“Terminology use case for LT-Web metadata”

Documentation of the Terminology Use Case for LT-Web Metadata

Date: 31/05/2013

Version: 1.00

Revision & Sign-off Sheet

Change Record

Date	Authors	Version	Change Reference	File name
18/01/2013	Mārcis Pinnis	0.1	Draft version	Tilde-LT-Web-Documentation.docx
26/04/2013	Pēteris Ņikiforovs	0.2	Technical documentation	Tilde-LT-Web-Documentation.docx
29/04/2013	Mārcis Pinnis	0.3	Showcase user manual draft and review of the technical documentation	Tilde-LT-Web-Documentation.docx
29/04/2013	Pēteris Ņikiforovs	0.4	Corrections in the technical documentation	Tilde-LT-Web-Documentation.docx
12/05/2013	Tatiana Gornostay	0.5	Internal review, minor corrections and approval for delivery to LT-Web partners for external review	Tilde-LT-Web-Documentation.docx
16/05/2013	Pēteris Ņikiforovs	0.6	Updated XLIFF examples	Tilde-LT-Web-Documentation.docx
16/05/2013	Pēteris Ņikiforovs	0.7	Updated documentation about its-annotators-ref	Tilde-LT-Web-Documentation.docx
16/05/2013	Pēteris Ņikiforovs	0.8	Updated TBX examples	Tilde-LT-Web-Documentation.docx
17/05/2013	Mārcis Pinnis	0.9	Updated showcase user manual	Tilde-LT-Web-Documentation.docx
28/05/2013	Pēteris Ņikiforovs	0.10	Expanded the XliffAnnotator section, updated a TBX example, added a section about dependencies, added information about *EuroVoc*	Tilde-LT-Web-Documentation.docx
31/05/2013	Artūrs Vasiļevskis	1.00	Review, documentation finalized.	Tilde-LT-Web-Documentation.docx

Document Properties

Item	Details
Document Title	Documentation of the Terminology Use Case for LT-Web Metadata
Authors	Mārcis Pinnis; Pēteris Ņikiforovs
Internal review	Tatiana Gornostay
External review	Felix Sasaki, LT-Web consortium
Creation Date	18/01/2013
Last Updated	31/05/2013

CONTENTS

1. INTRODUCTION

The document „*Documentation of the Terminology Use Case for LT-Web Metadata*" has been created in order to provide the documentation for the Terminology Use Case implementation. The documentation provides the following information:

- Section 2 briefly describes the implementation of the terminology use case with respect to the provided functionality.
- Section 3 describes the ITS 2.0 data categories supported by the use case.
- Section 4 provides a user manual for the Web-based showcase of the terminology use case[1].
- Section 5 provides the technical documentation of the Web-based showcase and the Terminology Annotation Web Service (TAWS) including the guidelines for the access of the Web Service by third party solutions.

[1] http://taws.tilde.com/

2. OVERALL DESCRIPTION OF THE TERMINOLOGY USE CASE

The Terminology Use Case is designed and implemented by Tilde. It consists of two modules:

- The Web-based showcase;
- The Terminology Annotation Web Service.

Figure 1 below reflects the architecture of the terminology use case.

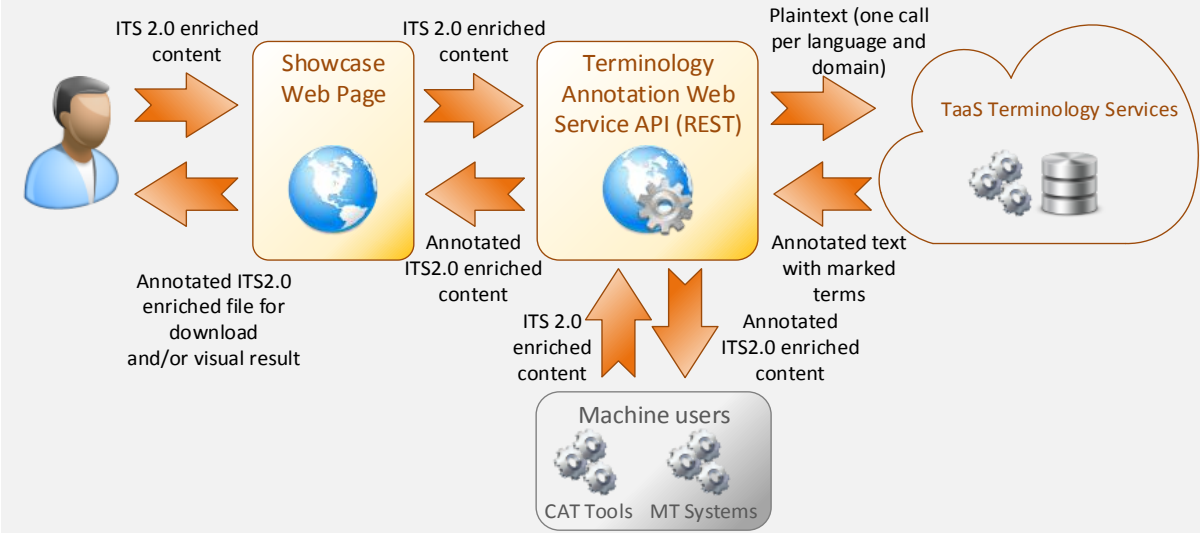

Figure 1 Overall architecture of the Terminology Use Case

2.1. Web-Based Showcase

The Web-based showcase provides a graphical user interface (GUI) for terminology annotation in ITS 2.0 enriched documents. The showcase is accessible from the Web page http://taws.tilde.com/. The showcase allows submitting plaintext, HTML5, and XLIFF documents for automatic terminology annotation. The user uploaded (or submitted) documents are forwarded to the Terminology Annotation Web Service (TAWS) that performs automatic terminology annotation. The results of the terminology annotation are presented to the user in a Web Frame and the annotated document can also be downloaded. For a detailed user guide of the Web-based showcase refer to the section 3.

2.2. Terminology Annotation Web Service

The Terminology Annotation Web Service (TAWS) performs ITS 2.0 content analysis, calls external terminology services for terminology annotation provided by the TaaS platform[2], and applies ITS 2.0 compatible terminology annotation in user submitted documents. The Web service accepts only text documents in input (file upload and URL content processing is performed by the Web-based showcase[3]). A detailed technical documentation is provided in the section 5.

3. ITS 2.0 DATA CATEGORIES IMPLEMENTED

The following data categories of the ITS 2.0 standard are supported (either consumed and/or produced) by TAWS:

- ***Domain***

The domain information is used to split and analyse the content per domain separately – this allows filtering terms in the term bank based terminology annotation as well as identifying domain-specific content using statistical term extraction systems. The user will be asked to provide a default domain for the term bank based terminology annotation, however, the default domain will be overridden with Domain metadata if present in the ITS 2.0 enriched content.

- ***Elements Within Text***

The information is used to decide which elements are extracted as in-line codes and sub-flows in the ITS 2.0 enriched content.

- ***Language Information***

The language information is used to split and analyse the content per each language separately. The user will be asked to provide a default language. The default language will, however, be overridden with Language information metadata if present in the ITS 2.0 enriched content.

- ***Locale Filter***

Whenever used only the text in the locale as specified by the user defined language is analysed. The remaining content is ignored.

[2] The TaaS platform is being developed in the TaaS project – http://www.taas-project.eu.

[3] http://taws.tilde.com/

- ***Terminology***

For existing metadata the markup is preserved and terminology markup overlaps are not allowed; for the remaining content – terms are marked according to the rules of the Terminology data category if such are present in the ITS 2.0 enriched content. TAWS will produce only Terminology metadata. Other ITS 2.0 data categories are only consumed by the terminology use case.

4. WEB-BASED SHOWCASE USER GUIDE

The Web-based showcase allows users to automatically annotate terminology in ITS 2.0 enriched content in HTML5, XLIFF, and plaintext formats.

4.1. Home page

In order to annotate documents in one of the supported formats, the user has to select the appropriate option from the Home page of the Web-based showcase (see Figure 2 below). The following three options are provided:

- "*Text*" for terminology annotation in plaintext documents;
- "*HTML5*" for terminology annotation in HTML5 documents;
- "*XLIFF*" for terminology annotation in XLIFF documents.

