FALCON

D3.1: SYSTEM TEST SUITE

Leroy Finn, David Lewis, Kevin Koidl, Sandipan Dandipat, Ankit Srivastava, Alfredo Maldonado, Bruno Sanz del Campo

Distribution: Public Report

Federated Active Linguistic data CuratiON (FALCON)

FP7-ICT-2013-SME-DCA

Project no: 610879

Document Information

Deliverable number:	D3.1
Deliverable title:	System Test Suite
Dissemination level:	PU
Contractual date of delivery:	31st May 2014
Actual date of delivery:	Draft submitted , final 22 Oct 2014
Author(s):	Leroy Finn, David Lewis, Kevin Koidl, Sandipan Dandipat, Ankit Srivastava, Alfredo Maldonado, Bruno Sanz del Campo
Participants:	Interverbum, TCD, DCU
Internal Reviewer:	DCU
Workpackage:	WP3
Task Responsible:	TCD
Workpackage Leader:	Interverbum

Revision History

Revision	Date	Author	Organization	Description
1	30/4/2014	Leroy Finn	TCD	Initial test suite draft
2	6/5/2014	Leroy Finn	TCD	Described the testing methodology for the L3Data model and api along with the RESTful Service
3	22/10/2014	Sandipan Dandipat, Ankit Srivastava, Alfredo Maldonado, Bruno Sanz del Campo	TCD, DCU	Add and describe the linguistic data part of test suite

Contents

Document Information 2

Revision History 2

1. Introduction 4

2. Functional Data Testing 6

2.1. JUnit testing L3Data bootstrap 6

JUnit testing 6

What should be tested? 6

JUnit testing pattern 6

2.2. Testing of L3Data RESTful API 7

Rest Stress Tests 7

2.3. L3Data model query testing 7

How gold standard L3Data output is compared to implementers output? 8

Running SPARQL test script files 8

2.4. Results 9

3. Linguistic Data Testing 9

3.1. Overview of incremental MT training and source term capture tests 10

3.2. Feature Definitions 10

MTC - Initial MT Confidence score 10

QE - Quality Estimation score 10

TCF - Term Candidate Frequency 10

TTS - Source Terms in Termbase 11

TTT - Source Terms in Termbase but not translated as per Termbase 12

TTM - Term morphologies and TTN - Term morphologies not translated as per Termbase 13

3.3. Initial Feature Set Selection 13

Revised scores 13

Combining scores and feature scaling 14

Training schemes 14

3.4. Specification of Linguistic Test Data 17

4. Conclusions 19

1. Introduction

This is the FALCON test suite documentation that describes the testing approach that will be used for testing the various aspects of the L3Data Platform, including the core logger component as described in deliverable D2.2 and the associated language technology components.

The components and interfaces within the FALCON Showcase System that are therefore subject to the test suite are highlighted in bold in Figure 1.

