Deliverable D 3.4.3

Documentation of Infrastructure

Author(s): Juli Bakagianni (ILSP)
Aljoscha Burchardt (DFKI)
Arle Lommel (DFKI)
Stelios Piperidis (ILSP)
Kashif Shah (USFD)
Lucia Specia (USFD)

Dissemination Level: Public
Date: 30.June 2014 (original)
05.December 2014 (revised)

Grant agreement no.	296347
Project acronym	QTLaunchPad
Project full title	Preparation and Launch of a Large-scale Action for Quality Translation Technology
Funding scheme	Coordination and Support Action
Coordinator	Prof. Hans Uszkoreit (DFKI)
Start date, duration	1 July 2012, 24 months
Distribution	Public
Contractual date of delivery	June 2014
Actual date of delivery	June 2014
Deliverable number	3.4.3
Deliverable title	Documentation of Infrastructure
Type	Report
Status and version	Version 2.0 (December 2014), see change log
Number of pages	
Contributing partners	ILSP, USFD
WP leader	DFKI
Task leader	DFKI
Authors	Juli Bakagianni, Aljoscha Burchardt, Arle Lommel, Stelios Piperidis, Kashif Shah, Lucia Specia
EC project officer	Aleksandra Wesolowska
The partners in QTLaunchPad are:	Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI), Germany
	Dublin City University (DCU), Ireland
	Institute for Language and Speech Processing, R.C. “Athena” (ILSP/ATHENA RC), Greece
	The University of Sheffield (USFD), United Kingdom

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

DFKI GmbH
QTLaunchPad
Dr. Aljoscha Burchardt
Alt-Moabit 91c
10559 Berlin, Germany

aljoscha.burchardt@dfki.de
Phone: +49 (30) 23895-1838
Fax: +49 (30) 23895-1810

Copies of reports and other material can also be accessed via http://www.qt21.eu/launchpad

Table of Contents

1 Executive Summary

This Deliverable provides user documentation for the Infrastructure developed in the QTLaunchPad project. It provides guidelines and instructions for using these resources. More details about these resources may be found in D3.3.1 (Adapted tools for the QTLaunchPad infrastructure), D3.5.2 (User report about the system), and D3.4.2 (MT-speific infrastructure). In particular, it contains documentation for translate5 (section 2), the QT21 META-SHARE Repository (section 3), QuEST (section 4), and the Scorecard and Metric Builder (section 5).

Because of the nature of the respective infrastructure components, the documentation for translate5, the QT21 META-SHARE Repository, and the Scorecard and Metric Builder is focused on high-level user tasks, while the documentation for QuEst is focused much more on configuration and set-up and requires more technical knowledge than do the other components.

Additional information about translate5 and the MQM scorecard and how they compare is available in the QTLaunchPad supplemental report *Practical Guidelines for the Use of MQM in Scientific Research on Translation Quality*. This report is available from the following link: http://qt21.eu/downloads/MQM-usage-guidelines.pdf

2 translate5

This section provides a guide for using the translate5 infrastructure. The translate5 system incorporates many of the features described in D3.4.1 and provides an easy-to-use interface for annotating data with MQM issues, as well as for post-editing, ranking, and other common translation research tasks.

UPDATE: The content of this section has been moved to the following URL:

http://www.qt21.eu/launchpad/node/1337

Instructions for installing translate5 may be found at the following URL:

http://www.qt21.eu/launchpad/node/1344

2.1 License

translate5 is available under the GLP 3.0 license, with the exception that the Ext JS framework used for UI components is only available for non-commercial development unless a separate license is obtained. Details of the license exceptions are available online.[1] Note that *all* components funded by the QTLaunchPad project are available under the terms of the GPL 3.0 license.

[1] https://bitbucket.org/mittagqi/translate5_released/src/1d032dda6a703b6c1c10574843ddc88fb688c17a/floss-exception.txt?at=master

2.2 translate5 API Documentation

This section contains a description of the translate5 API

General REST API

translate5 provides a REST API to create, read and manipulate all stored data. REST is a HTTP based, resource-oriented architecture.

For further information see: http://en.wikipedia.org/wiki/Representational_state_transfer

The data format is normally JSON. On PUT requests not all data fields must be given, only the fields with changed data are sufficient.

To see the use of the REST API in action, a developer may have a look on the way translate5 JS-GUI frontend uses the REST API through tools like Firebug. This way it is easy for a programmer, to understand the way it works and to see, which values are used for which resources in which situations.

REST Resource Description

Task

Provides access to the tasks stored in translate5.

URL:	/editor/task/[ID]
Available Methods:	GET / POST / PUT / DELETE
Specialities:	Since on a POST request the import data is given, the POST must be transmitted as “multipart/form-data”. By setting the userState to the value “edit” the task is registered as to be edited by the current user. Setting it back to the userState “open” means that the the task is open again for all users.

Special Task Export URLs

URL to get the edited task data in a ZIP container:	/editor/task/export/id/[ID]/
As above, but with diff tags:	/editor/task/export/id/[ID]/diff/1
MQM Statistic Export:	/editor/qmstatistics/index/taskGuid/[GUID]/?type=[TYPE]

Where [ID] is the value of the mysql-id field, [GUID] the mysql-taskGuid and [TYPE] a valid field type of the task (field type refers to the column the MQM has been used on in the editor).

Task Resource Layout

Name	Type	Info
id	int	GET only
taskGuid	string	GET only
entityVersion	integer	GET only, set by server
taskNr	string	
taskName	string	
sourceLang	integer	
targetLang	integer	
relaisLang	integer	
locked	date	GET only
lockingUser	string	GET only
lockingUsername	string	GET only
state	string	
workflow	string	
pmGuid	string	
pmName	string	
wordCount	integer	
targetDeliveryDate	date	
realDeliveryDate	date	
referenceFiles	boolean	GET only
terminologie	boolean	GET only
orderdate	date	
edit100PercentMatch	boolean	POST only
enableSourceEditing	boolean	POST only
qmSubEnabled	boolean	GET only
qmSubFlags	auto	GET only
qmSubSeverities	auto	GET only
userState	string	As defined by the workflow.
userRole	string	GET only
userStep	string	GET only
users	auto	GET only
userCount	integer	GET only
defaultSegmentLayout	boolean	GET only
importUpload	HTTP Upload	POST only

Task User Associations

Provides the associations between users and tasks.