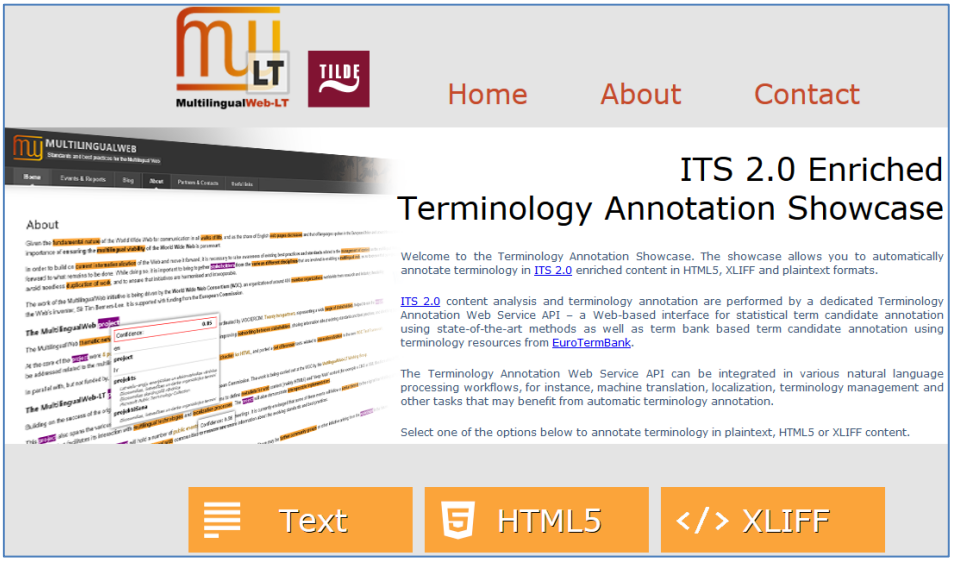

Figure 2. The Home page of the Web-based Showcase

4.2. Terminology annotation in plaintext documents

If the user selects the "*Text*" option as input, a new form is opened (see Figure 3 below) where the user has to perform the following activities:

- **Select the language** of the text that is entered for terminology annotation. Three languages are supported by the showcase: English, Latvian, and Lithuanian. The language is a mandatory parameter, and the system will not allow terminology annotation without specifying a language.
- **Select** which **terminology annotation method** to use for terminology annotation. Three options are available:
    - *Statistical terminology annotation* (terms considered domain-specific by statistical measures are annotated as term candidates with a confidence score).
    - *Term bank based terminology annotation* (only terms found in the *EuroTermBank* are annotated).
    - *Both methods combined*.

  The terminology annotation method is a mandatory parameter, and the system will not allow submitting the form for terminology annotation if none of the methods is selected.
- In the case of term bank based terminology annotation the user also has to **select a domain** in which to search for terms. If the user does not know the domain or cannot find the domain in the list, it is possible to select "All domains" in order to search for terms in the whole *EuroTermBank*[4]. The domain classification follows the structure of the first two level domain structure of the *EuroVoc thesaurus*[5].

Once the user has selected the appropriate configuration for terminology annotation, he/she is offered three options to pass data (plaintext) to the system. The user can perform the following activities:

- **Enter (or paste) plaintext** directly into the text box under the heading "*Text*".
- **Enter (or paste) a URL** that links to a plaintext file that is available on the Web under the heading "*URL*".

[4] www.eurotermbank.eu

[5] http://eurovoc.europa.eu/

- **Select and upload a file containing a plaintext document** that is located on the user's local computer under the heading "*File Upload*".

It is mandatory to select at least one data input mechanism (pasting the text, entering the URL or uploading a file), otherwise the system will return an exception saying that: "*No text was submitted.*"

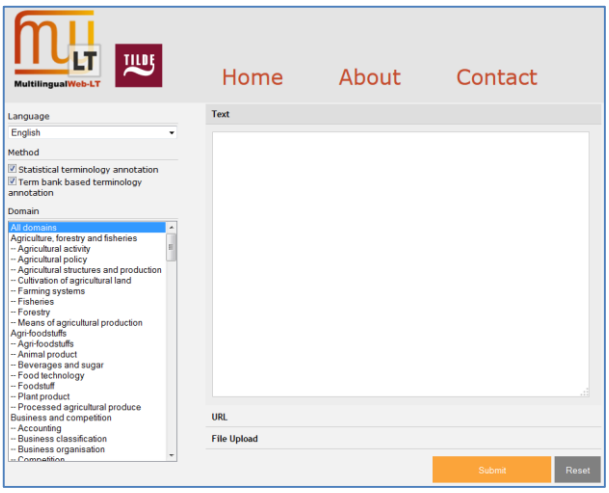

Figure 3. The "*Text*" input form of the Showcase Web Page

Once all options are selected and the data input mechanism is selected, the user has to click on the "*Submit*" button in order to start terminology annotation. Alternatively, the user can click on the "*Reset*" button in order to reset the form to its initial configuration.

After selecting "*Submit*", a new window is opened in the same tab indicating that the request for terminology annotation is being processed. As terminology annotation and lookup in *EuroTermBank* are resource intensive processes, it can take up to 1 minute to get the results for a document of roughly 50 000 characters. The user in the meantime will see the "*waiting window*" (see Figure 4 below). The showcase is also limited to 50 000 characters. If a longer document is submitted, the showcase will limit terminology annotation to the first 50 000 characters.

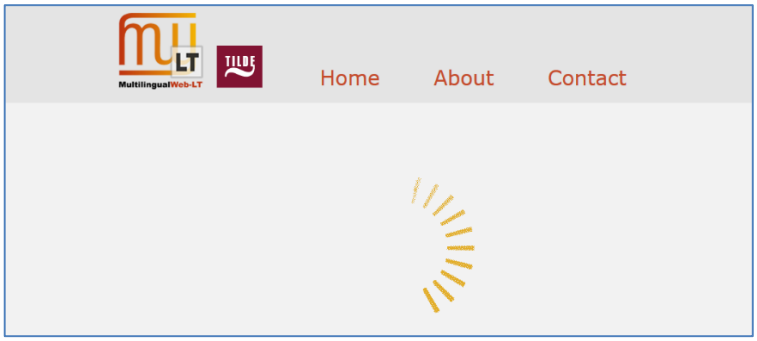

Figure 4. The waiting window

The results of terminology annotation for plaintext documents are visualised in a new window in the same tab (see Figure 5 below). Terms that are identified by the "*Statistical terminology annotation*" method are marked in orange and terms that are identified by the "*Term bank based terminology annotation*" method are marked in purple.

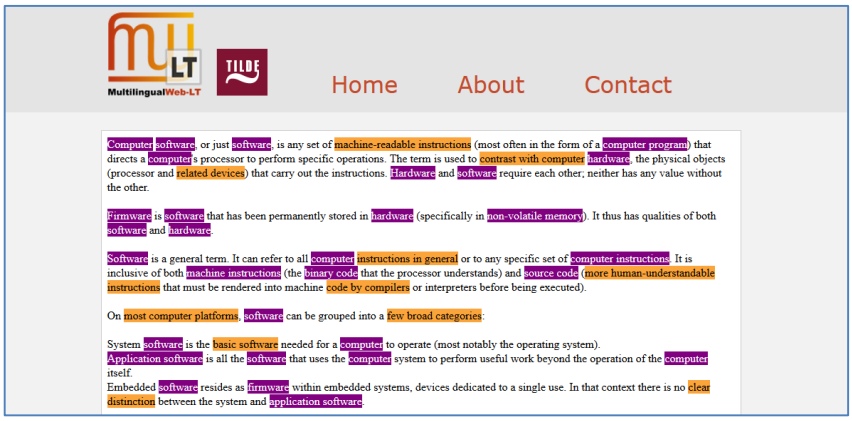

Figure 5. Visualisation example for annotated terminology in plaintext content

The user can hover with the mouse cursor over the identified terms in order to acquire more information. The terms identified by the "*Statistical terminology annotation*" method are considered to be term candidates. Therefore, for these phrases the visualisation provides a term confidence measure that is assigned to the term candidate by the statistical terminology annotation system (see Figure 6 below).