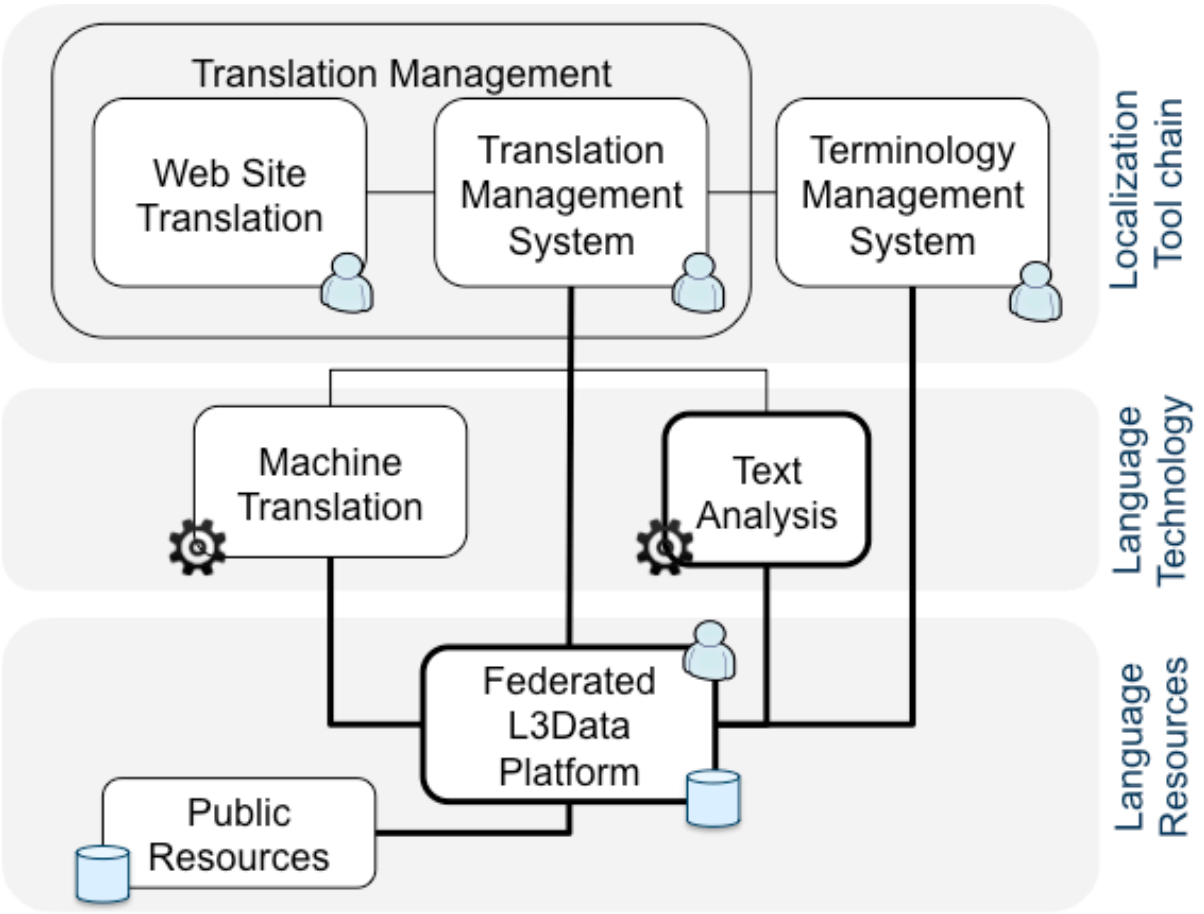

Figure 1: Component and Interface Subject to the L3Data Platform Test Suite

The test suite addresses two types of testing:

- **Functional Data Testing:** This exercises the conversion of the native language data formats into RDF according to the L3Data Schema. For operation in the FALCON Showcase System the primary input data format is XLIFF files from XTM, and associated Term Base eXchange (TBX) files from XTM and TermWeb. However other data formats relevant to interoperation with existing language resource formats will be explored, including TMX and Comma Separate Value for parallel text and the output of text analysis services. These tests therefore consist of small input format samples to test the correct functionality of the L3Data logger component.
- **Linguistic Data Testing:** This exercises both the scalability of the L3Data Schema on an open source linked data store and the performance of language technology components, specifically machine translation and term identification components.

The FALCON test suite will be distributed publicly via the FALCON Github repository[1]. Data generated from importing external language resource data sources will be published in the FALCON

[1] https://github.com/CNGL-repo/Falcon

Data Portal, both in serialised Turtle format[2] and via a SPARQL (i.e. standard RDF Query) end point[3]. The Data Portal will provide a source for further linguistic data testing.

The two forms of testing are described in more detail below.

2. Functional Data Testing

The FALCON test suite will consist of tests of the following which are described in this section.

1. JUnit testing L3Data bootstrap
2. Testing of L3Data RESTFul API using TIPP files from XTM containing XLIFF data
3. L3Data model query testing

2.1. JUnit testing L3Data bootstrap

A ***unit test*** is a piece of code which is written test some specific functionality of an application. JUnit is a unit testing framework in Java. In the context of the L3Data bootstrap, the JUnit framework will test the following complex functionalities:

- Uploading new files and converting them into RDF
- Uploading new versions of files and merging the data with the old semantic graph
- To send SPARQL queries to the SPARQL endpoint

JUnit testing

What should be tested?

Generally, trivial code should not be tested such as getters and setters methods. Doing these trivial tests is pointless because in essence you are just testing the JVM by doing these tests. JUnit tests should be written for any critical and complex locations of the application. For example, in L3Data bootstrap tests will be written for uploading new files, uploading new versions of files and using the bootstrap to send SPARQL queries to the SPARQL endpoint.

JUnit testing pattern

JUnit testing is process driven so it is good to have a pattern/procedure to do these tests (as seen in the Figure 2). The ***test fixture*** is a fixed state of the software under test used as a baseline for running tests. The ***TestFixture*** attribute tags the test class, next the ***SetUp*** attribute tags the method that sets up the tests, tests are done in the Test tagged methods and then final the cleaning up after the test is done by the method tagged by ***TearDown***.

Due to there being a couple of JUnit test classes in the L3Data bootstrap implementation they have to be tied together using a JUnit ***test suite***. Running a test suite will execute all test classes in that suite in the specified order. Extra test class can be added by using ***@SuiteClasses({MyTest.class, MyTest2.class })***.

[2] http://phaedrus.scss.tcd.ie/falcon/web/
[3] http://phaedrus.scss.tcd.ie/falcon/sparql/

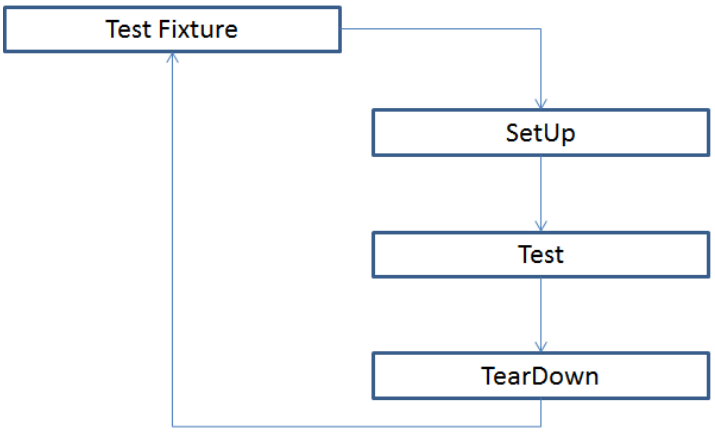

Figure 2: JUnit testing pattern

2.2. Testing of L3Data RESTful API

This part of the test suite is used to determine if the RESTful service is working correctly by checking does it give the excepted HTTP response from a set of HTTP requests. Testing and validating REST services in Java is harder than in dynamic languages such as Ruby and Groovy. REST Assured brings the simplicity of using these languages into the Java domain. More details can be found at the following address: https://code.google.com/p/rest-assured/.