URL:	/editor/taskuserassoc/[ID]
Available Methods:	GET / POST / PUT / DELETE
Specialities:	

Resource Layout

Name	**Type**	**Info**
id	int	GET only
entityVersion	int	GET only, set by server
taskGuid	string	
userGuid	string	
login	string	GET only, set by server
surName	string	GET only, set by server
firstName	string	GET only, set by server
state	string	
role	string	

Segment

Provides access to the data of a single segment.

URL:	/editor/segment/[ID]
Available Methods:	GET / PUT
Specialities:	Only possible with a task in edit mode. Resource Layout is dynamic.

Resource Layout

Name	**Type**	**Info**
id	int	GET only
fileId	int	GET only, set by import
segmentNrInTask	int	GET only, set by import
userName	string	GET only, set by server
timestamp	date	GET only, set by server
editable	boolean	GET only, set by import
autoStateId	int	GET only, set by server

Name	Type	Info
workflowStep	string	GET only, set by server
matchRate	int	GET only, set by import
durations	object	
comments	string	GET only, set by server
qmId	string	
stateId	int	
[FIELD]	string	GET only
[FIELD]Edit	string	Optional, only if field is editable.

Segments contains dynamic fields, defined on import. This dynamic fields contain the segment payload.

Segment Fields

Provides the segment fields of one task.

URL:	/editor/segmentfield/[ID]
Available Methods:	GET
Specialities:	Only possible with task in edit mode. Returns the available segment fields in this task.

Resource Layout

Name	Type	Info
id	int	
taskGuid	string	
name	string	
label	string	
type	string	
rankable	boolean	
editable	boolean	

Comment

Provides access to comments of segments.

URL:	/editor/comment/[ID]
Available Methods:	GET / POST / PUT / DELETE
Specialities:	Only possible with a task in edit mode and filtered by GET Parameter segmentId.

Resource Layout

Name	Type	Info
id	int	
segmentId	int	
userName	string	GET only, set by server
comment	string	
isEditable	boolean	GET only, set by server
modified	date	GET only, set by server
created	date	GET only, set by server

File

Provides the files of a task, and the possibility to reorder the files in the filetree.

URL:	/editor/file/[ID]
Available Methods:	GET / PUT
Specialities:	Only possible with a task in edit mode. GET returns a JSON tree of file metadata

Resource Layout of a single file node

Name	Type	Info
id	int	GET only
filename	string	GET only
path	string	GET only
cls	string	GET only
children	array	GET only
parentId	int	GET / PUT
index	int	PUT only

MQM Statistics

Provides access to the MQM Statistics.

URL:	/editor/qmstatistics/index/taskGuid/[GUID]/?type=[TYPE] [GUID] is the taskGuid of the task to get the statistics from. [TYPE] is the fieldname of the datafield to get the statistics from
Available Methods:	GET (list only)
Specialities:	returns a XML file

Reference File

Provides the reference files of a task.

URL:	/editor/referencefile/[FILEPATH] [FILEPATH] is the whole path to the file, as given on import.
Available Methods:	GET
Specialities:	Only possible with a task in edit mode. GET without a [FILEPATH] returns a JSON tree of reference files in the same format as described under “Files”. GET with a [FILEPATH] provides the desired file as a download.

User

Provides access to the users available in the application.

URL:	/editor/user/[ID]
Available Methods:	GET / POST / PUT / DELETE
Specialities:	If an empty password field is given on PUT, password will be reset and user is requested by email to set a new password.

Resource Layout

Name	Type	Info
id	int	GET only
userGuid	string	GET only, set by server
firstName	string	
surName	string	
gender	string	
login	string	
email	string	
roles	string	
passwd	string	
editable	boolean	GET only, set by server
locale	string	GET only, set by server

Workflow User Preferences

Provides access to the workflow based preferences a user can have for a task.

URL:	/editor/workflowuserpref/[ID]
Available Methods:	GET / POST / PUT / DELETE
Specialities:	

Resource Layout

Name	Type	Info
id	int	GET only
entityVersion	int	GET only, set by server
taskGuid	string	
workflow	string	A valid workflow name
workflowStep	string	A valid workflow step to the above defined workflow
anonymousCols	boolean	
visibility	string	
userGuid	string	
fields	string	

Filtering

All GET calls without an ID provides a list of resources. With the GET Parameter “filter” this list can be filtered. In general all fields are filterable. Exceptions in single cases.

Authentication

A dynamic authentication by API is currently not implemented. API calls by a non-browser-based software must have the roles and rights of a hardcoded dedicated user. Access restriction is done by network access restriction. For all requests the authenticated user must have the right to access the resource in the requested context.

3 QT21 META-SHARE Repository

3.1 Overview

This section is a guide for the users of the prototype of the QT21 repository. The repository constitutes an extension of the META-SHARE infrastructure, providing an additional language processing mechanism for processing language datasets (essentially language corpora) with appropriate natural language processing tools. Thus, this document describes the language processing functionality and provides the users with step-by-step explanations of how to exploit it.

3.2 Introduction

The QT21 repository (http://qt21.metashare.ilsp.gr/) is a repository of language resources (LRs), including both language data and language processing tools and services, described through a set of metadata, allowing for uniform search and access to resources. LRs can be both open and with restricted access rights, either for free or for-a-fee. More specifically, the QT21 repository offers to the user the possibility to:

- have access as a registered user
- search and browse the catalogue

- view details about a LR
- view general statistics
- download a LR dataset
- process a LR with appropriate services
- access the community forum

3.3 Getting Started

In the QT21 repository the user can process language resources (LRs) with language services, where both are described in the repository and thus all the functionalities that are offered from the infrastructure are applicable to them. The output of the processing procedure is stored and indexed in the repository.