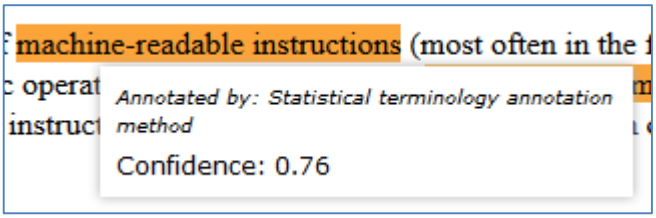

Figure 6. A tooltip window example of a term candidate identified by the Statistical terminology annotation method

The terms identified by the "*Term bank based terminology annotation*" method upon hovering over the marked phrases are visualised (see Figure 7 below) by listing the terminology resources in which the phrases can be found in *EuroTermBank*. There is no confidence score assigned to terms identified in a term bank, because it is assumed that a term within a term bank automatically gets a confidence score of "1.0" assigned, thus the confidence score is redundant.

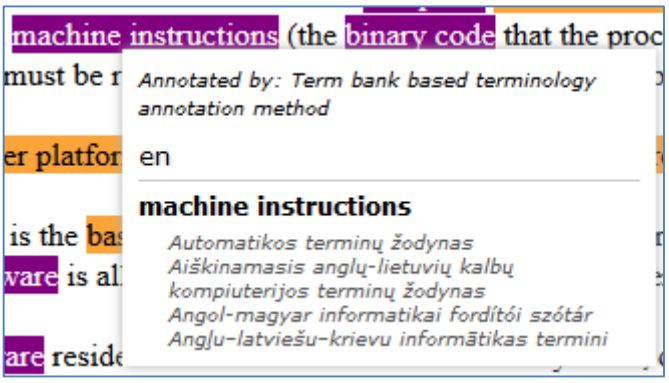

Figure 7. A tooltip window example of a term identified by the Term bank based terminology annotation method

The user can also download the annotated document by selecting the "*Download*" button at the bottom right corner of the page (see Figure 8 below).

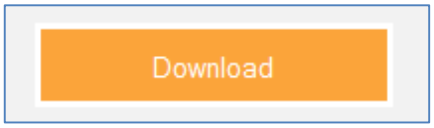

Figure 8. The "Download button"

The annotated plaintext content for the visualisation and downloading functionality is transformed to an HTML5 document in order to include ITS 2.0 metadata in the content. The terms identified by the "*Statistical terminology annotation*" method are marked using an HTML5 `<span>` element with the `its-term="yes"` and the `its-term-confidence="#.##"` attributes (see Figure 9 below).

```
<span its-term="yes" its-term-confidence="0.23">physical objects</span>
```

Figure 9. Example of an annotated term candidate using the "Statistical terminology annotation" method in HTML5

The terms identified by the "*Term bank based terminology annotation*" method are marked using an HTML5 `<span>` element with the `its-term="yes"` and the `its-term-info-ref="#tilde-tbx-taas-##-etb-##"` attributes (see Figure 10 below). The `its-term-info-ref` attribute refers to additional information describing the annotated term.

The additional information in the showcase implementation is stored within the HTML5 document itself.

```
<span its-term-info-ref="#tilde-tbx-taas-6-etb-6" its-term="yes">Hardware</span>
```

Figure 10. Example of an annotated term using
the "Term bank based terminology annotation" method in HTML5

The additional information for each term identified in *EuroTermBank* is stored in a separate `<script>` element of the type "*text/xml*", which is stored under the `<head>` element. The content of the `<script>` element contains a TBX[6] document. An example of a term entry stored in the TBX format and embedded in an HTML5 document is given in Figure 11 below. The example identifies term collections in which the term was found in *EuroTermBank* with the `<xref>` element.

```
</martif></script><script id="tilde-tbx-taas-8-etb-8" type="text/xml"><?xml version='1.0'?>
<!DOCTYPE martif SYSTEM "TBXcoreStructV02.dtd">
<martif type="TBX">
  <martifHeader>
    <fileDesc>
      <sourceDesc>
        <p>Tilde Terminology Annotation Service</p>
      </sourceDesc>
    </fileDesc>
    <encodingDesc>
      <p type="XCSURI">http://www.ttt.org/oscarstandards/tbx/TBXXCS.xcs</p>
    </encodingDesc>
  </martifHeader>
  <text>
    <body>
      <termEntry>
        <admin type="sourceLanguage">en</admin>
        <descrip type="subjectField">3236</descrip>
        <langSet xml:lang="en">
          <ntig>
            <termGrp>
              <term>non-volatile memory</term>
              <termCompList type="lemma">
                <termCompGrp>
                  <termComp>non-volatile</termComp>
                  <termNote type="partOfSpeech">adjective</termNote>
                  <termNote type="transferComment">JJ</termNote>
                </termCompGrp>
                <termCompGrp>
                  <termComp>memory</termComp>
                  <termNote type="partOfSpeech">noun</termNote>
                  <termNote type="grammaticalNumber">singular</termNote>
                  <termNote type="transferComment">NN</termNote>
                </termCompGrp>
              </termCompList>
            </termGrp>
            <descrip type="reliabilityCode">1</descrip>
            <admin type="score">0.82</admin>
            <admin target="4e0e22c8-2c41-44b0-9cfb-87e3da6d6b2f.output.step1.completed" type="sourceIdentifier" />
            <xref target="http://www.eurotermbank.com/Collection.aspx?collectionid=310" type="xSource">Aiškinamasis anglų-lietuvių kalbų kompiuterijos terminų žodynas</xref>
            <xref target="http://www.eurotermbank.com/Collection.aspx?collectionid=351" type="xSource">Angol-magyar informatikai ford&#237;t&#243;i sz&#243;t&#225;r</xref>
            <xref target="http://www.eurotermbank.com/Collection.aspx?collectionid=396" type="xSource">Anglu-latviešu-krievu informātikas termini</xref>
          </ntig>
        </langSet>
      </termEntry>
    </body>
  </text>
</martif></script><script id="tilde-tbx-taas-9-etb-9" type="text/xml"><?xml version='1.0'?>
```

Figure 11. An example of a term entry stored in a separate <script> element

In order to return to the text input form, the user has to follow the "*Go back*" link at the bottom left corner of the results page (see Figure 12 below).

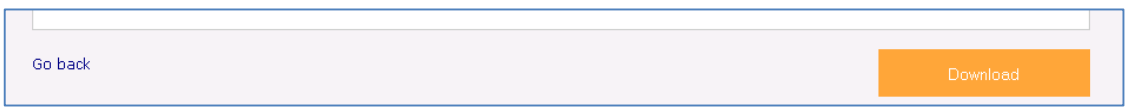

Figure 12. The "Go Back" link at the bottom left corner of the results page

[6] http://www.ttt.org/oscarStandards/tbx/

4.3. Terminology annotation in HTML5 documents

If the user selects "*HTML5*" option as input, a new form is opened (see Figure 13 below) where the user has to enter the same configuration options as for the plaintext input (i.e., language, terminology annotation method and domain; see section 4.2 for more details).

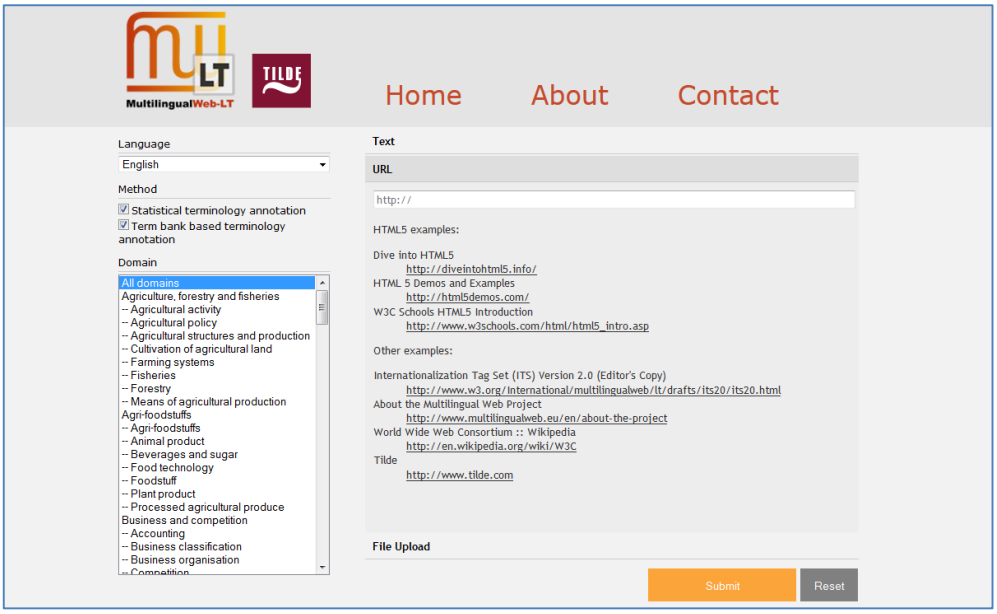

Figure 13. The "HTML5" input form of the Web-based Showcase

Once the user has selected the appropriate configuration for terminology annotation, the user is offered three options to pass data (HTML5 content) to the system.