REST STRESS TESTS

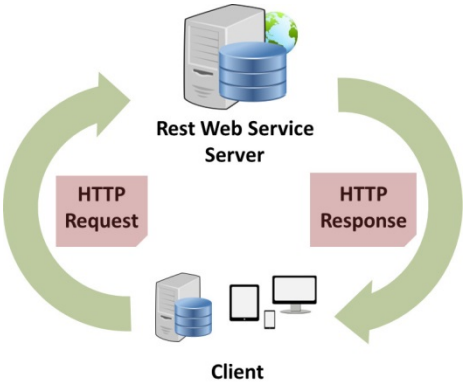

Figure 3: Rest testing

REST Assured is a Java DSL for simplifying testing of REST based services built on top of HTTP Builder. It supports POST, GET, PUT, DELETE, OPTIONS, PATCH and HEAD requests and can be used to validate and verify the response of these requests. The test will take place on all available RESTful services available which are:

- POST Uploading TIPP files
- PUT Updating TIPP files
- GET Querying the triple store

2.3. L3Data model query testing

This part of the test suite is used to determine whether an implementation of L3Data model ontology has been mapped correctly according to the L3Data model. The linked data test suite has a set of test documents in XLIFF 1.2 standard which tests the various aspects of the L3Data model used by FALCON. These XLIFF input files which are required to do these are located in the f**alcon/testsuite/input** folder. The validation of an implementers mapping is done through a set of SPARQL queries which are located in the **/falcon/testsuite/test/validation.jar.** These SPARQL queries are done over the resulting RDF, which the implementer will create when they convert the test input XLIFF files using some XML processing to RDF. If the queries are successful then there mappings to the L3Data model are correct.

How gold standard L3Data output is compared to implementers output?

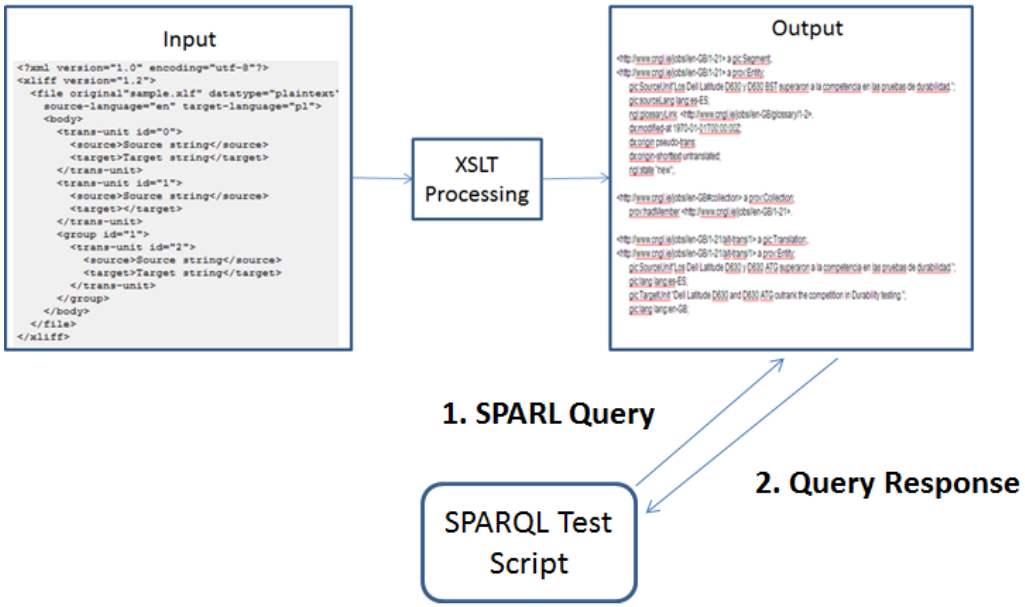

Figure 4: Model Testing

The test starts off with converting input XLIFF files (.xlf) using XSLT to output turtle files(.ttl) . The output turtle files are compared via the use of SPARQL queries done over the implementer's turtle output files (.ttl). If the SPARQL queries are successful then the implementer's output files are correct and meet the gold standard. The implementer's test output files are located in the f**alcon/testsuite/expected** folder.

Running SPARQL test script files

Prerequisites: Java and UNIX Shell

- create a temporary folder for output files (henceforth referred to as $datafolder)
- read ITS files from " **falcon/testsuite/input/"** one by one, convert to NIF and write output files in turtle to $datafolder
- go to directory **cd falcon/testsuite/sparqltest**
- to run the test script : **./executeAllQueries.sh ../relative/pathTo/$datafolder**

Explanations of output:

- If no message appears between "Running: test1.sparql" and "Done: test1.sparql" the test was successful.
- Otherwise the output filename and additional debug output is shown.

