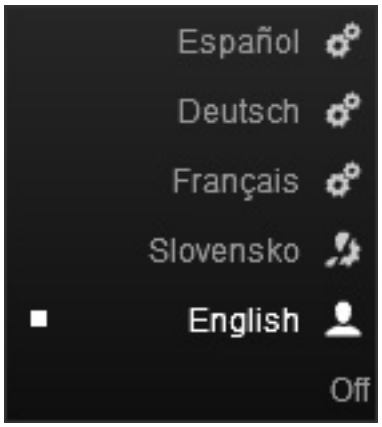

Figure 6: Selection of languages in CC button

- **Highlighted subtitles**:
  Subtitles, for which the *confidence measure* is available, are highlighted in different color according to the *confidence measure* level:
  - less than 0.8: red
  - 0.8 - 0.89: orange
  - 0.9 - 0.99: yellow
  - 1.0 or not present: white

Web Service Update

Local implementation of transLectures Web Service responsible for serving transcriptions and translations to the Web player was updated according to changes described in chapter 2.2 (Web Service):

- implementation of new API interface */lecturedata*
- rewrite of parts of lower layer, dealing with the new *status* attribute
- improved browser cache control

EML Integration

Since EML offers transcription as a (commercial) Web service to outside customers, we implemented an extension to the existing */ingest* functionality of transLectures Web Service supporting the integration with EML's transcription Web service.

Typical process of using the EML's transcription Web service is the following:

- Audio is extracted from the video and remixed to the format required by EML (WAV, 16 Khz, mono)
- Job is given to the EML's Web service with a POST request to their external server, including all metadata and a link to the extracted audio file.
- EML service puts the job in the internal queue.
- When the job finally starts, it connects to the *audio-url* given previously in metadata, downloads the audio file and starts transcribing it.
- This process take some time, during which there is a possibility to check for the status of submitted jobs.
- When EML service finishes the job (or some error occurs), it connects back to the *callback-url* given previously in metadata and POSTs the result in XML format. The result includes some metadata (call id, possible errors) and transcription itself.
- If the transcription was successful, EML's XML format of transcription is translated to our DFXP format with an external tool, written in Python and the new DFXP file is stored in transLectures repository, where it's immediately available through the transLectures Web Service to the users of VideoLectures.NET.

transLectures Player

For the purpose of continuing WP6 evaluations new transLectures player, described in chapter 2.4 (transLectures Player), was installed at VideoLectures.NET. This will enable our evaluators (internal and external) a better experience and minimize the effort needed for the manual supervision.

3.2. poliMedia

Repository Update

All transcriptions and translations in poliMedia repository were updated to include the new *status* attribute. This new attribute shows the supervision status of the whole DFXP file:

- completely automatic (no supervision)
- mixed (partially supervised)
- completely human (fully supervised)

This allows the player to provide indications of the quality of the transcriptions/translations to the users, so they know in advance what kind of quality they may expect.

Web Player

Up to now poliMedia recordings were displayed using a Web Player based on Adobe Flash. In order to be able to view the recordings on mobile devices and also to provide broader compatibility, the poliMedia web site has been redesigned and is now using HTML5 templates and the new Paella Player (paellaplayer.upv.es) as a main application for displaying poliMedia videos.

The use of Paella Player is also a step forward in achieving better Opencast Matterhorn compatibility as the Paella Player is designed to work with Matterhorn.

In order to use the transLectures Web Service with Paella Player we have developed a plugin which is able to retrieve and show the transcriptions from the transLectures local repository (fuster.cc.upv.es).

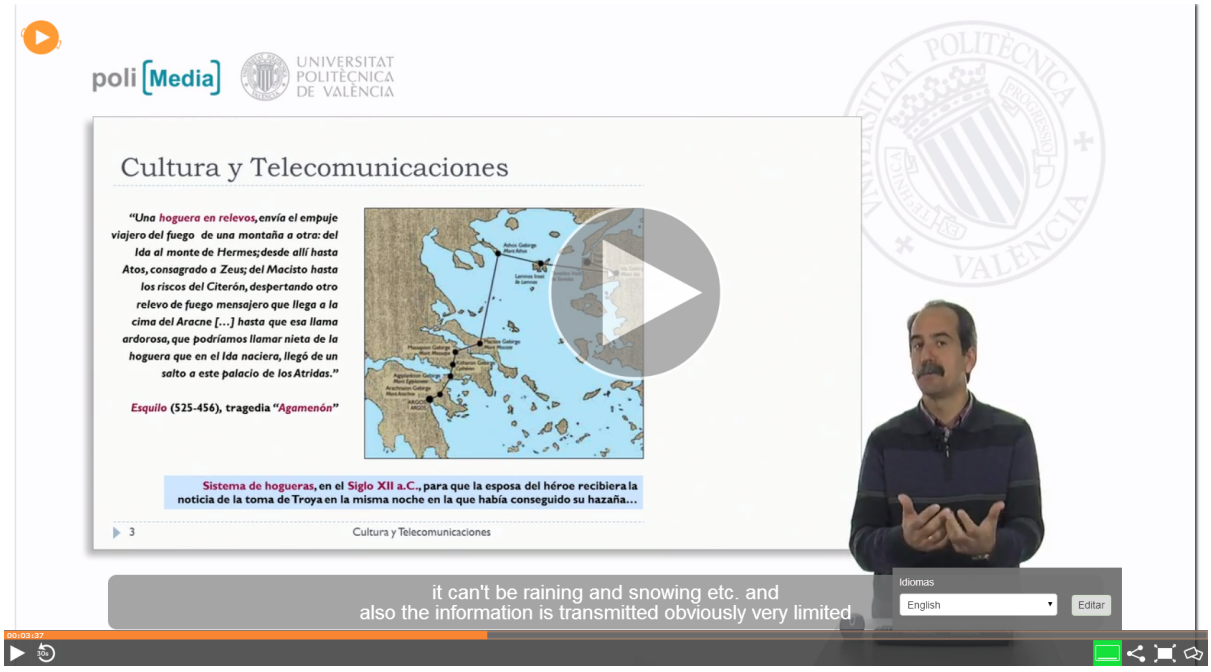

Figure 7: poliMedia's new Web Player (Paella Player) displaying transcriptions

Subtitles displayed by the new player are now broken down to several lines if the subtitles are too long to fit the player's display area.

There is also an "Edit" button that allow the users to jump to the transLectures platform for editing the current transcription/translation. In the current implementation, changes are forwarded to the author of the video in order to decide if they should be incorporated into the stored transcription/translations. This behavior is subject to change upon evaluation.

Ingest service

poliMedia is now using the procedures described in D5.3.1 for maintaining an up-to-date database of transcriptions and translations. The transLectures Web Service APIs */ingest*, */status* and */dfxp* are therefore now fully in use.

4. Relationship with other work packages

Work on WP5 is closely connected with WP2 (Maintenance of current transcriptions and translations), WP4 (Intelligent interaction with users) and WP6 (Evaluation). Active interaction exists between these work packages and WP5 regarding the changes needed for the transLectures Web Service (*/ingest* interface) and the transLectures player (editing interface). The main goal is to provide the best experience for the maintainers of video lectures' portals (VideoLectures.NET and poliMedia), supervisors and for the end users of those portals.

4.1. transLectures Web Service Updates

When the transLectures Web Service changed the underlying format of the DFXP files (to include the supervision status), all partners in WP2 had to follow the same format, when supplying new transcriptions and translations to common repositories.