The user can:

- **Enter (or paste) HTML5 content** directly in the text box under the heading "*Text*".
- **Enter (or paste) a URL** that links to an HTML5 document that is available on the Web under the heading "*URL*". For testing purposes the Web-based showcase offers HTML5 as well as other HTML (or XHTML) examples that can be used by the user. The examples can be passed to the system by selecting a link from the list. The Web-based showcase will automatically apply a default configuration (i.e., language, annotation method, and domain) for the selected example.
- **Select and upload a file containing an HTML5 document** that is located on the user's local computer.

It is mandatory to select at least one data input mechanism (pasting the HTML5 content, entering the URL, or uploading a file), otherwise the system will return an exception saying that: "*No text was submitted.*"

Once all options are selected and the data input mechanism is selected, the user has to click on the "*Submit*" button in order to start terminology annotation. Alternatively, the user can click on the "*Reset*" button in order to reset the form to its initial configuration.

After selecting "*Submit*", similarly to plaintext annotation (see Figure 4 above) a new window is opened in the same tab indicating that the request for terminology annotation is being processed. The results of terminology annotation for HTML5 documents are visualised in a new window in the same tab (see Figure 14 below). Similarly to the plaintext visualisation, terms are marked in the HTML5 content with orange and purple backgrounds. For more details of the visualisation please refer to section 4.2 (the visualisation differs only in the content visualisation – the functionality is the same).

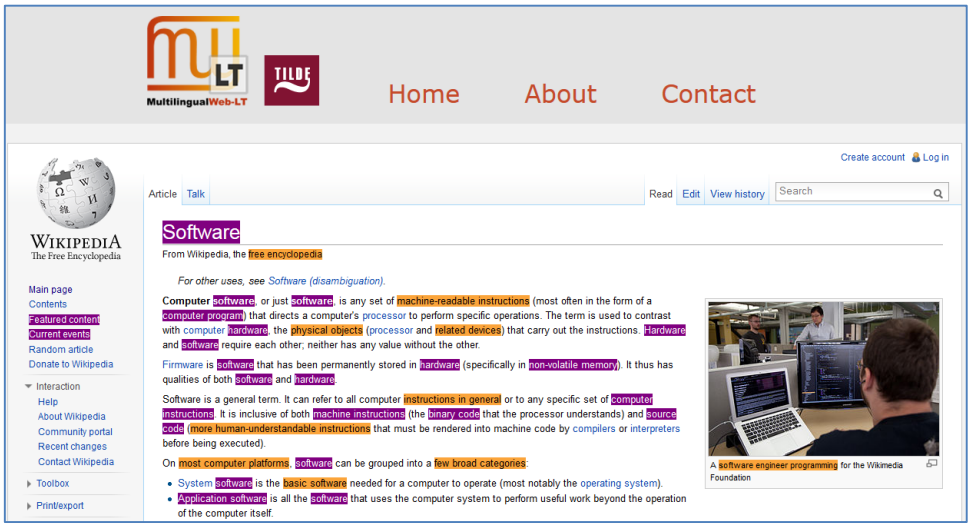

Figure 14. Visualisation example for annotated terminology in HTML content

The HTML content visualisation will show also pre-existing terminology annotation if such will be present in the HTML document submitted for terminology annotation. Pre-existing terms in the content will be marked with a grey background (see Figure 15 below).

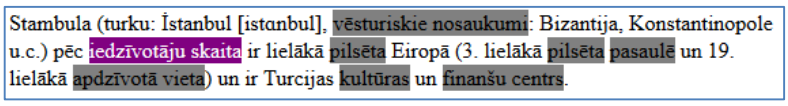

Figure 15. Example of visualisation of pre-existing terminology annotation

4.4. Terminology annotation in XLIFF documents

If the user selects "*XLIFF*" option as input, a new form is opened (see Figure 16 below) where the user has to enter the same configuration options as for the plaintext input (i.e., language, terminology annotation method and domain; see section 4.2 for more details). The showcase supports only XLIFF 1.2 documents (non-standard XLIFF documents may not be supported).

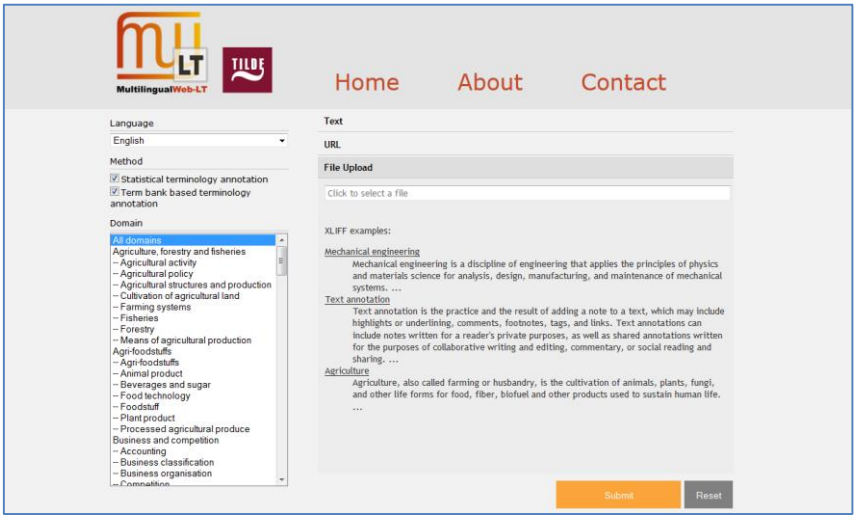

Figure 16. The "XLIFF" input form of the Web-based Showcase

Once the user has selected the appropriate configuration for terminology annotation, the user is offered three options to pass data (XLIFF content) to the system. The user can:

- **Enter (or paste) XLIFF content** directly in the text box under the heading "*Text*".
- **Enter (or paste) a URL** that links to an XLIFF document that is available on the Web under the heading "*URL*". For testing purposes the Web-based showcase offers several XLIFF 1.2 examples that can be used by the user. The examples can be passed to the system by selecting a link from the list. The Web-based showcase will automatically apply a default configuration (i.e., language, annotation method, and domain) for the selected example and fill the XLIFF content in the text box.
- **Select and upload a file containing an XLIFF document** that is located on the user's local computer.

It is mandatory to select at least one data input mechanism (pasting the XLIFF content, entering the URL, or uploading a file), otherwise the system will return an exception saying that: "*No text was submitted.*"

Once all options are selected and the data input mechanism is selected, the user has to click on the "*Submit*" button in order to start terminology annotation. Alternatively, the user can click on the "*Reset*" button in order to reset the form to its initial configuration.

After selecting "*Submit*", similarly to plaintext annotation (see Figure 4 above) a new windows is opened in the same tab indicating that the request for terminology annotation is being processed. The results of terminology annotation for XLIFF documents are visualised in a new window in the same tab (see Figure 17 below). The visualisation for terms annotated in XLIFF content differs from the plaintext and HTML5 visualisation.