2.4. Results

The results of the functional tests will be published in the test suite GitHub. The test results are located into 4 different folders to indicate the four different test types. The full results of the test suite will be located in the GitHub as testing should be an on-going process during development.

3. Linguistic Data Testing

Distinct from software testing, the testing of the performance of language technology components in FALCON, specifically the MT component and the text analysis component, used for term identification is dependent on the training corpora used and its relationship to the textual content being processed. Selection of these corpora will be based on the following criteria:

As retraining of LT components using corrections made during a translation project is a key objective of FALCON, MT testing must demonstrate the impact of incremental retraining. For these purposes the automated measurement techniques used for MT evaluation can be modified, such that the translations of the test data to be translated are progressively fed back into a retraining process thereby simulating the post-editing of the source. In such evaluation we are less interested in achieving high MT evaluation scores, but more in the rate at which such scores improve over retraining iterations, normalised for the volume of retraining data. At a later stage these automated score will be cross-referenced with evaluation of post-editing time conducted with professional translators.

For evaluating the impact of terminology, we aim to use text analytics services to annotate terms or phrases and then feed translations of those terms into the MT engine to force their correct translation as well as directing posteditors to capture terms that were not previously translated in the term base or did not have translation for the morphology being translated.

This will later also be evaluated from a usability viewpoint, since the role of approving or correcting such annotation and their translation are not as common a function. In this case we aim to use a realistic test case. We have secured access to the SNOMED[4] clinician term base through a research license agreement with the International Health Terminology Standards Development Organisation. We are aiming to apply this in a scenario involving the translation of a medical web site, with collaboration already underway with the International Society for Stem Cell Research in relation to translation of aspects of their website content[5].

Below we describe the LT incremental training evaluation in more detail, provide an overview of the test data assembled and describe the range of linguistic features being considered for prioritising segments for post-editing with incremental MT training.

[4] http://www.ihtsdo.org/snomed-ct/
[5] http://www.isscr.org/

3.1. Overview of incremental MT training and source term capture tests

This section defines features for sorting segments imported from the XTM system into the FALCON L3Data component. The order in which post-editors review and correct machine-translated segments can have an impact on the quality score differential between partial dynamic retrains of an SMT system. That is, assuming that post-editors work on batches in which each batch contains a subset (but not the totality) of the segments from a sizeable translation project, and assuming that after each batch the SMT system is dynamically retrained, the order in which post-editors work on these batches will have an impact on how quickly the overall translation quality output from the dynamically retrained SMT system grows. The expectation is that if post-editors work first on the segments that have the most quality issues, the system will be able to *learn* the corrections for these issues early and as a consequence, translation quality will increase more steeply in the first few retraining iterations. It is possible to devise a process in which the most experienced and potentially more expensive post-editors/translators tackle the first few batches of segments, leaving the rest of the segments to either be worked upon by less experienced and potentially cheaper post-editors/translators, or to be left completely unedited, depending on the quality vs. cost requirements of the actual translation project at hand.

We therefore need to decide on strategies to sort these segments and group them into batches based on some criteria. The purpose of this document is to define the sorting criteria or features. We consider the following sorting features:

- MTC - Initial MT Confidence score
- QE - Quality Estimation score
- TCF - Term Candidate Frequency
- TTS - Source Terms in Termbase
- TTT - Source Terms in Termbase but not translated as per Termbase
- TTM - Term morphologies
- TTN - Term morphologies not translated as per Termbase

The following sections define each of these features.

3.2. Feature Definitions

MTC - Initial MT Confidence score

This feature is the initial MT Confidence score that we obtain directly from the DCU MT system. This information comes in the XLIFF file. We only need to extract it and sort segments based on these confidence scores, from the lowest confidence values to the highest.

QE - Quality Estimation score

Quality Estimation is a modelling technique in MT that aims to forecast the post-editing effort of specific machine-translated output. Several methods exists and some partners in CNGL are currently working on this topic. It could be beneficial to incorporate it in FALCON as a sorting strategy. Segments would be ordered by quality estimation scores in ascending order, i.e. the segments estimated to have the worst quality would be at the beginning of the list.

TCF - Term Candidate Frequency

Along with the segmented XLIFF, the XTM system will return a list of term candidates mined from the source language segments. It is assumed that these term candidates are mined using standard terminology extraction techniques that take into account a combination of pattern

matching (i.e. noun phrases, certain verb phrases, etc.) and statistical techniques such as termhood statistical measures and/or machine learning techniques. This way, potentially important terms are identified, whilst at the same time highly frequent word combinations (like stop words) are excluded.

The TCF feature sorts the segments based on the mined term candidates they contain. The expectation is that those segments containing the mined terms should be post-edited first, as they could contain important, potentially domain-specific terminology that must be translated correctly. Since this potential terminology is translated and corrected in the initial stages of dynamic retraining, this could prove to be an effective way of introducing domain-specific terminology in the SMT training data quickly and as part of the natural translation and post-editing workflow.