Also, when WP4 has new requests or proposes changes, this affects both the transLectures Web Service and the transLectures Player and are updated accordingly.

The transLectures Web Service */ingest* and */mod* interfaces are being constantly improved via the ongoing WP6 evaluation and processing of new videos in VideoLectures.NET and poliMedia.

4.2. transLectures Player Development

To offer better experience for the supervising users in WP6 the transLectures Player has been completely redesigned. It now offers:

- side-by-side editing of transcriptions and translations
- user authentication system
- faster loading and saving of transcription changes
- improved error reporting
- better HTML5 compatibility
- support for media fragments URI to start editing at a specified time
- a lot of other usability changes

The transLectures Player also continues to collect detailed usage statistics for the purposes of WP4.

5. Conclusions

In this deliverable we have presented the second report on integration of the models and tools developed in WP3 and WP4 into VideoLectures.NET and poliMedia. The report covers the transLectures integration into the case studies up to M30. Task 5.3 itself will continue until the end of the project (M36).

The main achievement of WP5 in this time has been the complete integration of the transLectures tools into the case studies. Now both VideoLectures.NET and poliMedia are able to fully use the transLectures Platform using the predefined API.

The API itself has been updated several times because of the WP6 evaluation activities and the WP3 and WP4 developments. The transcriptions and translations available to the users are also continuously updated from WP2.

Some of the work has been devoted to the development of external Web interfaces to fully support the transLectures Web Service as it is implemented now and in the future. Both VideoLectures.NET and poliMedia websites have changed their Web Players, one of the main reasons being the ability to use the transLectures Web Service.

References

[1] EC FP7 ICT. First transLectures technical review report (from 01/11/2011 to 31/10/2012). Technical report, EC, 2013.

A. DFXP Format

Objectives

Objective for this proposal is to define a new extension of the DFXP format to reflect the needs of transLectures. To do this we propose to add several new tags for a DFXP transLectures Document.

Types of tags

We need two types of tags:

- One type to reflect the fact that there is an interactive correction from multiple users. So we need a way to "wikify" a DFXP file.
- A second one to reflect the fact that there are different confidence levels for different transcription/translations and even for different parts of a particular transcription/translation.

Namespace

We will add a new namespace to reflect the XML changes; so the new tags will be called <*tl:XXX*> where tl is the new header and XXX is the tag. We'll add a reference to a XSL file for XML validation at the opening <*tt*> tag:

```
<tt xml:lang="en" xmlns="http://www.w3.org/2006/04/ttaf1"
xmlns:tts="http://www.w3.org/2006/10/ttaf1#style" xmlns:tl="translectures.eu">
```

User cases

1. A transcription/translation is automatically generated by an automatic system creating a DFXP file from scratch.
2. A transcription/translation is manually generated by a human expert creating a new DFXP file.
3. A user supervises an automatic transcription/translation.
   (a) The user substitutes a word/sequence of words.
   (b) The user deletes a word/sequence of words.
   (c) The user adds a word/sequence of words.
4. A user supervises an automatic/corrected transcription/translation
5. User splits/joins/erases a segment.

Proposal of new tags

Tags are defined at four levels: document, segment, group and word. Document tags are located at the head section, while segment, group and word tags are located at the body section. An additional tag to relate alternative transcriptions/translations is also included. A detailed explanation of tags follows:

- *<tl:document>* : This tag defines the attributes of the transcription/translation at the top level. As the attributes are inherited, the value of the attributes defined here are the default values, unless otherwise redefined. It contains a specific attribute to associate the current file to a unique video id. Abbreviature: *<tl:d>*
- *<tl:alt>* : The objective of this tag is to maintain a log of modifications for each segment. The scope defined by this tag includes alternative transcriptions/translations for the same audio segment/source sentence. Alternative segments can be also identified as they have the same segment id. Abbreviature: *<tl:a>*
- ** : The aim of this tag is for the Automatic Speech Recognition (ASR) system to define the transcription of an audio segment. Abbreviature: *<tl:s>*
- *<tl:group>* : This tag is thought to define group of words inside a segment. This tag will usually appear as a result of the interaction with the user. Abbreviature: *<tl:g>*
- *<tl:word>* : simple tag used to specify single word properties mostly used for time alignments and confidence measures. Other attributes are mostly inherited. Abbreviature: *<tl:w>*

Next, we define the set of attributes related to the tags just defined. Most of the attributes are applicable to all levels:

- *authorType* : Type of author. Their values are automatic or human. Human for those transcriptions/translations generated by human experts or completely supervised by human experts. Automatic for those transcriptions/translations fully generated by an ASR/MT system. Abbreviature: *aT*
- *authorId* : Author identifier. For example: RWTH, XEROX, UPV, Maria Gialama, etc. Abbreviature: *aI*
- *authorConf* : Confidence measure of the author when the authorType is human. This attribute is coupled with an authorId. This tag could be useful for non-native users supervising a foreign language. Abbreviature: *aC*
- *wordSegId* : It identifies the system that performs the automatic segmentation at word level. It could be different from the authorId, since group of words supervised by the user may be segmented at word level with a different system from that providing the automatic transcription. Abbreviature: *wS*
- *timeStamp* : Instant of creation or modification. The timestamp format is a combination of data and time of day in Chapter 5.4 of ISO 8601. The format is [-]CCYY-MM-DDThh:mm:ss[Z|(+|-)hh:mm]. Abbreviature: *tS*
- *confMeasure* : Confidence measure of the level. These values are generated by ASR and MT systems. Abbreviature: *cM*
- *videoId* : Tag only defined at the document level. It links the current transcription or translation dfxp file to a unique video. Abbreviature: *vI*

- *segmentId* : It is used to uniquely identify a segment in a transcription or translation file. As mentioned above, alternative segments have the same segmentId. Abbreviature: *sI*
- *begin* : Instant of the beginning of an audio portion of the current tag in seconds. Abbreviature: *b*
- *end* : Instant of the end of an audio portion of the current tag in seconds. Abbreviature: *e*
- *elapsedTime* : Processing time. Abbreviature: *eT*
- *modelID* : Model used by decoder. Abbreviature: *mI*
- *processingSteps* : Processing steps of decoder. Abbreviature: *pS*
- *audioLength* : Complete length of the video. Abbreviature: *aL*
- *status* : Supervision status of the subtitles. Abbreviature: *st*. Possible values:
  - fully_automatic → All segments are automatic.
  - partially_human → Some segments have been supervised.
  - fully_human → All segments have been supervised.

Special characters such as & " < > ' must be escaped in the DFXP files according to the XML standard (see http://xml.silmaril.ie/specials.html).