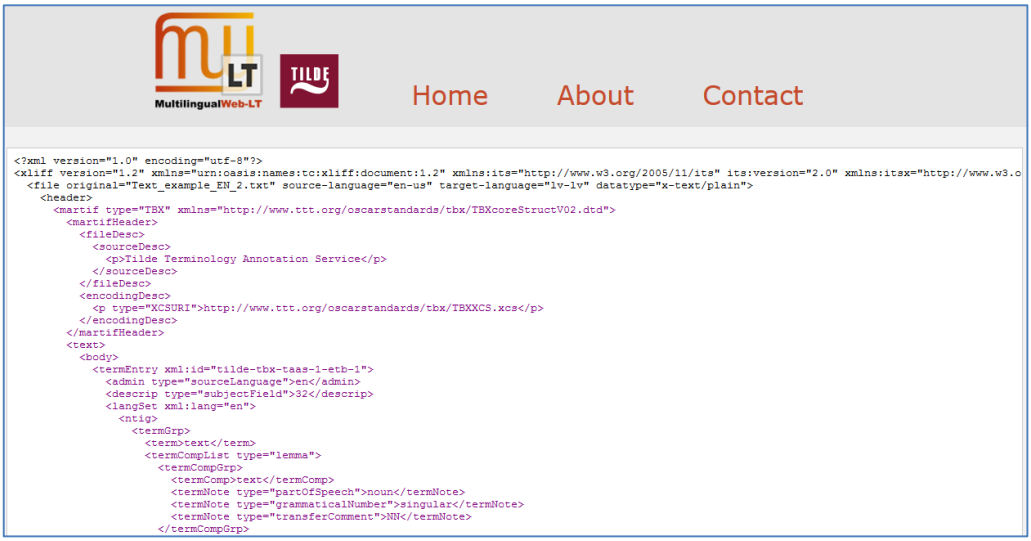

Figure 17. Visualisation example for annotated terminology in XLIFF content

Annotated terminology in XLIFF content is visualised by color-coding the added markup. Term candidates identified by the "*Statistical terminology annotation*" method are marked using an XLIFF `<mrk>` element with the `mtype="term"` and the `itsx:termConfidence="#.##"` attributes (see Figure 17 below).

```
<mrk mtype="term" itsx:termConfidence="0.08">discipline of engineering</mrk>
```

Figure 18. Example of an annotated term candidate using
the "Statistical terminology annotation" method in XLIFF

The terms identified by the "*Term bank based terminology annotation*" method are marked using an XLIFF `<mrk>` element with the `mtype="term"` and the `itsx:termInfoRef="#tilde-tbx-taas-##-etb-##"` attributes (see Figure 19 below). The `itsx:termInfoRef` attribute refers to additional information describing the annotated term. The additional information in the showcase implementation is stored within the XLIFF document itself.

```
<mrk itsx:termInfoRef="#tilde-tbx-taas-1-etb-1" mtype="term">Mechanical engineering</mrk>
```

Figure 19. Example of an annotated term using the "Term bank based terminology annotation" method in XLIFF

The additional information for each term identified in *EuroTermBank* is stored in a TBX document which is embedded in the `<header>` element of each file. An example of a term entry stored in the TBX format and embedded in an XLIFF document is given in Figure 20 below. The example identifies term collections in which the term was found in *EuroTermBank* with the `<xref>` element.

```
<?xml version="1.0" encoding="utf-8"?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2" xmlns:its="http://www.w3.org/2005/11/its"
  its:version="2.0" xmlns:itsx="http://www.w3.org/ns/its-xliff/" its:annotatorsRef="terminology|http://tilde.com/term-annotation-service">
  <file original="Text_example_EN_1.txt" source-language="en-us" target-language="lv-lv" datatype="x-text/plain">
    <header>
      <martif type="TBX" xmlns="http://www.ttt.org/oscarstandards/tbx/TBXcoreStructV02.dtd">
        <martifHeader>
          <fileDesc>
            <sourceDesc>
              <p>Tilde Terminology Annotation Service</p>
            </sourceDesc>
          </fileDesc>
          <encodingDesc>
            <p type="XCSURI">http://www.ttt.org/oscarstandards/tbx/TBXXCS.xcs</p>
          </encodingDesc>
        </martifHeader>
        <text>
          <body>
            <termEntry xml:id="tilde-tbx-taas-14-etb-14">
              <admin type="sourceLanguage">en</admin>
              <descrip type="subjectField">68</descrip>
              <langSet xml:lang="en">
                <ntig>
                  <termGrp>
                    <term>modeling</term>
                    <termCompList type="lemma">
                      <termCompGrp>
                        <termComp>modeling</termComp>
                        <termNote type="partOfSpeech">noun</termNote>
                        <termNote type="grammaticalNumber">singular</termNote>
                        <termNote type="transferComment">NN</termNote>
                      </termCompGrp>
                    </termCompList>
                  </termGrp>
                  <descrip type="reliabilityCode">1</descrip>
                  <admin type="score">0.06</admin>
                  <admin target="92539277-0a11-43d6-9bc5-a45f57a698ee.output.step1.completed" type="sourceIdentifier" />
                  <xref target="http://www.eurotermbank.com/Collection.aspx?collectionid=277" type="xSource">Mehānikas terminoloģijas vārdnīca</xref>
                </ntig>
              </langSet>
            </termEntry>
          </body>
        </text>
      </martif>
    </header>
```

Figure 20. An example of a term entry stored in TBX format embedded in an XLIFF document

In order to return to the input form, similarly to plaintext and HTML5 processing scenarios the user has to follow the "*Go back*" link at the bottom left corner of the results page (see Figure 12 above). Similarly to plaintext and HTML5 processing scenarios the user can download the annotated XLIFF document by clicking on the "*Download*" button that is located in the bottom right corner of the results page (see Figure 8 above).

5. TECHNICAL DOCUMENTATION

The solution is organised into the following three logical projects:

- *Tilde ITS* – a reusable software library that implements the Internationalization Tag Set (ITS) Version 2.0;
- *Tilde TAWS API* – a web service that annotates terminology in the submitted document;
- *Tilde TAWS Showcase* – a user interface for TAWS in the form of a web site.

For ease of deployment, the solution is organised into the following two physical projects:

- *Tilde.Its – Tilde ITS,*
- *Tilde.Taws – Tilde TAWS API* and *Tilde TAWS Showcase.*

The following sections describe each logical project. Other projects and libraries that the mentioned projects depend on are described in another section at the end.

5.1. Tilde ITS

`Tilde.Its` is a .NET class library written in C# for parsing ITS 2.0 annotated content in XML and HTML formats. All functionality is located in the `Tilde.Its` namespace.

5.1.1. ItsDocument

The abstract class `ItsDocument` represents an ITS 2.0 annotated document. It performs the following functionalities:

- loads and parses the ITS 2.0 enriched document;
- finds and loads global rules associated to the document.

The class `ItsXmlDocument`, which inherits from `ItsDocument`, represents an XML document. Similarly, the class `ItsHtmlDocument` represents an HTML document.

Code examples for usage of the `ItsXmlDocument` class are given below.

The `ItsXmlDocument` class requires the declaration of the *Tilde.Its* namespace:

```
using Tilde.Its;
```

A new document can be created in the following ways:

- from a file:

```
ItsXmlDocument doc = new ItsXmlDocument(uri: "document.xml");
```

- from a string:

```
ItsXmlDocument doc = new ItsXmlDocument(xml: "<root/>");
```

If a new document is created from a string and it has external rules with relative paths, the user can specify the base path for these rules:

```
ItsXmlDocument doc = new ItsXmlDocument(
  xml: "<root><its:rules ... xlink:href='rules.xml'/></root>",
  uri: "http://example.com"); // will load rules from
http://example.com/rules.xml
```

A new document can be also created by cloning an existing document:

```
ItsXmlDocument doc2 = new ItsXmlDocument(doc);
```

A document can be converted to an XML document or a string (e.g., to save it in a file):

```
System.Xml.Linq.XDocument xmlDoc = doc.Document;
string xml = doc.Document.ToString();
```

5.1.2. Data categories

Each data category is represented by a class that inherits from `DataCategory`. Class names are formed using the data category name and the suffix `DataCategory`, for example:

`TranslateDataCategory`, `ElementsWithinTextDataCategory`, `MtConfidenceDataCategory`, `IdValueDataCategory`.