There are several specific TCF sorting strategies. We propose the following strategies:

- TCF.N: Segments are sorted based on the number of term candidates they contain in descending order. Those segments that contain many of the term candidates appear first whilst those segments with few or no term candidates appear last. This strategy also favours longer segments, as the more term candidates a segment contains, the longer it has to be.
- TCF.NP: Segments are sorted based on the proportion of term candidates they contain in descending order. The proportion p of term candidates is p=tn, where t is the number of "term words" (i.e. number of singleton term candidates as well as number of words that belong to a multi-word term candidate) and n is the total number of words the segment has. This strategy does not favour longer segments.
- TCF.F: Segments are sorted based on the number of term candidates they contain, but weighted by the term candidate's total corpus frequency. Segments are sorted in descending order. The expectation is that those segments that have highly occurring terms should be reviewed and corrected first so that these corrections are learned in the early retraining iterations of the SMT system and subsequent MT output incorporates these corrections. TCF.F for a given segment is computed as follows: tcf.f=i(1+fi) where i is the i-th term candidate in a segment and fi is its overall corpus frequency (i.e. its frequency in the whole translation project). The log function is only used as a dampening function here in order to avoid numerical problems in case the frequencies are too high. It does not affect the ranking of the segments.
- TCF.FP: This is the proportion version of TCF.F. Segments are sorted based on the number of term candidates they contain, weighted by the total term candidate's corpus frequency and normalised by segment length. tcf.fp=1nili(1+fi). Again, i is the i-th term candidate in a segment and fi is its overall corpus frequency. n is the total number of words the segment has and li is the number of words that form term candidate i (i.e. li=1 for single-word terms, li=2 for bigrams, etc.)

TTS - Source Terms in Termbase

This feature assumes that there is a multilingual, domain-specific termbase or lexical resource. This feature does not take into account the term candidates imported from XTM as was the case in TCF, but instead uses a termbase that is expected to be manually curated and be of better quality. This termbase can be an internal or corporate termbase or a publicly available termbase. In either case, we assume to have access to such termbase via some web-exposed API.

This feature prioritises those segments that contain instances of the source terms documented by the termbase. The actual strategies used in this case, mirror those of TCF:

- TTS.N: Segments are sorted based on the number of terms they contain in descending order.
- TTS.NP: Segments are sorted based on the proportion of terms they contain in descending order. The proportion p of terms is p=tn, where t is the number of "term words" (i.e. number of singleton term candidates as well as number of words that belong to a multi-word term candidate) and n is the total number of words the segment has.
- TTS.F: Segments are sorted based on the number of terms they contain, but weighted by the term's total corpus frequency. Segments are sorted in descending order. TTS.F for a given segment is computed as follows: tts.f=i(1+fi) where i is the i-th term in a segment and fi is its overall corpus frequency (i.e. its frequency in the whole translation project).
- TTS.FP: This is the proportion version of TTS.F. Segments are sorted based on the number of terms they contain, weighted by the term's total corpus frequency and normalised by segment length. tts.fp=1nili(1+fi). Again, i is the i-th term candidate in a segment and fi is its overall corpus frequency. n is the total number of words the segment has and li is the number of words that form term candidate i (i.e. li=1 for single-word terms, li=2 for bigrams, etc.)

TTT - Source Terms in Termbase but not translated as per Termbase

The TTS features prioritise those segments that contain terms that are important enough to be documented in a termbase. However, it ignores completely how they are actually being translated by the MT engine. The TTT feature aims to prioritise segments that use source terms contained in the termbase but have not been translated as per documented in the termbase. This feature assumes that we have access to the target segments as translated by the MT engine at each step of the retraining process. At each retraining step, we can recompute the TTT feature and re-sort segments accordingly.

The TTT feature can make use of the TTS feature. That is, we can first compute a weighting of those segments that contain source terms from the termbase as per a TTS strategy, and then adjust the weighting based on whether they were translated as prescribed by the termbase or not. In this initial specification, we propose to modify this weighting by simple multiplication. We therefore can define the following TTT strategies:

- TTT.N: ttt.n=tts.n(1+itst1)
- TTT.NP: ttt.np=tts.np(1+itst1)
- TTT.F: ttt.f=tts.f(1+itst1)
- TTT.FP: ttt.fp=tts.fp(1+itst1)

In all of these equations we add a multiplicative term (1+itst1). Here, it is the target term corresponding to a source term is occurring in a source segment ss and st is the target segment that translates ss. Basically, we add a weight of 1 every time the target term is not found in the target segment. This search can be performed using regular expressions using some stemming technique (to avoid flagging errors due to morphological variation such as plurals, conjugations, etc.)