3.3.1 Logging In

The processing service is provided only to the logged in users. If the user is not registered yet, she can register following the steps provided in section 3.1.1. If the user has already an account, she can log in to the QT21 repository following the steps described in section 3.1.2

3.3.1.1 Register as a new user

In order to register to the QT21 repository and get an account:

1. Click the “Register” button at the top right of the QT21 repository home page.
2. Fill in the “Create Account” form with an Account Name of your choice, your First Name, Last Name, your Email address and the password of your choice.
3. Click the “Create Account” button.
4. The following message appears in the QT21 repository home page: “We have received your registration data and sent you an email with further activation instructions.”
5. The system generates a message asking for a confirmation of the registration and sends it to the address provided when registering.Click on the link provided in the message to confirm the registration (this has also the effect of logging in the user for the first time).
6. A confirmation message appears in the QT21 repository home page as follows: “We have activated your user account.”

3.3.1.2 Log In

Registered users can use their credentials to log in to the QT21 repository:

1. Click the “Login” button at the top right of the home page.
2. Fill in the “User Authentication” form with Username and Password.
3. Click the “Login” button.

3.3.2 Repository services

The language resources sharing services (provision and consumption oriented services) of the QT21 repository, are described in detail at https://github.com/metashare/META-

SHARE/tree/master/misc/docs. In what follows, we focus on the new processing layer of the QT21 repository.

3.3.3 Input Specifications

This section defines the specifications of the input for the processing mechanism. The input is:

- a text corpus,
- bilingual or monolingual, which
- represents one (for the monolingual LRs) or two (for the bilingual LRs) of the following languages:
    - German,
    - Greek, Modern (1453)
    - English
    - Portuguese
- has either raw data or is consistent with one of the following annotation levels:
    - segmentation in token segments
    - segmentation in sentence segments
    - morphosyntacticAnnotation, posTagging
    - lemmatization
    - morphosyntacticAnnotation, bPosTagging
- has data that are consistent with one of the following formats:
    - for monolingual text corpora:
        - text,
        - XCES,
        - XML
    - for bilingual text corpora:
        - TMX,
        - MOSES,
        - XCES
- is distributed with public licence that accepts derivatives, and
- is downloadable from the repository storage.

On the provider's side, the steps to follow to accurately document a dataset intended to be processable, are described in Appendix A.

3.3.4 Processing a Language Resource

The user can process the LRs of the QT21 repository that fill the above specifications. Thus, the user can either use the faceted browse (see Figure 1) or the simple search to detect the processable resources.

The user can click on the name of a LR from the results page obtained by any type of search to open the page with the details for that LR (see Figure 2). If the resource is processable, the button “Process” gets activated, and by a click from the user the processing mechanism starts.

Figure 1. Search Language Resources

Figure 2. View Language Resources

The processing mechanism needs to be configured to the needs of the user. It begins with a form that requests from the user to select the annotation level of the to-be-annotated LR. Depending on the input specifications, the available annotation levels can vary from input resource to input resource. The user can select the annotation level of her choice and submit the form (see Figure 3).

Following the submission of the annotation levels form, the user is directed to the services form. For the selected input LR and the annotation level of the to-be-annotated LR, all the available services are presented in the services form. The user can choose the service of

her preference and submit the form (see Figure 4). If the input LR is bilingual, then distinct lists of services will be available for each language side of the data (see Figure 5).