Examples

Document tags

```
<tl:document authorType="human" authorId="John Doe"
timeStamp="2012-10-03T21:32:52" authorConf="1.0"
confMeasure="1.0" videoId="00505-Profesores_Alcoy.M03.B01"
begin="1.0" end="400.6"/>

<tl:document authorType="automatic" authorId="UPV-v1.0"
timeStamp="2012-10-03T21:32:52" authorConf="0.6"
confMeasure="0.75" videoId="00505-Profesores_Alcoy.M03.B01"
begin="0.0" end="400.6"/>
```

Alternative tags

```
<tl:alt>

<tl:segment segmentId="1" authorType="human" authorId="RWTH"
wordSegId="RWTH" timeStamp="2012-10-03T21:32:52" confMeasure="1.0"
begin="0.0" end="15.6">
   ...
</tl:segment>

<tl:segment segmentId="1" authorType="human" authorId="UPV"
wordSegId="UPV" timeStamp="2012-10-03T21:32:52" confMeasure="1.0"
begin="0.0" end="15.6">
   ...
</tl:segment>

</tl:alt>
```

Segment tags

```
<tl:segment segmentId="1" authorType="human" authorId="John Doe"
timeStamp="2012-10-03T21:32:52" confMeasure="1.0"
begin="0.0" end="15.6">
   ...
</tl:segment>

<tl:segment segmentId="1_2" authorType="automatic" authorId="UPV-v1.0"
timeStamp="2012-10-03T21:32:52" confMeasure="0.75"
begin="15.7" end="21.6">
   ...
</tl:segment>
```

Group tags

```
<tl:group  authorType="human" authorId="John Doe"
timeStamp="2012-10-03T21:32:52" confMeasure="1.0"
begin="2.7" end="3.5">
   the way we train in IBM
</tl:group>

<tl:group authorType="automatic" authorId="UPV"
wordSegId="RWTH" timeStamp="2012-10-03T21:32:52"
confMeasure="0.75" begin="33.7" end="45.5">
   greedy algorithms tend to
</tl:group>
```

Word tags

```
<tl:w authorType="manual" authorId="John Smith"
timeStamp="2012-10-03T21:32:52" confMeasure="1.0"
begin="1.3" end="2.1">the</tl:w>

<tl:w authorType="automatic" authorId="EML"
timeStamp="2012-10-03T21:32:52" confMeasure="0.75"
begin="2.3" end="3.5">way</tl:w>

<tl:word confMeasure="1.0" begin="1.3" end="2.1">the</tl:word>

<tl:word confMeasure="0.75" begin="2.3" end="3.5">way</tl:word>
```

Use case examples

A transcription/translation is generated by an automatic system creating a DFXP file from scratch.

```
<?xml version="1.0" encoding="utf-8"?>
<tt xml:lang="en" xmlns="http://www.w3.org/2006/04/ttaf1"
xmlns:tts="http://www.w3.org/2006/10/ttaf1#style" xmlns:tl="translectures.eu">
  <head>
    <tl:document authorType="automatic" authorId="UPV-v1.0" wordSegId="UPV-v1.0"
    timeStamp="2012-10-03T21:32:52" authorConf="0.56" confMeasure="0.75"
    videoId="00505-Profesores_Alcoy.M03.B01"  begin="0.0" end="12.50"/>
  </head>
  <body>
    <tl:segment segmentId="1" confMeasure="0.75" begin="0.0" end="3.2">
      <tl:w confMeasure="0.85" begin="0.0" end="0.75">most</tl:w>
      <tl:w confMeasure="0.89" begin="0.75" end="0.95">of</tl:w>
      <tl:w confMeasure="0.63" begin="0.95" end="1.15">you</tl:w>
      <tl:w confMeasure="0.40" begin="1.15" end="1.35">are</tl:w>
      <tl:w confMeasure="0.90" begin="1.35" end="1.50">probably</tl:w>
      <tl:w confMeasure="0.85" begin="1.50" end="1.75">ventured</tl:w>
      <tl:w confMeasure="0.55" begin="1.75" end="2.00">the </tl:w>
      <tl:w confMeasure="0.98" begin="2.00" end="2.75">problem</tl:w>
      <tl:w confMeasure="0.60" begin="2.75" end="3.20">that</tl:w>
    </tl:segment>
    <tl:segment segmentId="2" confMeasure="0.19" begin="8.5" end="12.50">
      <tl:w confMeasure="0.1" begin="8.5" end="9">To</tl:w>
      <tl:w confMeasure="0.2" begin="9" end="10">solve</tl:w>
      <tl:w confMeasure="0.1" begin="10" end="10.7">on</tl:w>
      <tl:w confMeasure="0.1" begin="10.7" end="12.5">this</tl:w>
    </tl:segment>
  </body>
</tt>
```

A transcription/translation is manually generated by a human expert creating a new DFXP file

```
<?xml version="1.0" encoding="utf-8"?>
<tt xml:lang="en" xmlns="http://www.w3.org/2006/04/ttaf1"
xmlns:tts="http://www.w3.org/2006/10/ttaf1#style" xmlns:tl="translectures.eu">
  <head>
    <tl:document authorType="manual" authorId="Maria" authorConf="1.0"
    videoId="00505-Profesores_Alcoy.M03.B01" timeStamp="2012-10-03T21:32:52"
    confMeasure="1.0" begin="0.0" end="12.50"/>
  </head>
  <body>
    <tl:segment segmentId="1" begin="0.0" end="8.50">
      most of you have probably ventured into the problem set.
    </tl:segment>
    <tl:segment segmentId="2" begin="8.50" end="12.50">
      The solution is:
    </tl:segment>
```

```
  </body>
</tt>
```

A user supervises an automatic transcription/translation substituting a word/group of words

```
<?xml version="1.0" encoding="utf-8"?>
<tt xml:lang="en" xmlns="http://www.w3.org/2006/04/ttaf1"
xmlns:tts="http://www.w3.org/2006/10/ttaf1#style" xmlns:tl="translectures.eu">
  <head>
    <tl:document authorType="automatic" authorId="UPV-v1.0" wordSegId="UPV-v1.0"
    timeStamp="2012-10-03T21:32:52" authorConf="0.56" confMeasure="0.75"
    videoId="00505-Profesores_Alcoy.M03.B01" begin="0.0" end="12.50"/>
  </head>
  <body>
    <tl:alt>
      <tl:segment segmentId="1" confMeasure="0.75" begin="0.0" end="3.20">
        <tl:w confMeasure="0.85" begin="0.0" end="0.75">most</tl:w>
        <tl:w confMeasure="0.89" begin="0.75" end="0.95">of</tl:w>
        <tl:w confMeasure="0.63" begin="0.95" end="1.15">you</tl:w>
        <tl:w confMeasure="0.40" begin="1.15" end="1.35">are</tl:w>
        <tl:w confMeasure="0.90" begin="1.35" end="1.50">probably</tl:w>
        <tl:w confMeasure="0.85" begin="1.50" end="1.75">ventured</tl:w>
        <tl:w confMeasure="0.55" begin="1.75" end="2.00">the</tl:w>
        <tl:w confMeasure="0.98" begin="2.00" end="2.75">problem</tl:w>
        <tl:w confMeasure="0.60" begin="2.75" end="3.20">that</tl:w>
      </tl:segment>
      <tl:segment segmentId="1" confMeasure="0.75" begin="0.0" end="3.20"
      timeStamp="2012-10-04T13:31:45">
        <tl:w confMeasure="0.85" begin="0.0" end="0.75">most</tl:w>
        <tl:w confMeasure="0.89" begin="0.75" end="0.95">of</tl:w>
        <tl:w confMeasure="0.63" begin="0.95" end="1.15">you</tl:w>
        <tl:group authorType="human" authorConf="0.5" confMeasure="1"
        authorId="Jonh" begin="1.15" end="1.35">
          have already ventured
        </tl:group>
        <tl:w confMeasure="0.55" begin="1.75" end="2.00">the</tl:w>
        <tl:w confMeasure="0.98" begin="2.00" end="2.75">problem</tl:w>
        <tl:w confMeasure="0.60" begin="2.75" end="3.20">that</tl:w>
      </tl:segment>
    </tl:alt>
    <tl:segment segmentId="2" confMeasure="0.19" begin="8.5" end="12.50">
      <tl:w confMeasure="0.1" begin="8.5" end="9">To</tl:w>
      <tl:w confMeasure="0.2" begin="9" end="10">solve</tl:w>
      <tl:w confMeasure="0.1" begin="10" end="10.7">on</tl:w>
      <tl:w confMeasure="0.1" begin="10.7" end="12.5">this</tl:w>
    </tl:segment>
  </body>
</tt>
```