The constructor of these classes accepts two arguments: `ItsDocument` which contains the global rules, and the node (element or attribute) to analyse.

```
using Tilde.Its;
using System.Xml.Linq; // XEement, XAttribute
using System.Diagnostics; // Assert

ItsHtmlDocument doc =
  new ItsHtmlDocument("<html><body translate=no>0x01 0x02
0x03</body></html>");
XElement html = doc.Document.Root;
XElement body = html.Element(ItsHtmlDocument.XhtmlNamespace + "body");
XAttribute bodyAttribute = body.Attribute("translate");

TranslateDataCategory htmlTranslate = new TranslateDataCategory(doc, html);
Assert.IsTrue(htmlTranslate.IsTranslatable); // default value

TranslateDataCategory bodyTranslate = new TranslateDataCategory(doc, body);
Assert.IsFalse(bodyTranslate.IsTranslatable); // local value

TranslateDataCategory bodyAttributeTranslate =
```

```
  new TranslateDataCategory(doc, bodyAttribute);
Assert.IsFalse(bodyAttributeTranslate.IsTranslatable); // default value
```

The `annotatorsRef` attribute is not a data category but it can be used in a similar way.

```
AnnotatorAnnotation annotators = new AnnotatorsAnnotation(doc, element);
annotators.AnnotatorRef; // terminology|http://1 text-analysis|http://2
annotators["terminology"]; // AnnotatorsRef with "terminology" and
"http://1"
// annotators is IEnumerable<AnnotatorsRef>
```

5.1.3. Annotations

The class `System.Xml.Linq.XObject` (from which `XElement` and `XAttribute` inherit) supports adding annotations.

```
html.AddAnnotation(new TranslateDataCategory(doc, html));
TranslateDataCategory htmlTranslate =
html.Annotation<TranslateDataCategory>();
```

The class `ItsDocument` takes advantage of this functionality and provides a quick way to annotate all elements and attributes in the document. In order to find all data categories in *Tilde.Its* and annotate all elements and attributes in the document, use:

```
doc.AnnotateAll();
```

In order to find all data categories in *Tilde.Its* and annotate the `html` element and its attributes, use:

```
doc.AnnotateAll(html);
```

In order to annotate the `html` element and its attributes with the *Translate* data category, use:

```
doc.Annotate<TranslateDataCategory>(html);
```

In order to annotate all elements and attributes in the document with the *Translate* data category, use:

```
doc.Annotate<TranslateDataCategory>(doc.Document.Descendants());
```

Before adding an annotation, previous annotations of the same type are removed. That means that the `Annotate*()` methods can be used to re-annotate the content as well.

When an annotation is added, its value is not computed. Its value is computed lazily. For instance, in the following example:

```
string html = "<html><body><p></p></body></html>";
```

We add all data categories to all nodes (no value computations are performed):

```
doc.AnnotateAll();
```

The following *Assert* computes the value for `<body>` (result: *no local, no global rules*), then for `<html>` (result: *no local, no global rules*), then it computes the default value (*true*); `<p>` is not computed as it is not necessary:

```
Assert.IsTrue(body.Annotation<TranslateDataCategory>().IsTranslatable);
```

Once the value has been computed for an annotation, it is cached. In the following code the value for `<body>` has been already computed in the last example (*true*) and the value for `<html>` has also been already computed (*true*).

```
Assert.IsTrue(body.Annotation<TranslateDataCategory>().IsTranslatable);
```

Some of the data categories support inheritance. When a node is annotated and looks for the inherited value, it will use the annotation on the parent element if there is one (if there is no annotation, it is not added). Thus, if the parent elements are already annotated, they do not have to be annotated again, and it improves performance.

5.1.4. Overriding values

You can override the computed values for some data categories.

```
doc.AnnotateAll();
body.Annotation<TranslateDataCategory>().IsTranslatable = false;
// <p>   : false   (will be computed and taken from <body>)
// <body>: false   (already computed = overriden)
// <html>: true    (will be computed)
```

However, if you override a value and the values for children nodes have already been computed, you will get incorrect results.

```
doc.AnnotateAll(); // computes values for all elements
Assert.IsTrue(p.Annotation<TranslateDataCategory>().IsTranslatable);
// override the value
body.Annotation<TranslateDataCategory>().IsTranslatable = false;
// <p>   : true    (already computed in Assert.IsTrue)   <- INCORRECT
// <body>: false   (already computed = overriden)
// <html>: true    (already computed in Assert.IsTrue)
```

When you override a value, you should re-annotate all descendants to avoid such situations.

```
doc.AnnotateAll(); // computes values for all elements
Assert.IsTrue(p.Annotation<TranslateDataCategory>().IsTranslatable);
// override the value
body.Annotation<TranslateDataCategory>().IsTranslatable = false;
// re-annotate descendants
doc.Annotate<TranslateDataCategory>(body.Descendants());
// <p>   : false   (will be computed and taken from <body>)
// <body>: false   (already computed = overriden)
// <html>: true    (already computed in Assert.IsTrue)
```

5.2. Tilde TAWS API

The Terminology Annotation Web Service (TAWS) can annotate terminology in plaintext and in ITS 2.0 enriched HTML5 and XLIFF documents. `Tilde.Taws` is an ASP.NET Web API project written in C#. It exposes a RESTful API over HTTP.

5.2.1. API

TAWS exposes a RESTful API over HTTP.

5.2.1.1. HTML5

Request:

```
POST /api/html5 HTTP/1.1
Host: taws.tilde.com
Content-Length: 62

<!DOCTYPE html><html lang="en"><body>hello world</body></html>
```

Response:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8

<!DOCTYPE html>
<html lang="en">
<body its-annotators-ref="terminology|http://tilde.com/term-annotation-
service">
<span its-term="yes" its-term-confidence="1">hello world</span>
</body></html>
```

5.2.1.2. XLIFF

Request:

```
POST /api/xliff HTTP/1.1
Host: taws.tilde.com
Content-Length: 307

<?xml version="1.0" encoding="utf-8"?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">
<file original="hello.txt" source-language="en-us" target-language="lv-lv"
datatype="plaintext">
<body>
<trans-unit id='1'>
<source>hello world</source>
</trans-unit>
</body>
</file>
</xliff>
```

Response:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8

<?xml version="1.0" encoding="utf-8"?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2"
xmlns:its="http://www.w3.org/2005/11/its"
xmlns:itsx="http://www.w3.org/ns/its-xliff/"
its:annotatorsRef="terminology|http://tilde.com/term-annotation-service">
 <file original="hello.txt" source-language="en-us" datatype="plaintext">
  <body>
   <trans-unit id="1">
      <source><mrk mtype="term" itsx:termConfidence="1">hello
world</mrk></source>
   </trans-unit>
  </body>
 </file>
</xliff>
```

Every single piece of text in a document must have a language identifier or it will not be annotated since the language is *unknown*. You can add a `lang` parameter to the query string to set the default language of the content without modifying the markup.

```
/api/html5?lang=en
```

The *Domain* data category identifies the topic of the document content. If the document contains no domain information, terminology from all domains is annotated. You can optionally add one or more `domain` parameters to the query string to set the default domain(s) of the content without modifying the markup. Each domain must be a *EuroVoc* domain code (two or four digit string). A parent domain (two digit code) includes all child domains.

```
/api/html5?domain=32
/api/html5?domain=66&domain=2441
/api/html5?lang=en&domain=32&domain=40
```

If your document contains references to external rules with relative paths (e.g., `<link rel="its-rules" href="rules.xml">`), you can add a `baseUri` parameter specifying an accessible base path (e.g., `http://example.org/its/`), otherwise the rules cannot be loaded by TAWS.

```
/api/html5?baseUri=http://example.org/its/
/api/html5?lang=en&baseUri=http://example.org/its/
```

5.2.1.3. Plaintext

For convenience, it is possible to annotate terminology in plaintext documents as well.