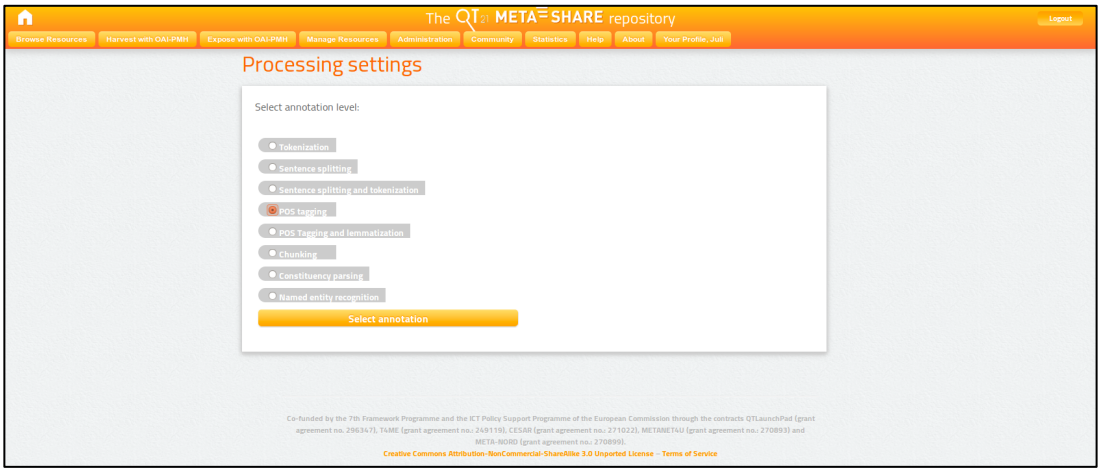

Figure 3. Annotation levels form

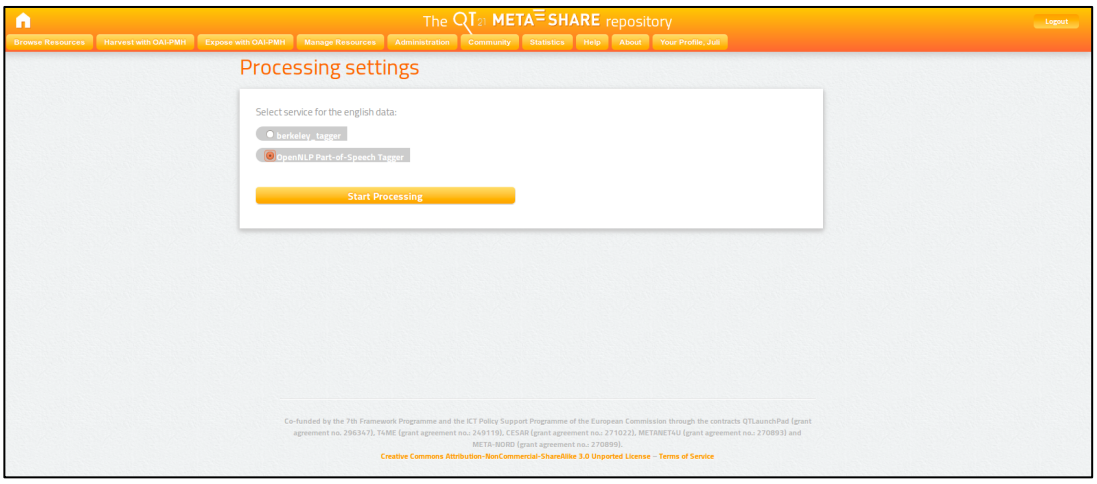

Figure 4. Language services form for monolingual input data

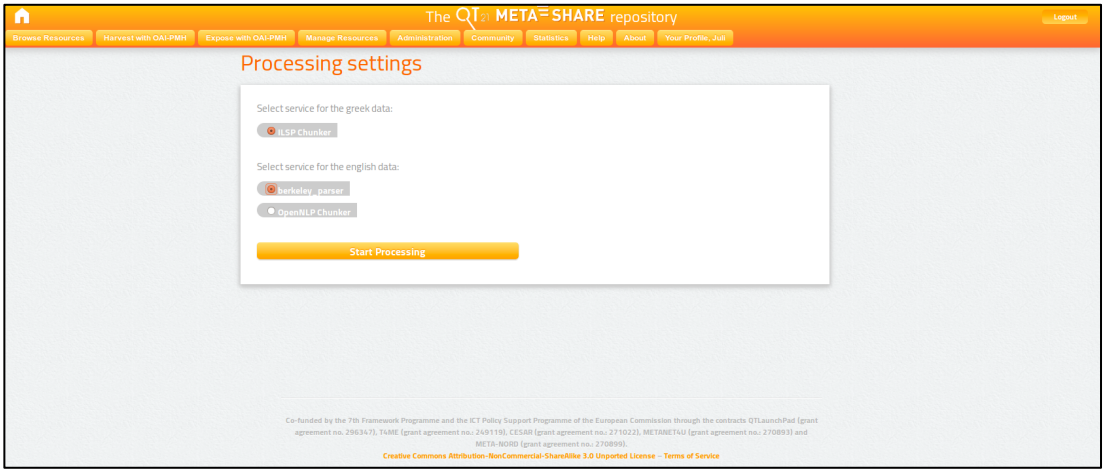

Figure 5. Language services form for bilingual input data

With the submission of the services form, the processing settings are configured and the input LR will be dispatched to the selected language services for processing. When the processing finishes, the user will be informed via email. The email will contain the link of the new annotated LR on the QT21 repository (see Figure 6) and the user can download the annotated data from the platform (see Figure 7).

If the user requests to process a LR with a specific web service, the result of which has already been provided for another user, then the user will be just forwarded to the annotated LR that has been created in the repository (see Figure 8).