A different user (Carlos) from a previous user (John) supervises an automatic transcription/translation substituting a word/group of words

```
<?xml version="1.0" encoding="utf-8"?>
<tt xml:lang="en" xmlns="http://www.w3.org/2006/04/ttaf1"
xmlns:tts="http://www.w3.org/2006/10/ttaf1#style" xmlns:tl="translectures.eu">
  <head>
    <tl:document authorType="automatic" authorId="UPV-v1.0" wordSegId="UPV-v1.0"
    timeStamp="2012-10-03T21:32:52" authorConf="0.56" confMeasure="0.75"
    videoId="00505-Profesores_Alcoy.M03.B01" begin="0.0" end="12.50"/>
  </head>
  <body>
    <tl:alt>
      <tl:segment  segmentId="1" confMeasure="0.75" begin="0.0" end="3.20">
        <tl:w confMeasure="0.85" begin="0.0" end="0.75">most</tl:w>
        <tl:w confMeasure="0.89" begin="0.75" end="0.95">of</tl:w>
        <tl:w confMeasure="0.63" begin="0.95" end="1.15">you</tl:w>
        <tl:w confMeasure="0.40" begin="1.15" end="1.35">are</tl:w>
        <tl:w confMeasure="0.90" begin="1.35" end="1.50">probably</tl:w>
        <tl:w confMeasure="0.85" begin="1.50" end="1.75">ventured</tl:w>
        <tl:w confMeasure="0.55" begin="1.75" end="2.00">the</tl:w>
        <tl:w confMeasure="0.98" begin="2.00" end="2.75">problem</tl:w>
        <tl:w confMeasure="0.60" begin="2.75" end="3.20">that</tl:w>
      </tl:segment>
      <tl:segment  segmentId="1" confMeasure="0.75" begin="0.0" end="3.20"
      timeStamp="2012-10-04T13:31:45">
        <tl:w confMeasure="0.85" begin="0.0" end="0.75">most</tl:w>
        <tl:w confMeasure="0.89" begin="0.75" end="0.95">of</tl:w>
        <tl:w confMeasure="0.63" begin="0.95" end="1.15">you</tl:w>
        <tl:group authorType="human" authorConf="0.5" confMeasure="1"
        authorId="John" begin="1.15" end="1.35">
          have already ventured
        </tl:group>
        <tl:w confMeasure="0.55" begin="1.75" end="2.00">the</tl:w>
        <tl:w confMeasure="0.98" begin="2.00" end="2.75">problem</tl:w>
        <tl:w confMeasure="0.60" begin="2.75" end="3.20">that</tl:w>
      </tl:segment>
      <tl:segment  segmentId="1" confMeasure="0.75" begin="0.0" end="3.20">
        <tl:w confMeasure="0.85" begin="0.0" end="0.75">most</tl:w>
        <tl:w confMeasure="0.89" begin="0.75" end="0.95">of</tl:w>
        <tl:w confMeasure="0.63" begin="0.95" end="1.15">you</tl:w>
        <tl:group authorType="human" authorConf="0.5" confMeasure="1"
        authorId="Jonh" wordSegId="RWTH" begin="1.15" end="1.35">
          <tl:w begin="1.15" end="1.25">have</tl:w>
          <tl:w begin="1.30" end="1.35">already</tl:w>
        </tl:group>
        <tl:group authorType="human" authorConf="0.7" confMeasure="1.0"
        authorId="Carlos" begin="1.35" end="2.70">
          solved similar problems to
        </tl:group>
        <tl:w confMeasure="0.60" begin="2.75" end="3.20"> that</tl:w>
      </tl:segment>
    </tl:alt>
    <tl:segment segmentId="2" confMeasure="0.19" begin="8.5" end="12.50">
```

```
      <tl:w confMeasure="0.1" begin="8.5" end="9">To</tl:w>
      <tl:w confMeasure="0.2" begin="9" end="10">solve</tl:w>
      <tl:w confMeasure="0.1" begin="10" end="10.7">on</tl:w>
      <tl:w confMeasure="0.1" begin="10.7" end="12.5">this</tl:w>
    </tl:segment>
  </body>
</tt>
```

A user adds a new segment

```
<?xml version="1.0" encoding="utf-8"?>
<tt xml:lang="en" xmlns="http://www.w3.org/2006/04/ttaf1"
xmlns:tts="http://www.w3.org/2006/10/ttaf1#style" xmlns:tl="translectures.eu">
  <head>
    <tl:document authorType="automatic" authorId="UPV-v1.0"
    wordSegId="UPV-v1.0" timeStamp="2012-10-03T21:32:52"
    confMeasure="0.75" begin="0.0" end="20.60"/>
  </head>
  <body>
    <tl:segment segmentId="1" confMeasure="0.75" begin="0.0" end="3.20">
      <tl:w confMeasure="0.85" begin="0.0" end="0.75">most </tl:w>
      <tl:w confMeasure="0.89" begin="0.75" end="0.95">of</tl:w>
      <tl:w confMeasure="0.63" begin="0.95" end="1.15">you</tl:w>
      <tl:w confMeasure="0.40" begin="1.15" end="1.35">are</tl:w>
      <tl:w confMeasure="0.90" begin="1.35" end="1.50">probably</tl:w>
      <tl:w confMeasure="0.85" begin="1.50" end="1.75">ventured</tl:w>
      <tl:w confMeasure="0.55" begin="1.75" end="2.00">the</tl:w>
      <tl:w confMeasure="0.98" begin="2.00" end="2.75">problem</tl:w>
      <tl:w confMeasure="0.60" begin="2.75" end="3.20">that</tl:w>
    </tl:segment>
    <tl:segment segmentId="2" authorType="manual" authorId="Maria"
    timeStamp="2012-10-04T13:31:45" authorConf="1" confMeasure="1.0"
    begin="3.20" end="20.60">
      I think the best way to learn acid-base titrations is usually
      you hear a little bit about acid-base titrations, then sit down
      and try to do problems, go back and revisit
    </tl:segment>
  </body>
</tt>
```