The 1 inside the the log function is added to avoid numerical errors in case all target terms are found in the target segment. Notice as well, that in this case the log would be zero, rendering the full value to zero. This might or might not be a desired characteristic. It might be desirable because it would fully de-prioritise segments that present termbase terms that have been translated correctly, assuming that there is a 1-1 correspondence between source and target terms, i.e. that there are no synonymous terms in the target side of the termbase. It might be undesirable because the MT engine could still be producing target terms in the wrong morphological form. Since these segments do contain important terminology, we might be interested in not fully clearing their weight so they do have a chance of being reviewed. If this is more desirable, the following alternative formulations can be employed:

- TTT.N2: ttt.n=tts.n(1+(1+itst1))
- TTT.NP2: ttt.np=tts.np(1+(1+itst1))
- TTT.F2: ttt.f=tts.f(1+(1+itst1))
- TTT.FP2: ttt.fp=tts.fp(1+(1+itst1))

The TTT.*2 strategies are effectively a combination of the TTS and the TTT features.

TTM - Term morphologies and TTN - Term morphologies not translated as per Termbase

The TTS and TTT features largely ignore the actual morphological realisations of terms in natural text or employ simplistic stemming rules based on regular expressions that might not completely detect errors in terminology usage. Assuming that morphological information is stored in the termbase, we could implement morphological-sensitive versions of the TTS and TTT features, resulting in the TTM and TTN features respectively. In addition, this morphological information can be captured as post-editors correct segments and stored in the termbase for future usage.

Since TTM and TTN require more advanced NLP-aware techniques and more discussion and since they will not be ready for the short term demo, their definition is left for a future iteration of this document.

3.3. Initial Feature Set Selection

This section documents some details about the feature subset that was implemented for the demo purposes for the Loc World Conference in Vancouver. The implementation does not include all of the features documented above, but a subset following some modifications which are mentioned here.

Revised scores

For the demo, it was decided to settle on a combination of a version of TCF.F (which we renamed as **Frequency Score** or **FS**) with confidence score.

The Frequency Score of a segment i is computed as follows:

FSi=j(fj)

where j is a term occurring in segment i and fj is the **segment frequency** of j, that is, the number of segments in the project or sample where j appears at least once. It was thought that segment frequency (rather than total term frequency) is slightly more indicative of how widespread a term is in the project, sample or corpus. The range of values for FS is 0FS+.

It was also decided to use MT **Confidence Score** (**CS**), which seems to be a log value of some description. The range of values for CS seems to be -CS+, with the lower values indicating less confidence and higher values higher confidence. Notice however that we want to select those

segments with the least confidence score, whilst we want to select those segments with the highest frequency score. To make the two features rank segments in descending order, it was decided to use an "inverse" of the confidence score that we just call **inverse Confidence Score** (**iCF**), in which every CF is just multiplied by -1, therefore reversing the scale.

Combining scores and feature scaling

It was decided to combine the FS and iCS as a weighted sum. The issue is that both FS and iCS seem to be in different "units" in different scales. So, for the weighting to be effective, it was decided to bring them to a similar scale based on conventional feature scaling techniques. The strategy applied can be described as follows. Given a vector of feature scores s (scores can be FS, iCS or something else), normalise and scale as follows:

$$sn=s-(s)$$
$$sf=sn(sn)$$

This has the effect of assigning values between 0 and 1 to all values in the vector, with a value of 0 for the lowest value and 1 to the highest value.

If fs and ics are the scaled versions of FS and iCS, respecitvely, then they are combined in the following manner to form the new ranking value stat:

$$stat=w1fs+w2cs$$

where w1 and w2 are weights such that w1+w2=1. For the demo, we decided to try out the following value combinations for w1 and w2:

w1	w2
0.0	1.0
0.2	0.8
0.5	0.5
0.8	0.2
1.0	0.0

Training schemes

In-domain and cross-domain schemes are being considered:

- **COMBINATION 1**
  - **In-domain 1 (trn:dgt, tst:dgt)**
    - in-domain/dgt/
      - en-fr
      - en-pl
    - Files: in_domain/dgt/*/*-mix-*
  - **Cross-domain 1 (trn:ep, tst:dgt)**
    - cross-domain/ep/
      - en-fr
      - en-pl

        - Files: cross_domain/ep/*/*-mix-*
- **COMBINATION 2**
    - **In-domain 2 (trn:ep, tst:ep)**
        - in_domain/ep/
            - en-fr
            - en-pl
        - Files: in_domain/ep/*/*-mix-*
    - **Cross-domain 2 (trn:dgt, tst:ep)**
        - cross_domain/dgt/
            - en-fr
            - en-pl
        - Files: cross_domain/dgt/*/*-mix-*

In **COMBINATION 1**, the following weight files are being created:

In-domain 1 (trn:dgt, tst:dgt)

pair	trn size	w1 (fs)	w2 (ics)	File
EN-FR	5	0.0	1.0	in_domain/dgt/en-fr/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf5-0.0_1.0
EN-FR	5	0.2	0.8	in_domain/dgt/en-fr/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf5-0.2_0.8
EN-FR	5	0.5	0.5	in_domain/dgt/en-fr/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf5-0.5_0.5
EN-FR	5	0.8	0.2	in_domain/dgt/en-fr/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf5-0.8_0.2
EN-FR	5	1.0	0.0	in_domain/dgt/en-fr/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf5-1.0_0.0
EN-FR	50	0.0	1.0	in_domain/dgt/en-fr/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf50-0.0_1.0
EN-FR	50	0.2	0.8	in_domain/dgt/en-fr/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf50-0.2_0.8
EN-FR	50	0.5	0.5	in_domain/dgt/en-fr/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf50-0.5_0.5
EN-FR	50	0.8	0.2	in_domain/dgt/en-fr/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf50-0.8_0.2
EN-FR	50	1.0	0.0	in_domain/dgt/en-fr/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf50-1.0_0.0
EN-PL	5	0.0	1.0	in_domain/dgt/en-pl/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf5-0.0_1.0
EN-PL	5	0.2	0.8	in_domain/dgt/en-pl/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf5-0.2_0.8
EN-PL	5	0.5	0.5	in_domain/dgt/en-pl/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf5-0.5_0.5
EN-PL	5	0.8	0.2	in_domain/dgt/en-pl/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf5-0.8_0.2
EN-PL	5	1.0	0.0	in_domain/dgt/en-pl/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf5-1.0_0.0
EN-PL	50	0.0	1.0	in_domain/dgt/en-pl/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf50-0.0_1.0
EN-PL	50	0.2	0.8	in_domain/dgt/en-pl/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf50-0.2_0.8
EN-PL	50	0.5	0.5	in_domain/dgt/en-pl/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf50-0.5_0.5
EN-PL	50	0.8	0.2	in_domain/dgt/en-pl/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf50-0.8_0.2

EN-PL	50	1.0	0.0	in_domain/dgt/en-pl/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf50-1.0_0.0

Cross-domain 1 (trn:ep, tst:dgt)

pair	trn size	w1 (fs)	w2 (ics)	File
EN-FR	5	0.0	1.0	cross_domain/ep/en-fr/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf5-0.0_1.0
EN-FR	5	0.2	0.8	cross_domain/ep/en-fr/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf5-0.2_0.8
EN-FR	5	0.5	0.5	cross_domain/ep/en-fr/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf5-0.5_0.5
EN-FR	5	0.8	0.2	cross_domain/ep/en-fr/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf5-0.8_0.2
EN-FR	5	1.0	0.0	cross_domain/ep/en-fr/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf5-1.0_0.0
EN-FR	50	0.0	1.0	cross_domain/ep/en-fr/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf50-0.0_1.0
EN-FR	50	0.2	0.8	cross_domain/ep/en-fr/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf50-0.2_0.8
EN-FR	50	0.5	0.5	cross_domain/ep/en-fr/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf50-0.5_0.5
EN-FR	50	0.8	0.2	cross_domain/ep/en-fr/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf50-0.8_0.2
EN-FR	50	1.0	0.0	cross_domain/ep/en-fr/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf50-1.0_0.0
EN-PL	5	0.0	1.0	cross_domain/ep/en-pl/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf5-0.0_1.0
EN-PL	5	0.2	0.8	cross_domain/ep/en-pl/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf5-0.2_0.8
EN-PL	5	0.5	0.5	cross_domain/ep/en-pl/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf5-0.5_0.5
EN-PL	5	0.8	0.2	cross_domain/ep/en-pl/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf5-0.8_0.2
EN-PL	5	1.0	0.0	cross_domain/ep/en-pl/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf5-1.0_0.0
EN-PL	50	0.0	1.0	cross_domain/ep/en-pl/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf50-0.0_1.0
EN-PL	50	0.2	0.8	cross_domain/ep/en-pl/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf50-0.2_0.8
EN-PL	50	0.5	0.5	cross_domain/ep/en-pl/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf50-0.5_0.5
EN-PL	50	0.8	0.2	cross_domain/ep/en-pl/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf50-0.8_0.2
EN-PL	50	1.0	0.0	cross_domain/ep/en-pl/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf50-1.0_0.0

Notice that within a language pair the 5k vs 50k difference is only relevant when the ics feature is non-zero. This is because the fs feature on its own is not relevant to the training size. Also, the ranking by fs feature alone is not affected by whether in_domain or cross_domain training has been performed. Therefore, the following files are virtually equivalent to each other:

- EN-FR
    - in_domain/dgt/en-fr/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf5-1.0_0.0
    - in_domain/dgt/en-fr/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf50-1.0_0.0

    - cross_domain/ep/en-fr/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf5-1.0_0.0
    - cross_domain/ep/en-fr/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf50-1.0_0.0
- EN-PL
    - in_domain/dgt/en-pl/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf5-1.0_0.0
    - in_domain/dgt/en-pl/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf50-1.0_0.0
    - cross_domain/ep/en-pl/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf5-1.0_0.0
    - cross_domain/ep/en-pl/inc_train.tok_lc_clean.en.rank-mix-frscore_invconf50-1.0_0.0

As a consequence, we can run just one experiment per language pair for all of these configurations, thus reducing the experiment combination space.

Notice that the fs feature only takes into account source language data (ie English), so in principle there shouldn't be a need for running one experiment for (EN-FR, w1=1, w2=0) and another one for (EN-PL, w1=1, w2=0), as only one of these would suffice. However, there is a difference between the source segments selected for the EN-FR and the EN-PL pairs, so the rankings would also be different. That is why we need to run separate experiments for each language pair.