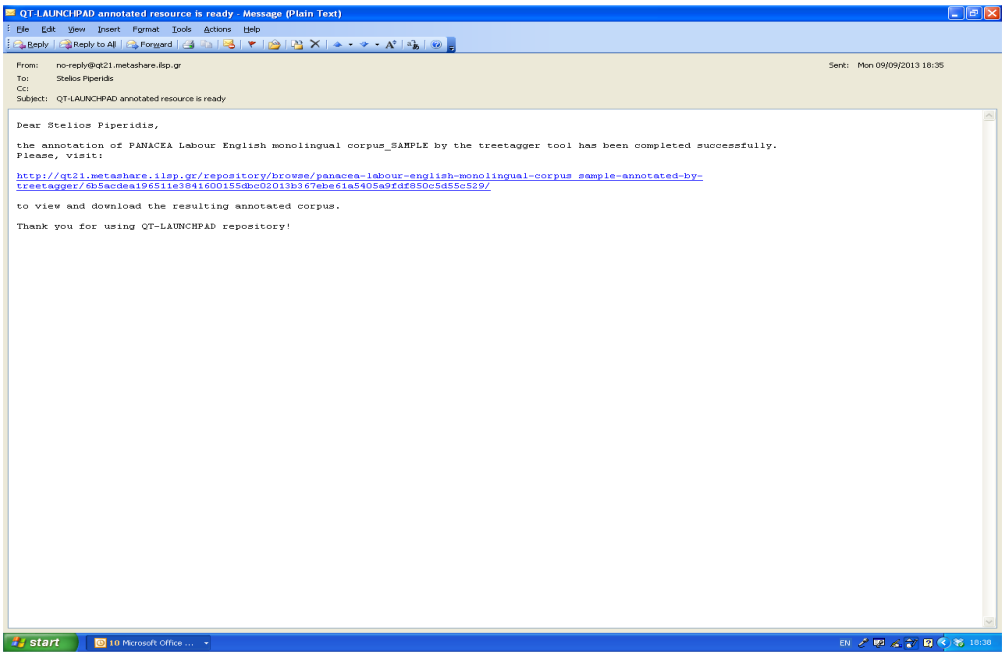

Figure 6. Email sent to the user

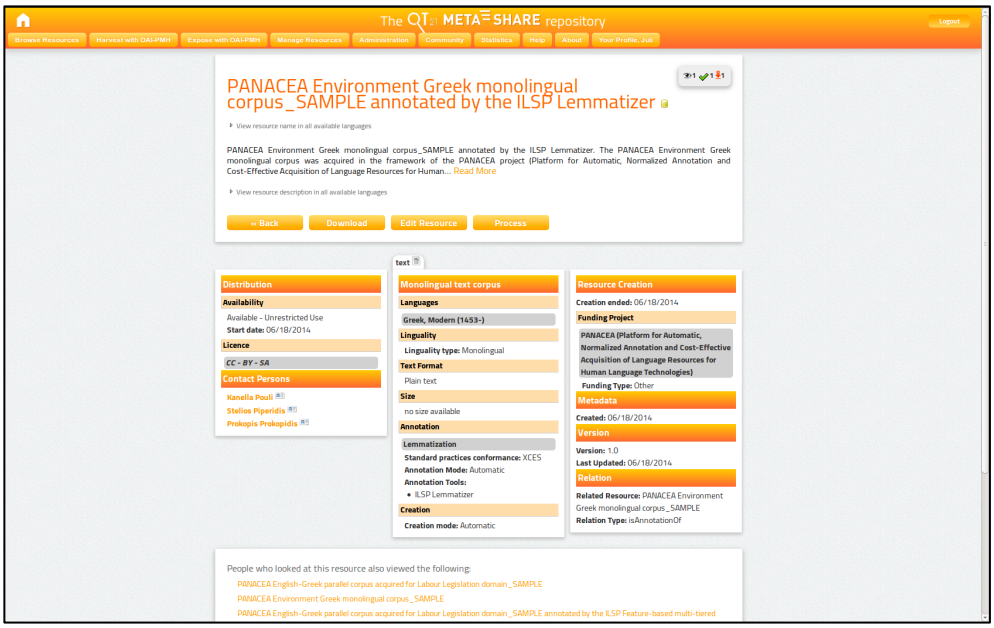

Figure 7. Output LR

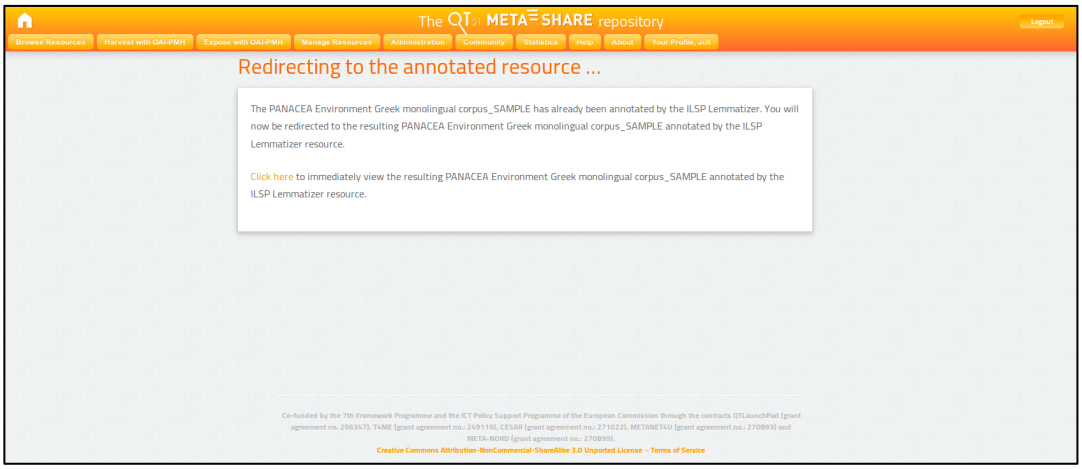

Figure 8. Forwarded output LR

3.4 Appendix A

This Appendix concerns the providers of Language Resources who wish to deposit their LRs to the QT21 repository and make them processable. A LR can be input to the processing mechanism offered by the QT21 repository, if it complies with the specifications described in Section 3.3. The provider can deposit her LR to the QT21 repository, by referring to the documentation in the provider's manual of the META-SHARE application (https://github.com/metashare/META-SHARE/tree/master/misc/docs). Additionally, the provider can upload the actual data of her LR to the QT21 repository storage (see Figure 9). The data must be compressed and must not exceed 20MB in size. The provider must additionally select the data format of her LR. Information regarding every data format is available by clicking the help buttons, which accompany each data format choice (see Figure 10).