Request:

```
POST /api/plaintext?lang=en HTTP/1.1
Host: taws.tilde.com
Content-Length: 11

hello world
```

Response:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
</head>
<body its-annotators-ref="terminology|http://tilde.com/term-annotation-
service">
  <span its-term="yes" its-term-confidence="1">hello world</span>
</body>
</html>
```

The text will be converted and returned as an HTML5 document since there is no standard way to annotate terminology in plaintext using ITS 2.0 metadata.

Because there is no ITS 2.0 markup present in plaintext content, the `lang` parameter is mandatory for plaintext documents. You can optionally add one or more `domain` parameters to specify the domain of the text.

5.2.1.4. Terminology Annotation Methods

By default, terminology is annotated using both the "*Statistical terminology annotation*" (`statistical`) and the "*Term bank based terminology annotation*" (`termbank`) method.

To use only one method for annotation of terminology, specify it with a `method` parameter in the query string:

```
/api/html5?method=statistical
/api/html5?method=termbank
/api/html5?lang=en&domain=32&method=termbank
```

Note that the *Domain* data category is ignored if the "Term bank based terminology annotation" (`termbank`) method is not used.

5.2.1.5. HTTP Status Codes

TAWS will respond with one of the following status codes:

- *200 OK* – document was annotated successfully;

- *400 Bad Request* – invalid document or parameters passed to TAWS;
- *500 Internal Server Error* – there is a problem with the service.

The content of the response will be the annotated document or an error message in case of an error.

An example of a response indicating an error is as follows:

```
HTTP/1.1 400 Bad Request
Content-Type: text/plain; charset=utf-8

Invalid root element.
```

5.2.1.6. Limitations

- Only input in UTF-8 is supported.
- Only the first 50 000 characters of the submitted document will be annotated. The remaining document will be returned to the user without annotated terminology. This is a limitation for showcase purposes in order not to allow misuse of the service.
- Domain values are limited to *EuroVoc* codes. The domains and their codes can be found on the *EuroVoc* website[7]. *EuroVoc Edition 4.4* is used. More information about *EuroVoc* classification can be found on the *EuroVoc* website[8].

5.2.2. API Implementation

This section describes the organisation of the code, responsibilities of classes and the work of software.

5.2.2.1. Requests

The implementation uses the Model-View-Controller (MVC) design pattern. Routes describe how requests should be mapped to controllers. The front controller (provided by the framework) uses the routes to determine which Controller to use. The Controller creates and operates Models that do the work and generates the output (or View) which is returned as the response.

When the application is started for the first time, the method `WebApiApplication.Application_Start` (in `Global.asax`) is called. The method

[7] http://eurovoc.europa.eu/drupal/?q=navigation&cl=en

[8] http://eurovoc.europa.eu/

configures the Web API by calling the methods in the static `WebApiConfig` class. One of these methods, `WebApiConfig.RegisterRoutes`, tells the server how to route incoming requests:

```
/api/plaintext -> Tilde.Taws.Controllers.ApiController.Plaintext
/api/html5     -> Tilde.Taws.Controllers.ApiController.Html5
/api/xliff     -> Tilde.Taws.Controllers.ApiController.Xliff
All routes accept only POST requests.
```

When `/api/html5` is requested, the method `ApiController.ModelBinder.BindModel()` is called. The method creates a new `ApiDocument` instance, which holds all current request parameters. Parameters at this stage are not validated and no error/warning message is given about any parameters which are not recognised or used.

Then the `ApiController.Html5(ApiDocument)` method is called with an `ApiDocument` as the only parameter.

In the controller method, a new instance of the `Html5Annotator` class is created and the `ApiDocument` is passed as a parameter. The constructor will validate the document and throw an exception if it is invalid. Then the `Html5Annotator.Annotate()` method, which performs the annotation, is called asynchronously.

The return value of the `Html5Annotator.Annotate()` method is returned as the response to the client.

`Plaintext` and `Xliff` methods work similarly to the `Html5` method.

5.2.2.2. Annotators

Annotators parse the submitted document, annotate terms in the document and then return the document. To accomplish this, annotators perform a number of steps.

Annotators take the submitted document as a parameter in the constructor where it is parsed and ITS 2.0 rules loaded. When an annotation is requested, the content of the document is split into chunks and these chunks are sent to the *Terminology as a Service (TaaS)* API which finds and annotates terms in these chunks. Annotated chunks are then merged back in the document, post-processed to make sure there is no invalid markup and finally additional information is added to the document to indicate that the document has been processed. Lastly, the document is returned.

The abstract `Annotator<T>` class contains most of the functionality while inherited classes add implementation specific functionality and define the order of execution.

5.2.2.3. PlaintextAnnotator

It is not really possible to visualise terminology in plaintext. Therefore, the visualisation of terminology in plaintext is done in HTML5. Since it is possible for an HTML5 document to look exactly like a plaintext document, a plaintext document is converted to an HTML5 document and then annotated using the `Html5Annotator` to avoid duplicating functionality.
Before conversion, HTML entities are escaped and new lines are preserved by using the `<br>` HTML tag. The processed text is placed in the `<body>` tag and there is a tag identifying the text encoding (`<meta charset="utf-8">`) added to the `<head>` element.

5.2.2.4. Html5Annotator

The `Html5Annotator` annotates terminology in HTML5 documents. XHTML and older versions of HTML may also be processed (however, they are not officially supported). Invalid markup is tolerated since we use a forgiving HTML5 parser (*HtmlParserSharp*, see the section about dependencies).
In the constructor, an instance of the class `Tilde.Its.ItsHtmlDocument` is created, which parses the submitted document (which is a string at this point) and loads ITS 2.0 rules.
`Annotate()` is an asynchronous method which performs the annotation. All annotation is performed on elements in the `<body>` element.

Default values

If `<body>` has the default *Language Information* data category value (i.e., there is no explicit language defined within the ITS 2.0 enriched content), the submitted language is used as the `<body>` language (and it will be inherited by all child nodes that have no language defined). Similarly, if `<body>` has the default *Domain* data category value (i.e., no domain metadata specified in the ITS 2.0 enriched content), the submitted domains are used (if there are any) and these will be also inherited by child nodes.

Chunking

The content of `<body>` is split into chunks by `Html5Chunker` (see section 5.2.2.6).

Chunks are sent to TaaS

After chunking, the chunks are sent to the *TaaS API* asynchronously and in parallel. In parallel means that all chunks are sent at the same time and responses can also be processed at the same time. Asynchronously means that the threads will not block and are able to perform other tasks while the I/O operation is processing.

Integration of chunks within the document

Annotated chunks are received from the API and integrated back in the document. At this point, terms in the document are represented by the `Tename` class (which inherits from the `XElement` class) in the document XML tree and appear as `<tename xmlns="">term</tename>` in XML.

Removal of duplicates

If a piece of text was in two domains (e.g., law and finance) and there was a word in the text that was a term in both domains, the word would be annotated twice. `Annotator<T>.MergeTenames()` merges such terms into one by averaging the confidence value and preserving term information (TBX).

Removal of invalid annotations

The *Terminology* data category does not support inheritance, and thus some annotations that are not valid (e.g., overlapping annotations) are removed. For instance, `<tename>terminology <span>annotation</span></tename>` is not valid and the annotation will be removed.

Removal of existing terms

If a term was already annotated in the original document with either local or global markup, the added annotation is removed in order not to conflict with the existing annotation.

To identify the annotation service used for annotation, the `its-annotators-ref` attribute is added to the `<body>` element if there are no terms in the document that were annotated by another annotator).

Replacement

All remaining term annotations (which are still `Tename` instances) are renamed to HTML tags. If `its-annotators-ref` for terminology is not inherited (i.e. it was not added to the `<body>` element), it is added as a local attribute.

```
<span its-term="yes" its-term-confidence="0.5">term</span>
```

For each term that has additional terminological information provided by the TaaS terminology annotation service in a *TBX* format, this information is added in a `<script>` tag to the `<head>` element and identified by a unique ID based on the term ID(s) (prefix with `tilde-tbx`). This ID is then added as an `its-term-info-ref` reference to each term.

```
<span its-term="yes" its-term-info-ref="#tilde-tbx-1" ...>
```

TBX is an XML document. Because terms could have been merged, each TBX can contain several entries.