3.4. Specification of Linguistic Test Data

In this section we detail the linguistic test suite based on the initial feature set selection. It will consist of translation and terms from good quality public resources, namely the DG-T translation memories, the EuroParl parallel text resource, the EURVOC term base and, later, the SNOMED-CT term base. For the LT component testing the test suite will be captured as Comma Separated Value files, as these can be easily consumed by existing LT testing scripts. These test sets will also be converted into the L3Data format in order to provide a substantial RDF test set over which SPARQL performance tests can be conducted that correspond to the scale of scenarios we are evaluating with the LT tests.

Specifically the Linguistic Data test suite will consist of the following per tested language pair (initially en-fr and en-pl):

- **EURVOC terms:** This is a tab-separated representation of the EUROVOC for terms only. Concepts are ignored. There are some blank terms. These are terms for which there is an ID but for which there was not a term in the term base. These can be ignored. It also contains terms with negative ID, which represent terms for which an ID was no present. Some of these terms could be related to the old way of capturing terms or they could be superseded terms. All of the terms with positive IDs are correctly captured by my code and are properly documented in EUROVOC.
- **In-domain parallel text:** These pertain to the MT System that was trained on DGT data and uses DGT data for retraining ("in-domain 1" system).
- **Initial MT training data:** This is a sample of the DGT files used to do the initial training the MT system. You probably won't need to work on this file, but decided to include it anyway.

- **Clean source training data:** This is a sample of the source (English) DGT files extracted for the en-fr language pair. It is tokenised and normalised to lowercase. Each line in this file represents a segment. This file is initially translated by the system and used for retraining. We match EUROVOC terminology against the segments contained in this file.
- **Source File reference:** This is a mapping of the segments in clean source training data to the actual segment in the DGT TMX files. Each line in this file corresponds to a line in clean source training data. Each line in inc_train_segment_id field reads like "vol_2013_2/32013d0283.tmx 1". This points to the file 32013d0283.tmx inside the vol_2013_2.zip file and the 1 represents the first segment in this file. Segments are counted from number 1. This allows the linked data version of the test data to be linked back to the original source document
- **Terms in source:** This is a tab-separated file containing the EUROVOC terms that were found in clean source training data, as per a regular expression matching script. It contains two columns: segix and termids. "segix" is the line number in clean source training data (the first line is 1), and "termids" is a space-separated list of term IDs found. The IDs are the same as those in eurovoc_flat.txt. Recall that positive IDs are EUROVOC IDs whilst negative IDs are internal IDs generated in response to errors in processing the source data.
- **Term Statistics:** A tab-separated file, this one counts overall term count statistics. Columns are:
  - termid - Same term ID as before
  - term - The actual term (normalised to lower case)
  - SegFreq - This is the "segment frequency" of the term, i.e. the number of segments in clean source training data that contain the term.
  - TMFreq - This is the overall "term frequency" of the term, i.e. the total number of times the term appears in clean source training data.
  - IDF - This is the "inverse document frequency" of the term. This is an information retrieval measure computed as log(N/SegFreq), where N is the total number of segments in clean source training data.
- **Frequency Score Segment Ranking:** This is the ranking of segments (lines) in clean source training data based on the "frequency score" feature. More info about this feature can be found in the "Initial Frequency Score" section below,
- **Initial 5k MT segment targets:** A tab-separated file that contains the initial MT translation for each segment (line) in clean source training data and the MT confidence score (number after the tab). The MT system that produced this file was trained with 5k segments from the DGT files.
- **Initial 50k MT segment targets:** Same as previous file, however the MT system that produced this file was trained with 50k segments from the DGT files.
- **MT target segment TER scores:** These files contain the TER score for each segment, for the 5k segment-trained system and the 50k segment-trained system, respectively. TER is translation Error Rate, an automated machine translation measure taken to be a good measure of human postediting effort.
- **Inverse confidence Score Ranking:** Ranking of segments (lines) based on the "inverse confidence score" feature described in the "Feature definitions" document (see same

section mentioned above), based on an MT system trained with 5k segments and 50k segments, respectively.

- **Weighted features:** Files that mix the fs and ics features to different proportions (see "Feature definitions" section for more info).
- **Europarl version:** Similar to the files above. The files inside this directory pertain to the MT System that was trained on Europarl data and uses Europarl data for retraining ("in-domain 2" system).
- **Cross domain test data, initial DGT:** These files are similar to those inside in-domain/dgt, except that the training here was performed on the DGT corpus and the incremental testing and retraining is performed on Europarl. ("cross-domain 2" system)
- **Cross domain test data, initial Europarl:** Similar to previous one, except that the training here was performed on the Europarl corpus and the incremental testing and retraining is performed on the DGT corpus. ("cross-domain 1" system)

4. Conclusions

This document provides a specification both of how the functional aspect of the L3Data component will be tested and how the associated language technology components will be tested, in terms of incremental MT training performance. The latter will also provide input data that will be converted to L3Data for SPARQL query scalability tests. The actual test data will be maintained online at the provided URLs and updated as the test set used in FALCON evolve over the course of the project.