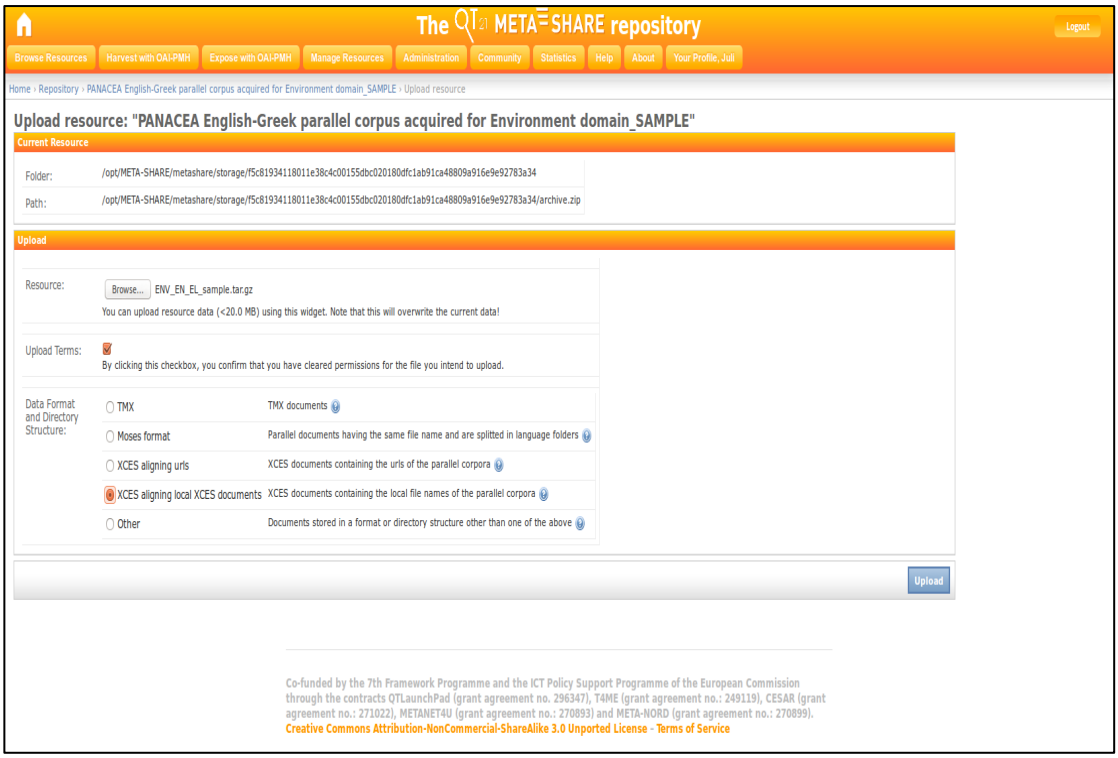

Figure 9. Upload data to the QT21 repository

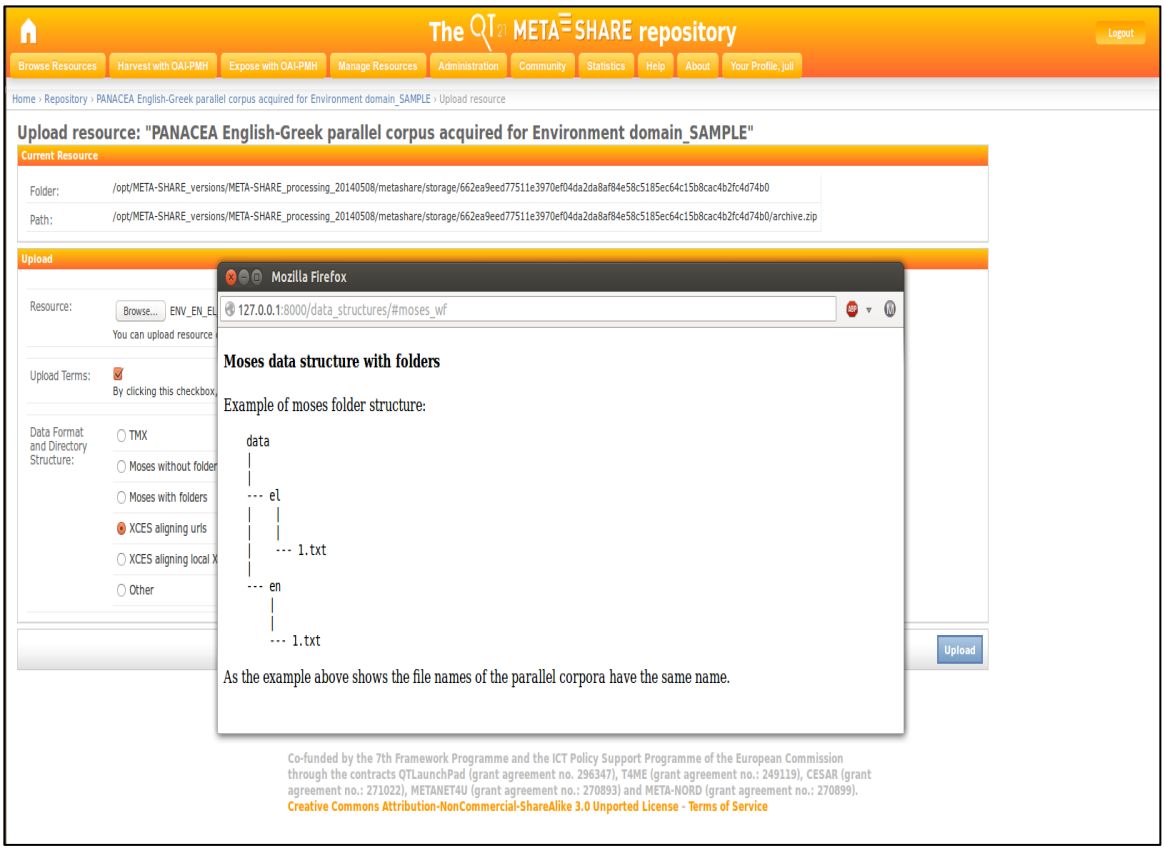

Figure 10. Information about data formats

4 QuEst

The QuEst open source software is aimed at quality estimation (QE) for machine translation. It was developed by Lucia Specia's team at the University of Sheffield. This particular release (October 31, 2013) was made possible through the QuEst project and QTLaunchPad projects (http://www.quest.dcs.shef.ac.uk/). The code has two main parts: a feature extractor and a machine learning pipeline.

4.1 Feature extractor

This code implements a number of feature extractors, including most commonly used features in the literature, as well as many of the features used by systems submitted to the WMT2012 shared task on QE. Extractors for new features can be easily added (see the documentation under dist/).

4.1.1 Installation

The program itself does not require any installation step. It requires the Java Runtime Environment, and depending on the features to be extracted, a few additional libraries (see below). If you change the code, it can be easily rebuilt using NetBeans, as a NetBeans project is distributed in this release.

4.1.2 Dependencies

The libraries required to compile and run the code are included in the "lib" directory in the root directory of the distribution. The Java libraries should be included there when possible. Here is a list of libraries that should be downloaded and placed in the "lib" directory:

- Stanford POS Tagger
- Berkeley Parser

Apart from these lib files, QuEst requires other external tools / scripts to extract the specified features. The paths for these external tools are set in configuration file under config folder:

- TreeTagger (http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/)
- SRILM (http://www.speech.sri.com/projects/srilm/download.html)
- Tokenizer (available from moses toolkit)
- Truecaser (available from moses toolkit)

Please note that above list is not exhaustive. Advance sets of features require external tools, see details in corresponding section.

4.1.3 To compile

```
ant -f build.xml
```

OR

rebuild using NetBeans (NetBeans project files are distributed in this release)

To prepare:

- We provide the system some language resources. These are copied to lang_resources folder. Resources are available from here: http://www.quest.dcs.shef.ac.uk/
- You copy these to: lang_resources/[language]/
- Edit the configuration file (i.e. config/config_en-es.properties)

4.1.4 Running

We tested our software on Linux and Mac OS. We have not tested it on Windows yet. We provide shell scripts to call the feature extractor for a pre-defined list of features.

For black box features:

```
./runBB.sh or bash runBB.sh
```

For glass box features:

```
./runGB_with_txt.sh or bash runGB.sh
```

Or

```
./runGB_with_xml.sh or bash runGB_with_xml.sh
```

More information about these scripts and the code itself can be found on the development guide (dist/MTFeatures.pdf).