A user splits a segment into more than one segment

```
<?xml version="1.0" encoding="utf-8"?>
<tt xml:lang="en" xmlns="http://www.w3.org/2006/04/ttaf1"
xmlns:tts="http://www.w3.org/2006/10/ttaf1#style" xmlns:tl="translectures.eu">
  <head>
    <tl:document authorType="automatic" authorId="UPV" wordSegId="UPV"
    timeStamp="2012-10-03T21:32:52" authorConf="0.2"
    confMeasure="0.75" begin="0.0" end="3.20"/>
  </head>
  <body>
    <tl:segment segmentId="1" confMeasure="0.75" begin="0.0" end="1.75">
      <tl:w confMeasure="0.85" begin="0.0" end="0.75">most</tl:w>
      <tl:w confMeasure="0.89" begin="0.75" end="0.95">of</tl:w>
      <tl:w confMeasure="0.63" begin="0.95" end="1.15">you</tl:w>
      <tl:w confMeasure="0.40" begin="1.15" end="1.35">are</tl:w>
      <tl:w confMeasure="0.90" begin="1.35" end="1.50">probably</tl:w>
      <tl:w confMeasure="0.85" begin="1.50" end="1.75">ventured</tl:w>
    </tl:segment>

    <tl:segment segmentId="2" confMeasure="0.75" begin="1.75" end="3.20">
      <tl:w confMeasure="0.55" begin="1.75" end="2.00">the</tl:w>
      <tl:w confMeasure="0.98" begin="2.00" end="2.75">problem</tl:w>
      <tl:w confMeasure="0.60" begin="2.75" end="3.20">that</tl:w>
    </tl:segment>
  </body>
</tt>
```

B. Web Service documentation

Ingest (*/ingest*)

Description

HTTP GET+POST operation. It is the entry point of the lecture upload service, the so-called Ingest Service, in which uploaded media is automatically transcribed and translated into several languages. Uploaded data (media, slides, documents, etc.) is bundled in a non-compressed zip file called Media Package (please refer to Section C). The Web Service stores that Media Package in the server and returns an upload ID, which can be used afterwards to check the upload progress via the /status interface. This interface should be called automatically by the remote repository once a new lecture is recorded.

This interface also offers Media Package processing & forwarding to an External Service Provider (it depends on the local implementation).

Input

- Required GET Parameters:
  - db = <STRING>: database/repository name (i.e. "vl", "pm").
- Optional GET Parameters:
  - esp = <STRING>: External Service Provider module name.
- Required POST data:
  - *Content-Type = application/zip*
  - Media Package in ZIP format. Please see Section C.
- Call Example:

```
http://my.server.com/path-to-ws/ingest?db=pm

  + POST data: Media Package (ZIP file)
```

Output

- HTTP Codes:
  - HTTP 400: bad request.
  - HTTP 403: forbidden (un-allowed DB access).
  - HTTP 500: internal error.
  - HTTP 200: JSON Object.
- JSON Object specification:

```
{
  'scode' : <INTEGER> ,
  'desc' : <STRING> ,
  'id' : <STRING>
}
```

  - scode → Status code of the WS call.
    - 0 → Upload completed. Returns upload ID.
  - desc → Description of the status code ('scode').
  - id → Upload ID, which can be used afterwards to check the progress of the upload via the /status interface.

- Successful JSON response examples:
  - Media Package successfully uploaded:

```
{
  'scode' : 0,
  'desc' : 'Ingestion complete',
  'id' : 'up-1234'
}
```

Status (*/status*)

Description

HTTP GET operation. Returns information about the progress of an uploaded lecture given an Upload ID. It allows the remote repository to keep track of the automatic uploads and to notice possible processing errors.

This interface also offers forwarding to an External Service Provider (it depends on the local implementation).

Input

- Required GET Parameters:
  - id = <STRING>: Upload ID, returned by the /ingest interface ('id' field).
  - db = <STRING>: database/repository name (i.e. "vl", "pm")
- Optional GET Parameters:
  - esp = <STRING>: External Service Provider module name.
- Call Example:

```
http://my.server.com/path-to-ws/status?id=up-1234&db=pm
```

Output

- HTTP Codes:
  - HTTP 400: bad request.
  - HTTP 403: forbidden (un-allowed DB access).
  - HTTP 500: internal error.
  - HTTP 200: JSON object:
- JSON Object specification:

```
{
  'scode' : <INTEGER> ,
  'desc' : <INTEGER> ,
  'upload_status' : <INTEGER> ,
  'info' : <STRING> ,
  'error_code' : <null || Integer> ,
  'event_code': <null || String> ,
  'up_ts' : <STRING> ,
  'last_check' : <STRING>
}
```

  - 'scode' → Status code of the WS call.
    - 0 → Upload ID exists. Returns upload status information.
    - 1 → Upload ID doesn't exist. All other fields are set to 'null'.
  - 'desc' → Description of the status code ('scode').

  - 'upload_status' → Status code of the upload.
    * 0 → Video ingested, not processed yet.
    * 1 → Processing Media Package file.
    * 2 → Transcription in progress.
    * 3 → Translation(s) in progress.
    * 4 → Preparing data for tL player.
    * 5 → Lecture successfully inserted.
    * 20 → Lecture successfully uploaded.
    * 30 → Lecture successfully deleted.
    * 100 → An unknown error occurred.
    * 101 → An error occurred while processing the Media Package.
    * 102 → An error occurred while transcribing the media.
    * 103 → An error occurred while translating the media.
    * 104 → An error occurred while preparing data for tL player.
    * 105 → An error occurred while inserting lecture into internal database.
    * 200 → An error occurred while updating the given lecture.
    * 300 → An error occurred while deleting the given lecture.
  - 'info' → Detailed information about the status code.
  - 'error_code' → Generic error code that identifies the operation that failed within the process, if any. Otherwise 'null'.
  - 'event_code' → Unique event code that identifies the operation that failed within the process, if any. Otherwise 'null'.
  - 'up_ts' → Upload timestamp.
  - 'last_check' → Last status check timestamp.

- Successful JSON response examples:
  - Provided Upload ID does not exist:

```
{
   "scode": 1,
   "desc" : "Upload ID [ up-1234 ] does not exist.",
   "upload_status": null,
   "info": null,
   "error_code": null
   "event_code": null,
   "up_ts": null,
   "last_check": null,
}
```

  - Transcription in progress:

```
{
   "scode": 0,
   "desc" : "Upload ID exists.",
   "upload_status": 2,
   "info": "Transcription in progress.
            It may take several hours until it finishes.",
   "error_code": null,
   "event_code": null,
   "up_ts": "2014-03-26 19:02:16.174944",
   "last_check": "2014-03-26 19:03:05.298861",
}
```

– Error processing the uploaded media package:

```
{
   "scode": 0,
   "desc" : "Upload ID exists.",
   "upload_status": 101,
   "info": "'external_id' key not found in
            'metadata' in manifest.json file.",
   "error_code": 1016,
   "event_code": "20140326190305298861.1016",
   "up_ts": "2014-03-26 19:02:16.174944",
   "last_check": "2014-03-26 19:03:05.298861",
}
```

– Process finished:

```
{
   "scode": 0,
   "desc" : "Upload ID exists.",
   "upload_status": 5,
   "info": "Lecture successfully inserted into DB with ID '1234-abcd'",
   "error_code": null
   "event_code": null,
   "up_ts": "2014-03-26 19:02:16.174944",
   "last_check": "2014-03-26 19:03:05.298861",
}
```

Lecture Data (*/lecturedata*)

Description

HTTP GET operation. Returns metadata and media file locations of a given lecture ID. This operation is called mainly by the transLectures Player, but it would be called by another external player.