Along with the code, we have provided configuration files and toy resources (SMT training corpus, language models, Giza files, etc) that should make the scripts above run without any problem. The actual resources used for the WMT12 shared task on QE You can download them from: http://www.quest.dcs.shef.ac.uk/

NOTE: One need to adapt the configuration file by providing the paths to the scripts where they are installed on your own system, i.e `config/config_en-es.properties`

4.1.5 Advanced Features

For these features more information about the input resources and how they can be created for new language pairs can be found in specific readme files under the relevant resource folders (all under 'lang_resources') which could be downloaded from: http://www.quest.dcs.shef.ac.uk/

4.2 Machine learning pipeline

The function of this package of Python scripts is to build models for machine translation (MT) quality estimation (QE). The input files are a set of instances with features that describe sentence pairs (source and target sentences). The features can be extracted using the FeatureExtractor program as explained above.

4.2.1 Installation

The program itself does not require any installation step, it is just a matter of running it provided that all the dependencies are installed.

4.2.2 Dependencies

All the machine learning algorithms are implemented by the scikit-learn library. This program provides a command-line interface for some of the implementations contained in this toolkit. In order to be able to run, the program requires that the following packages are installed in your Python distribution:

- numpy (http://scipy.org/Download)
- scikit-learn (http://scikit-learn.org/stable/install.html)
- pyyaml (http://pyyaml.org/)

4.2.3 Running

Note: Following commands are based on the assumption that all files are under the 'learning' directory.

The program takes only one input parameter, the configuration file. For example:

```
python src/learn_model.py config/svr.cfg
```

4.2.4 Configuration file

The configuration uses the YAML format (http://www.yaml.org/). Its layout is quite straight-forward. It is formed by key and value pairs that map directly to dictionaries (in Python) or hash tables with string keys. One example is as follows:

```
learning:
    method: LassoLars
    parameters:
        alpha: 1.0
        max_iter: 500
        normalize: True
        fit_intercept: True
        fit_path: True
        verbose: False
```

Each keyword followed by a ":" represents an entry in a hash. In this example, the dictionary contains an entry "learning" that points to another dictionary with two entries "method" and "parameters". The values of each entry can be lists, dictionaries or primitive values like floats, integers, booleans or strings. Please note that each level in the example above is indented with 4 spaces.

For more information about the YAML format please refer to http://www.yaml.org/.

The configuration file is composed of three main parts: input and generic options, feature selection, and learning.

Input comprises the following four parameters:

```
x_train: ./data/features/wmt2012_qe_baseline/training.qe.baseline.tsv
y_train: ./data/features/wmt2012_qe_baseline/training.effort
x_test: ./data/features/wmt2012_qe_baseline/test.qe.baseline.tsv
y_test: ./data/features/wmt2012_qe_baseline/test.effort
```

The first two are the paths to the files containing the features for the training set and the responses for the training set, respectively. The last two options refer to the test dataset features and response values, respectively.

The format of the feature files is any format that uses a character to separate the columns. The default is the tabulator char (tab, or '\t') as this is the default format generated by the features extractor module.

Two other options are available:

```
scale: true
separator: "\t"
```

'scale' applies scikit-learn's scale() function to remove the mean and divide by the unit standard deviation for each feature. This function is applied to the concatenation of the training and test sets. More information about the scale function implemented by scikit-learn can be found at http://scikit-learn.org/dev/modules/generated/sklearn.preprocessing.scale.html

'separator' sets the character used to delimit the columns in the input files.

Configuration files for some of the implemented algorithms are available in the config/ directory.

4.2.5 Available algorithms

Currently these are the algorithms available in the script:

4.2.5.1 SVR: epsilon Support Vector Regression

The parameters exposed in the "Parameters" section of the configuration file are:

- C
- epsilon
- kernel
- degree
- gamma
- tol
- verbose

Documentation about these parameters is available at
http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html#sklearn.svm.SVR

4.2.5.2 SVC: C-Support Vector Classification

The parameters exposed in the "Parameters" section of the configuration file are:

- C
- coef0
- kernel

- degree
- gamma
- tol
- verbose

Documentation about these parameters is available at
http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC

4.2.5.3 LassoCV: Lasso linear model with iterative fitting along a regularization path.

The best model is selected by cross-validation. The parameters exposed in the "Parameters" section of the configuration file are:

- eps
- n_alphas
- normalize
- precompute
- max_iter
- tol
- cv
- verbose

Documentation about these parameters is available at
http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoCV.html#sklearn.linear_model.LassoCV

4.2.5.4 LassoLars: Lasso model fit with Least Angle Regression (Lars)

The parameters exposed in the "Parameters" section of the configuration file are:

- alpha
- fit_intercept
- verbose
- normalize
- max_iter
- fit_path

Documentation about these parameters is available at
http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoLars.html#sklearn.linear_model.LassoLars

4.2.5.5 LassoLarsCV: Cross-validated Lasso using the LARS algorithm

The parameters exposed in the "Parameters" section of the configuration file are:

- max_iter
- normalize
- max_n_alphas
- n_jobs

- cv
- verbose

Documentation about these parameters is available at
http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoLarsCV.html#sklearn.linear_model.LassoLarsCV

4.2.6 Parameter optimization

It is possible to optimize the parameters of the chosen method. This option is set by the "optimize" setting under "learning" in the configuration file. The script uses the scikit-learn's GridSearchCV implementation of grid search to optimize parameters using cross-validation. To optimize the C, gamma and epsilon parameters for the SVR, the learning section of the configuration file could look as follows:

```
learning:
    method: SVR
optimize:
    kernel: [rbf]
    C: [1, 10, 2]
    gamma: [0.0001, 0.01, 2]
    epsilon: [0.1, 0.2, 2]
    cv: 3
    n_jobs: 1
    verbose: True
```

The parameter kernel is a list of strings representing the available kernels implemented by scikit-learn. In this example only the "RBF" kernel is used.