Input

- Required GET Parameters:
  - id = <STRING>: external lecture/video id.
  - db = <STRING>: database/repository name (i.e. "vl", "pm")
- Call Example:

```
http://my.server.com/path-to-ws/lecturedata?id=1234-abcd&db=pm
```

Output

- HTTP Codes:
  - HTTP 400: bad request.
  - HTTP 403: forbidden (un-allowed DB access).
  - HTTP 500: internal error.
  - HTTP 200: JSON Object:
- JSON Object specification:

```
{
  "scode": <INTEGER> ,
  "desc": <STRING> ,
  "lectureinfo": {
                   "language" : <STRING> ,
                   "title" : <STRING> ,
                   "speaker_name" : <STRING> ,
                   "duration" : <INTEGER>
                 } ,
  "media": [
             {
               "is_url" : <INTEGER> ,
               "type_code" : <INTEGER> ,
               "media_format" : <STRING> ,
               "location" : <STRING>
             } ,
             ...
           ]
}
```

  - 'scode' : <INTEGER> → Status code of the WS call.

    * 0 → Lecture ID exists. Returns lecture data information.
    * 1 → Lecture ID doesn't exist. All other fields are set to 'null'.
  - 'desc' : <STRING> → Description of the status code ('scode').
  - 'lectureinfo' <Dict> → Lecture's metadata.
    * "language" : <STRING> → Lecture's language.
    * "title" : <STRING> → Lecture's title.
    * "speaker_name" : <STRING> → Lecturer's full name.
    * "duration" : <INTEGER> → Media duration (In seconds).
  - "media" : <DICT_LIST>
    * "is_url" : <INTEGER> → Defines whether 'location' is an url or a relative physical path.
      · 0 → False
      · 1 → True
    * "type_code" : <INTEGER> → File type code.
      · 0 → Media file.
      · 1 → Slides file.
      · 2 → Related document file.
      · 3 → Video Snapshot/Thumbnail.
      · 4 → Subtitles.
    * "media_format" : <STRING> → Format of the attachment (i.e. "mp4", "ogv", ...)
    * "location" : <STRING> → Relative path or URL in which is located the media file.

- Successful JSON response examples:
  - Lecture does not exist or has no media.

```
{
  "scode" : 1,
  "desc" : "Lecture ID [ 1234-abcd ] does not exist or has no media",
  "lectureinfo" : null,
  "media" : null
}
```

  - Lecture exists.

```
{
  "scode": 0,
  "desc": "Media list and lecture info available",
  "lectureinfo": {
                   "duration": 583,
                   "speaker_name": "Obama, Barack",
                   "title": "Yes we can!"
                 } ,
  "media": [
             { "is_url": 0,
               "type_code": 0,
               "media_format": "mp4",
               "location": "1/1234-abcd/lecture.mp4"
             },
```

```
                { "is_url": 0,
                  "type_code": 0,
                  "media_format": "ogv",
                  "location": "1/1234-abcd/lecture.ogv"
                },
                { "is_url": 0,
                  "type_code": 3,
                  "media_format": "jpg",
                  "location": "1/1234-abcd/lecture_thumbail.jpg"
                }
              ]
}
```

Languages (*/langs*)

Description

HTTP GET operation. Returns list of caption languages available for an specific lecture given a lecture ID. This operation should be called by the transLectures Player and the media player of the client repository.

Input

- Required GET Parameters:
  - id = <STRING>: external lecture/video id.
  - db = <STRING>: database/repository name (i.e. "vl", "pm")
- Call Example:

```
http://my.server.com/path-to-ws/langs?id=1234-abcd&db=pm
```

Output

- HTTP Codes:
  - HTTP 400: bad request.
  - HTTP 403: forbidden (un-allowed DB access).
  - HTTP 500: internal error.
  - HTTP 200: JSON Object.
- JSON Object specification:

```
{
  "scode" : <INTEGER> ,
  "desc" : <STRING> ,
  "media_lang" : <STRING> ,
  "langs" : [
              {
                "code" : <STRING> ,
                "value" : <STRING> ,
                "type" : <INTEGER>
              }
              ...
            ]
}
```

  - "scode" : <INTEGER> → Status code of the WS call.
    - 0 → Lecture ID exists. Returns caption language information.
    - 1 → Lecture ID doesn't exist. All other fields are set to 'null'.
  - "desc" : <STRING> → Description of the status code ('scode').
  - "media_lang" : <STRING> → Language code (ISO-639-1) of the lecture's spoken language.

- "langs" : <DICT_LIST>
  - "code" : <STRING> → Captions' language code (ISO-639-1).
  - "value" : <STRING> → Local language name.
  - "type" : <INTEGER> → Caption type code. Defines the level of human supervion of the captions.
    - 0 → Fully Automatic: whole captions are automatic.
    - 1 → Partially Human: captions are supervised, but not for the whole video.
    - 2 → Fully Human: captions are fully supervised by an human.
    - None → Unknown: type of captions couldn't be determined.

- Successful JSON response examples:
  - Lecture does not exist or has no captions

```
{
  "scode" : 1 ,
  "desc" : "ID 1234-abcd does not exist or has no captions" ,
  "media_lang" : null ,
  "langs" : null
}
```

  - Lecture exists and has captions available

```
{
  "scode" : 0 ,
  "desc": "Language list available" ,
  "media_lang" : "es" ,
  "langs" : [
              { "code" : "es", "type" : 2, "value" : "Español" } ,
              { "code" : "en", "type" : 2, "value" : "English" } ,
              { "code" : "ca", "type" : 2, "value" : "Catalan" }
            ]
 }
```

DFXP (*/dfxp*)

Description

HTTP GET operation. Returns captions in DFXP format for an specific lecture in an specific language, given a lecture ID and a language code. This operation should be called by the transLectures Player and the media player of the client repository.

Input

- Required GET Parameters:
  - id = <STRING>: External lecture/video id.
  - lang = <STRING>: Language code (ISO 639-1).
  - db = <STRING>: Database/repository name (i.e. "vl", "pm")
- Optional GET Parameters
  - altFiltPol = <INTEGER>: Segment selection/filtering policy on <tl:alt>. Possible values:
    - 0 → No filtering (Returns the whole DFXP file).
    - 1 → Return only last modifications.
    - 2 (DEFAULT) → return best alternatives: best automatic transcription (based on confMeasure) if no human, or best human transcription (based on authorConf).
    - 3 → Return original/oldest automatic transcription.
  - segFiltPol = <INTEGER>: Segment text filtering policy. Possible values:
    - 0 → Filtering disabled.
    - 1 (DEFAULT) → Filters out special annotations.
  - form = <INTEGER>: Subtitles format. Possible values:
    - 0 (DEFAULT) → tL-extended DFXP format.
    - 1 → Non-extended DFXP format.
    - 2 → SRT format.
    - 3 → VTT format.
  - intSel = <INTEGER>: Intelligent Selection Mode. Generates a new DFXP file with low-confidence words or phases with context (see parameters "wb" and "wa"). Possible values:
    - 0 (DEFAULT) → Disabled.
    - 1 → Enabled.
  - pw = <INTEGER>: Percentage of words to select when Intelligent Selection Mode is enabled. (DEFAULT=3) (3%)
  - wb = <INTEGER>: Number of words taken into account in Intelligent Selection Mode BEFORE the target word. (DEFAULT=3)
  - wa = <INTEGER>: Number of words taken into account in Intelligent Selection Mode AFTER the target word. (DEFAULT=0)
- Call Example:

```
http://my.server.com/path-to-ws/dfxp?id=1234-abcd&lang=en&db=pm
```

Output

- HTTP Codes:
  - HTTP 400: bad request.
  - HTTP 403: forbidden (un-allowed DB access).
  - HTTP 500: internal error.
  - HTTP 200: OK.
- Response data:
  - 'Content-Type' = 'application/json' if Lecture ID or caption language does not exist.

```
{
  'scode' : <INTEGER> ,
  'desc' : <STRING>
}
```

    * "scode" : <INTEGER> → Status code of the WS call.
      · 1 → Lecture ID doesn't exist or captions in the supplied language does not exist.
    * "desc" : <STRING> → Description of the status code ('scode').
  - 'Content-Type' = 'application/ttml+xml' if DFXP format.
  - 'Content-Type' = 'application/x-subrip' if SRT format.
  - 'Content-Type' = 'text/vtt' if VTT format.