- The SVR parameters C, gamma and epsilon are set with lists with 3 indexes:
    - the beginning of the range (begin value included)
    - the end of the range (end value included)
    - the number of samples to be generated within [beginning, end]
- The remaining parameters modify the behavior of the GridSearchCV class:
    - cv is the number of cross-validation folds
    - n_jobs is the number of parallel jobs scheduled to run the CV process
    - verbose is a boolean or integer value indicating the level of verbosity

For more information about the GridSearchCV class please refer to
http://scikit-learn.org/stable/modules/generated/sklearn.grid_search.GridSearchCV.html#sklearn.grid_search.GridSearchCV

4.2.7 Feature selection

Another possible option is to perform feature selection prior to the learning process. To set up a feature selection algorithm it is necessary to add the "feature_selection" section to the configuration file. This section is independent of the "learning" section:

```
feature_selection:
    method: RandomizedLasso
    parameters:
        cv: 10

learning:
    ...
```

Currently, the following feature selection algorithms are available:

4.2.7.1 RandomizedLasso

Works by resampling the training data and computing a Lasso on each resampling. The features selected more often are good features. The exposed parameters are:

- alpha
- scaling
- sample_fraction
- n_resampling
- selection_threshold
- fit_intercept
- verbose
- normalize
- max_iter
- n_jobs

These parameters and the method are documented at:
http://scikit-learn.org/stable/modules/generated/
sklearn.linear_model.RandomizedLasso.html#sklearn.linear_model.RandomizedLasso

4.2.7.2 ExtraTreesClassifier:

Meta estimator that fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. The exposed parameters are:

- n_estimators
- max_depth
- min_samples_split
- min_samples_leaf
- min_density

- max_features
- bootstrap
- compute_importances
- n_jobs
- random_state
- verbose

Documentation about the parameters and the method can be found at:
http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html
#sklearn.ensemble.ExtraTreesClassifier

4.3 Learning with Gaussian Process

The function of this package of Python scripts is to build models for machine translation (MT) quality estimation (QE) using Gaussian Process. The input files are a set of instances with features that describe sentence pairs (source and target sentences).

4.3.1 Installation

The program itself does not require any installation step, it is just a matter of running it provided that all the dependencies are installed.

4.3.2 Dependencies

All the machine learning algorithms are implemented by the GPy library. This program provides a command-line interface for some of the implementations contained in this toolkit. In order to be able to run, the program requires that the following packages are installed in your Python distribution:

- GPy (https://pypi.python.org/pypi/GPy)
- sciPy (http://scipy.org/Download)

4.3.3 Running

Note: Following commands are based on the assumption that all files are under 'learning' directory.

The program takes only one input parameter, the configuration file. For example:

```
python src/GP_wmt_regression.py
```

Please set the path in above script to the input files. e.g

```
X = np.genfromtxt('train-79-features.qe.tsv')
test_X = np.genfromtxt('test-79-features.qe.tsv')
Y = np.genfromtxt('qe_reference_en-es.train.effort').reshape(-1, 1)
test_Y = np.genfromtxt('qe_reference_en-es.test.effort').reshape(-1, 1)
```

4.4 License

The license for the Java code and any python and shell scripts developed here is the very permissive BSD License (http://en.wikipedia.org/wiki/BSD_licenses). For pre-existing code and resources, e.g., scikit-learn and Berkeley parser, please check their websites.

5 Scorecard

UPDATE: The content of this section has been moved to the following URL:

http://www.qt21.eu/launchpad/node/1343

5.1 License

The Scorecard software is licensed under a Creative Commons Attribution-NoDerivatives 4.0 International License. Feedback and requests for permission to adapt this software should be sent info@qt21.eu. No warranty is made for the suitability of this software for any business purpose.

6 Workflows

The tools and resources developed in QTLaunchPad support data acquisition and preprocessing workflows on the one hand using the META-SHARE repository for (crawled) corpora and preprocessing pipelines (see D3.3.1, D4.3.2, D4.4.1, etc.). On the other hand, it supports evaluation workflows using QuEst, and translate5/scorecard.

While the usage scenario of corpus acquisition and preprocessing should be known to most developers, the evaluation pipelines probably need more explanation. In the course of preparing the corpora for the Shared Task (WP5) and error analysis (WP1), conducted together with GALA and several LSPs, the QTLaunchPad consortium has developed valuable experience for what could become best practice in evaluation in future work.[2]

Given MT-translated corpora and initial hypotheses of what issues may be encountered, the following steps are included in this workflow:

1. Definition of a concrete metric for the given purpose starting from an existing metric of from scratch.
2. Filtering the translation corpus to be evaluated in a triage step that distinguishes between:
    a. Perfect translations.
    b. Almost good translations that need further analysis
    c. Bad translations that do not qualify for further use.

These steps can be performed manually, supported by a very basic score card, or performed in a semi-automatic or automated fashion using QuEst. The best method de-

[2] Much of this experience is summarized in the QTLaunchPad supplemental report *Practical Guidelines for the Use of MQM in Scientific Research on Translation Quality* (http://qt21.eu/downloads/MQM-usage-guidelines.pdf)

pends on the size of the corpus, the available human resources, and the required precision and recall.

3. Deeper analysis of the segments of type b (almost good) is then conducted. If information on the segment level is sufficient, the scorecard can be used; alternatively, for detailed error annotation, translate5 can be used.
4. (Re-)Training of QuEst on the newly annotated data from step 3 and possibly on the new filtered data from step 2.
5. Inspection of the errors to:
    a. Confirm if the system output supports the hypotheses
    b. Get a quantitative basis to decide on MT development priorities
    c. Get a qualitative idea of remaining barriers

Based on the insights gained in step 5 (and eventually the annotations gained in step 3), the MT engine can then be improved. If the metric needs to be adjusted, another development cycle starts at step 1, otherwise, new translations of an improved engine can feed into step 2.

Through the improvement of QuEst over time, automation of the manual steps should become more reliable and reduce the human resources needed.

The procedure outlined above pertains to both MT development in research and in a production context. It will be tested and further developed in future research projects such as QT21 and also in cooperation with GALA through a CRISP initiative that seeks to support translate5 and other open source tools that help to implement important workflows needed by LSPs such as the one outlined above.

7 Change log

This version contains the following changes from the initial version of June 30, 2014:

- Tutorial-type usage guidelines for translate5 and the MQM Scorecard have been moved to the QT21 website.
- Information on the translate5 API has been added to this document.
- Information on the translate5 license has been added.
- Added section on workflows
- Added reference to supplemental report that describes usage scenarios for translate5 and the MQM scorecard and describes their respective features.