Modify (*/mod*)

Description

HTTP POST operation. Saves the modifications of a caption file made by a user in the transLectures player.

Input

- Required POST data: 'Content-type' = 'application/json' (JSON Object):

```
{
  "db" : <STRING> ,
  "id" : <STRING> ,
  "authorId" : <STRING> ,
  "authorConf" : <FLOAT> ,
  "lang" : <STRING> ,
  "mod": [
          {
            "segId" : <STRING> ,
            "mods": [
                      { "text" : <STRING>, "range" : <STRING> }
                      ...
                    ]
          }
          ...
        ]
 }
```

  - 'db' : <STRING> → Database/repository name (i.e. "vl", "pm")
  - 'id' : <STRING> → External lecture/video id.
  - 'lang' : <STRING> → Captions' language code (ISO 639-1).
  - 'authorId' : <STRING> → Author ID who made the changes.
  - 'authorConf' : <STRING> → Confidence level of the Author.
  - 'mod' : <DICT_LIST> → Array of Dicts which contains a list of Segment IDs to be changed, along with an array of modifications for each segment ID. In batch interaction (default), this array contains a single element.
    - "segId" : <STRING> → Segment ID to be modified.
    - "mods" : <DICT_LIST> → Array of Dicts which contains the modifications made on that specific Segment ID.
      - "range" : <STRING> → Time range of the audio signal that has been modified. <STRING> value has an specific format: "<FLOAT>:<FLOAT>", being the first float the start time, and the second float the end time. For batch interaction (default), time range should be the segment time range.
      - "text" : <STRING> → Amended text for the specified time range.
  - Optional key-value pairs:
    - 'alt_filt_pol':<INTEGER> → Segment selection/filtering policy on <tl:alt> used when the DFXP file was retrieved with a previous /dfxp call. See /dfxp interface documentation above for further details.

    - 'seg_filt_pol':<INTEGER> → Segment text filtering policy used when the DFXP file was retrieved with a previous /dfxp call. See /dfxp interface documentation above for further details.
    - 'intSel':<INTEGER> → Intelligent Selection Mode. See /dfxp interface documentation above for further details.

- Call Example:

```
http://my.server.com/path-to-ws/mod
   + POST data: JSON object
```

  - POST data example:

```
{
  "db":"pm",
  "id":"1234-abcd",
  "authorId":"barack_obama",
  "authorConf":0.97,
  "lang":"en",
  "mod": [
           {
             "segId" : "1" ,
             "mods" : [
                        { "text":"Good morning everybody",
                          "range":"5.87:9.23"
                        }
                      ]
           } ,
           {
             "segId" : "7" ,
             "mods" : [
                        { "text":"Thank you very much",
                          "range":"23.67:25.96"
                        } ,
                        { "text":"Thank you so much",
                          "range":"28.11:30.02"
                        }
                      ]
           }
         ]
}
```

Output

- HTTP Codes:
  - HTTP 400: bad request.
  - HTTP 403: forbidden (unallowed DB access).
  - HTTP 500: internal error.
  - HTTP 200: JSON Object:
- JSON Object specification:

```
{
  'scode' : <INTEGER> ,
  'desc' : <STRING> ,
  'errcode' : <INTEGER>
}
```

  - "scode" : <INTEGER> → Status code of the WS call.
    - 0 → Caption update successful.
    - 1 → Lecture ID doesn't exist or captions for the specified language does not exist.
    - 2 → Caption update failed. 'errcode' contains detailed information.
  - "desc" : <STRING> → Description of the status code ('scode').
  - "errcode" : <INTEGER> → Error code.
    - 1 → Failed to filter Segment (case when no alternates found for an specific segmentID).
    - 2 → Failed to filter Alternate.
    - 3 → Failed to filter Segment after filtering Alternate.
    - 4 → Tried to modify a Group of (already supervised) words (tl:group).
    - 5 → Unexpected XML element.
    - 6 → No segment was updated (i.e. supplied segment ID does not exist in source DFXP file).
    - 7 → Segment has no attributes begin/end.

- Successful JSON response examples:
  - Caption update successful

```
{
  'scode' : 0 ,
  'desc' : 'Update succesful' ,
  'errcode' : null
}
```

  - Lecture does not exist or captions for the specified language does not exist

```
{
  'scode' : 1 ,
  'desc' : "Lecture ID [ 1234-abcd ] does not exist or
            captions for language en not available" ,
  'errcode' : null
}
```

  - Caption update failed

```
{
  'scode' : 2 ,
  'desc' : 'Update failed' ,
  'errcode' : 1
}
```

C. Media Package documentation

Description

A Media Package is a non-compressed ZIP file (a container) used to upload new media and attachments to the transLectures Web Service via the */ingest* interface. It should contain at least a *manifest.json* file.

The Manifest file is a JSON string which declares the uploaded media files and attachments within the Media Package, in addition to the lecture's metadata.

If the implementation of the TransLectures Platform does not maintain a local copy of the repository, in addition to a *media* section it is required to declare a *media_urls* section, in which accessible URLs to several formats (typically mp4 and ogv) for the main media are provided. Otherwise, media files will be stored into the storage host.

Manifest file format

```
{
  "operation_code" : <INTEGER>,
  "media" : {
              "filename" : <STRING> ,
              "fileformat" : <STRING> ,
              "md5" : <STRING>
            } ,
  "attachments" : [
                    {
                      "filename" : <STRING> ,
                      "fileformat" : <STRING> ,
                      "md5" : <STRING> ,
                      "type_code" : <STRING>
                    },
                    ...
                  ] ,
  "metadata" : {
                 "external_id" : <STRING> ,
                 "language" : <STRING> ,
                 "duration" : <INTEGER> ,
                 "title" : <STRING> ,
                 "topic" : <STRING> ,
                 "keywords" : <STRING> ,
                 "date" : <STRING> ,
                 "speaker_id" : <INTEGER> ,
                 "speaker_name" : <STRING> ,
                 "speaker_gender" : <STRING>
               },
  "media_urls" : [
                    {
                      "url" : <STRING> ,
                      "fileformat" : <STRING>
                    } ,
                    ...
```

```
                ]
  "email" : <STRING> ,
  "transLecture" : <INTEGER> ,
  "tL-regenerate" : <INTEGER> ,
}
```

- "operation_code": Defines type of operation: New lecture, upload lecture, delete lecture.
  - operation_code = 0 → New Upload.
    The ingested lecture will be transcribed and translated into several languages, unless the ingest service doesn't feature an ASR system in the spoken language. All uploaded files and the automatically generated captions will be inserted into the internal database.
    You must provide at least a media file at "media" section, a detailed "metadata" section, and optionally some attachments in the "attachments" section.
    Required "metadata" values are:
    * "external_id": Lecture ID in the remote (client) repository.
    * "language": Spoken language of the lecture in ISO 639-1 format (e.g. "en", "es").
    * "title": Title of the lecture. It will be used as query text to retrieve related documents from the web in order to adapt the underlying ASR models to the lecture.
    * "speaker_id": Speaker ID in the remote (client) repository.

    The Ingest Service's behaviour, which depends on the attachments provided, is described as follows:
    * No attachments provided:
      The lecture will be transcribed with an ASR system adapted with external resources retrieved from the internet related on the lecture's title supplied in the "metadata" section. Afterwards it will be translated into the available destination languages.
    * Slides file:
      The lecture will be transcribed with an ASR system adapted with both external resources retrieved from the internet and the text extracted from the slides. Afterwards it will be translated into the available destination languages.
    * Related document(s) file(s):
      The lecture will be transcribed with an ASR system adapted with the attached related document(s), and with the text extracted from the slides if they were supplied. Afterwards it will be translated into the available destination languages.
    * Caption file in the spoken language:
      The lecture will be only translated into the available destination languages using as input the supplied caption.
    * Caption file not in the spoken language:
      The lecture will be first transcribed, and then translated into the available destination languages (except into the provided caption language).
    * Caption file both in the spoken language and other language:
      The lecture will be translated into the remaining available destination languages (if any).

    Automatic transcription and translation(s) of the uploaded lecture can be disabled by setting the "transLecture" option to 0 (please refer to its explanation below).
  - operation_code = 1 → Update lecture
    In this operation mode, the "media" section is optional.

        * If "media" is provided, then it is assumed that the uploaded media is a re-recording of the same lecture ID, and thus, a new transcription and new translations will be generated from that media. Old media will be backed up in any case.
        * If "media" is not provided, the ingest system will or not re-transcribe and/or re-translate and/or update the existing lecture's metadata and/or files depending on the provided attachments:
            · No attachments provided:
            Metadata update only.
            · Slides file and/or External Resource file(s):
            Re-transcription and re-translation of the existing lecture.
            · Caption file in the spoken language:
            Re-translation of the existing lecture from the supplied caption into the available destination languages.
            · Caption file not in the spoken language:
            Insert/Update of the corresponding caption file.
            · Caption file both in the spoken language and other language:
            Re-translation of the existing lecture into the remaining available destination languages (if any).
        * Special case: if "tL-regenerate" option is set to 1, lecture will be re-transcribed and re-translated.
    - operation_code = 2 → Delete lecture
      Must provide only its "external_id" in the "metadata" section. Lecture will be marked as "deleted" into the database and files will be backed up.

- "media" : <DICT>
    - "filename" : <STRING> → File name of the main media file.
    - "fileformat" : <STRING> → Format of the main media file. (See "Allowed attachments" below)
    - "md5" : <STRING> → MD5 checksum of the main media file.
      *** Note: main media file is assumed to have type_code = 0.

- "metadata" : <DICT>
    - "external_id" : <STRING> → External Lecture ID.
    - "title" : <STRING> → Lecture's title.
    - "language" : <STRING> → Lecture language code.
    - "speaker_id" : <INTEGER> → Lecturer's ID.
    - "speaker_name" : <STRING> → Lecturer's full name.
    - "speaker_gender" : <STRING> → Lecturer's gender.
    - "duration" : <INTEGER> → Media duration (In seconds).
    - "topic" : <STRING> → Lecture's topic.
    - "keywords" : <STRING> → Lecture's keywords.
    - "date" : <STRING> → Lecture's publication date.

- "attachments" : <DICT_LIST> (OPTIONAL)
    - "filename" : <STRING> → File name of the attachment.
    - "fileformat" : <STRING> → Format of the attachment. (See "Allowed attachments" below)

- "md5" : <STRING> → MD5 checksum of the attachment.
- "type_code" : <INTEGER> → Attachment type code. (See "Allowed attachments" below)
  - 0 → Media file.
  - 1 → Slides file.
  - 2 → Related document file.
  - 3 → Video Snapshot/Thumbnail.
  - 4 → Subtitles.

- "Media_urls" : <DICT_LIST> (OPTIONAL)
  - "url" : <STRING> → URL location of the media.
  - "fileformat" : <STRING> → Format of the attachment. (See "Allowed attachments" below)

    *** Note: media urls are assumed to have type_code = 0.

- "email" : <STRING> (OPTIONAL) → E-mail to send notifications about process failures and successes.
- "transLecture" : <INTEGER> (OPTIONAL)
  - 0 → Disable automatic transcription and translation of the uploaded lecture. (Will only insert lecture into DB / update metadata)
  - 1 (DEF) → Enable automatic transcription and translation of the uploaded lecture.
- "tL-regenerate" : <INTEGER> (OPTIONAL)
  - 0 (DEF) → Do not force re-generation of transcription and translations for the given lecture.
  - 1 → Force re-generation of transcription and translations for the given lecture.

    system upgrades. Please take this into consideration. Operation code must be set to 1 (Update). Only 'external_id' and 'language' keys from 'metadata' section are required.

Example of Manifest file:

```
{
  "operation_code" : 0,
  "media" : { "filename":"lecture.mp4" ,
              "fileformat":"mp4" ,
              "md5" : "44c6002dda76c2dcfe565c8881438a76"
            },
  "attachments" :
      [
        { "filename":"slides_in_video_format.wmv",
          "fileformat":"wmv",
          "type_code":1,
          "md5":"c8722d0e8e27d4b5caaa7122a14676e3"
        },
        { "filename":"highly_related_document.pdf",
          "fileformat":"pdf",
          "type_code":2,
          "md5":"3b1c0cd17db1ae840d7a8db0b7bb1e4f"
```

```
        },
        { "filename":"nice_thumbnail.jpg",
          "fileformat":"jpg",
          "type_code":3,
          "md5":"7b3f5ab34053b156db7c7fff1f97b844"
        }
      ],
  "metadata" :
     {
       "external_id":"1234-abcd",
       "language":"en",
       "duration":583,
       "speaker_id":1,
       "speaker_name":"Obama, Barack" ,
       "title":"NSA: Why do we spy people? An approach to true national security.",
     }
}
```

Allowed attachments

The list of allowed attachments depends on the Ingest Service configuration, but by default we'll admit the upload of the folowing file formats (fileformat) depending on the file type (type_code):

- type_code = 0 (media)
  - Video files [1]: mp4, m4v, ogv, wmv, avi, mpg, flv
  - Audio files[2]: wav, mp3, oga, flac, aac
- type_code = 1 (slides)
  - Text slides [3]: txt, ppt, pptx, pdf.
  - Video slides [4]: mp4, m4v, ogv, wmv, avi, mpg, flv.
- type_code = 2 (documents)
  - Text documents [5]: txt, doc, docx, ppt, pptx, pdf.
- type_code = 3 (thumbnails)
  - Image files: jpg.
- type_code = 4 (captions)
  - Caption files: dfxp, srt, trs.

[1] the transLectures player requires mp4 and ogv versions of the uploaded media in order to maximize compatibility with browsers. Then, the uploaded media will be converted in a final step into mp4 and ogv, if necessary.

[2] In the case of audio files, mp4 and ogv videos will be generated as well from the audio signal.

[3] Sorted by descending text extraction quality

[4] For better quality, a video slides file must be a video in which appears the slides and nothing else. Text from the slides is obtained using an OCR system.

[5] Sorted by descending text extraction quality