```
<script type="text/xml" id="tilde-tbx-1">
<?xml version='1.0'?>
<!DOCTYPE martif SYSTEM "TBXcoreStructV02.dtd">
<martif type="TBX">
  <martifHeader>
    <fileDesc>
      <sourceDesc>
        <p>Tilde Terminology Annotation Service</p>
      </sourceDesc>
    </fileDesc>
    <encodingDesc>
      <p type="XCSURI">http://www.ttt.org/oscarstandards/tbx/TBXXCS.xcs</p>
    </encodingDesc>
  </martifHeader>
  <text>
    <body>
     <termEntry>...</termEntry>
     <termEntry>...</termEntry>
   </body>
  </text>
</martif>
</script>
```

Finally, the underlying XML document is converted to an HTML5 document (e.g., `<script/>` is invalid, it must be `<script></script>`) and returned as a string.

5.2.2.5. XliffAnnotator

`XliffAnnotator` works similarly to `Html5Annotator`.

The `XliffAnnotator` annotates terminology in XLIFF 1.0, 1.1 and 1.2 documents.

In the constructor, an instance of the class `Tilde.Its.ItsXmlDocument` is created, which parses the submitted document. At this point the XML document is validated to make sure the document is in an XLIFF namespace and the root element is `<xliff>` and an exception is thrown if it is not. Before loading ITS 2.0 rules, the following custom rules are transparently inserted into the document (which can be overriden by the user defined ITS 2.0 rules). These rules are based on the information provided in the XLIFF Mapping wiki[9].

```
<its:rules version='2.0'
  xmlns:its='http://www.w3.org/2005/11/its'
  xmlns:itsx='http://www.w3.org/ns/its-xliff/'
  xmlns:xliff11='urn:oasis:names:tc:xliff:document:1.1'
  xmlns:xliff12='urn:oasis:names:tc:xliff:document:1.2'>

  <its:termRule selector='/xliff//mrk[@mtype=""term""]' term=""yes"" />
  <its:termRule selector='//xliff11:mrk[@mtype=""term""]' term=""yes"" />
  <its:termRule selector='//xliff12:mrk[@mtype=""term""]' term=""yes"" />
```

[9] http://www.w3.org/International/its/wiki/XLIFF_Mapping

```
  <its:termRule selector='/xliff//mrk[@mtype=""x-its-term-no""]'
                term=""no"" />
  <its:termRule selector='//xliff11:mrk[@mtype=""x-its-term-no""]'
                term=""no"" />
  <its:termRule selector='//xliff12:mrk[@mtype=""x-its-term-no""]'
                term=""no"" />

  <its:domainRule selector='//*' domainPointer='@itsx:domains'/>

  <its:withinTextRule withinText='yes' selector='/xliff//g | /xliff//x |
/xliff//bx | /xliff//ex | /xliff//bpt | /xliff//ept | /xliff//it |
/xliff//ph | /xliff//mrk'/>
  <its:withinTextRule withinText='yes' selector='//xliff11:g | //xliff11:x
| //xliff11:bx | //xliff11:ex | //xliff11:bpt | //xliff11:ept |
//xliff11:it | //xliff11:ph | //xliff11:mrk'/>
  <its:withinTextRule withinText='yes' selector='//xliff12:g | //xliff12:x
| //xliff12:bx | //xliff12:ex | //xliff12:bpt | //xliff12:ept |
//xliff12:it | //xliff12:ph | //xliff12:mrk'/>

  <its:withinTextRule withinText='nested' selector='/xliff//sub'/>
  <its:withinTextRule withinText='nested' selector='//xliff11:sub'/>
  <its:withinTextRule withinText='nested' selector='//xliff12:sub'/>

</its:rules>
```

`Annotate()` is an asynchronous method which performs the annotation. All annotation is performed only on `<source>`, `<seg-source>` and `<target>` elements.

Default values

If the XLIFF document has missing language information or has the default *Language Information* data category value (i.e., there is no explicit language defined within the ITS 2.0 enriched content), the submitted language is used as the source language of the document (and it will be inherited by all child nodes that have no language defined). Similarly, if `<xliff>` has the default *Domain* data category value (i.e., no domain metadata specified in the ITS 2.0 enriched content), the submitted domains are used (if there are any) and these will be also inherited by child nodes.

In XLIFF 1.2, only the following elements are allowed to have an `xml:lang` attribute: `<xliff>`, `<note>`, `<prop>`, `<source>`, `<target>`, `<alt-trans>`. If any of these elements has this attribute defined, as per ITS 2.0 rules, it will used as the value for the *Languge Information* data category and thus inherited by child elements. This is why the value of the `xml:lang` attribute is ignored on other elements (i.e. not `<xliff>`, `<note>`, `<prop>`, etc.) outside `<source>`, `<source-seg>` and `<target>`.

Chunking

The content of `<source>`, `<seg-source>` and `<target>` elements is split into chunks by `XliffChunker` (see section 5.2.2.6).

Chunks are sent to TaaS

After chunking, the chunks are sent to the *TaaS API* asynchronously and in parallel. In parallel means that all chunks are sent at the same time and responses can also be processed at the same time. Asynchronously means that the threads will not block and are able to perform other tasks while the I/O operation is processing.

Integration of chunks within the document

Annotated chunks are received from the API and integrated back in the document. At this point, terms in the document are represented by the `Tename` class (which inherits from the `XElement` class) in the document XML tree and appear as `<tename xmlns="">term</tename>` in XML.

Removal of duplicates

If a piece of text was in two domains (e.g., law and finance) and there was a word in the text that was a term in both domains, the word would be annotated twice. `Annotator<T>.MergeTenames()` merges such terms into one by averaging the confidence value and preserving term information (TBX).

Removal of invalid annotations

The *Terminology* data category does not support inheritance, and thus some annotations that are not valid (e.g., overlapping annotations) are removed. For instance, `<tename>terminology <g id='1'>annotation</g></tename>` is not valid and the annotation will be removed.

Removal of existing terms

If a term was already annotated in the original document with either local or global markup, the added annotation is removed in order not to conflict with the existing annotation.

To identify the annotation service used for annotation, the `its:annotatorsRef` attribute is added to the `<xliff>` element if there are no terms in the document that were annotated by another annotator).

If the ITS 2.0 namespace has not been defined in the document it is added with the prefix `its`. If this prefix is already used by another namespace, other prefixes (such as `its2`, `its3` etc.) are used. Similary, the XLIFF Mapping[10] namespace is defined with the prefix `itsx`.

Replacement

[10] http://www.w3.org/ns/its-xliff/

All remaining term annotations (which are still `Tename` instances) are renamed to XLIFF tags. If `its:annotatorsRef` for terminology is not inherited (i.e. it was not added to the `<xliff>` element), it is added as a local attribute.

```
<mrk mtype="term" itsx:termConfidence="0.5">term</mrk>
```

For each term that has additional terminological information provided by the TaaS terminology annotation service in a *TBX* format, this information is added in a `<martif>` tag to the `<header>` element of each file and each term in it is identified by a unique ID based on the term ID(s) (prefix with `tilde-tbx`). This ID is then added as an `itsx:termInfoRef` reference to each term.

```
<mrk mtype="term" itsx:termInfoRef="#tilde-tbx-1" ...>
```

TBX is an XML document embedded in the `header` element.

```
<xliff ...>
<file original='tbx.txt' source-language='en'>
  <header>
   <martif type='TBX'
xmlns='http://www.ttt.org/oscarstandards/tbx/TBXcoreStructV02.dtd'>
      <martifHeader>
       <fileDesc>
        <sourceDesc>
         <p>Tilde Terminology Annotation Service</p>
        </sourceDesc>
       </fileDesc>
       <encodingDesc>
        <p
type='XCSURI'>http://www.ttt.org/oscarstandards/tbx/TBXXCS.xcs</p>
       </encodingDesc>
      </martifHeader>
      <text>
       <body>
        <termEntry xml:id='tilde-tbx-etb-1'>...</termEntry>
       </body>
      </text>
   </martif>
  </header>
...
```

Finally, the underlying XML document is returned as a string.

5.2.2.6. Chunkers

Chunkers are classes that split the content of a document into smaller fragments called chunks. The text in each chunk has the same properties (i.e., the same language, the same domain). A fragment of a text can be in two chunks (e.g., a paragraph can be in two domains like finance and law) at the same time.

Chunking is necessary for the TaaS terminology annotation service as it is designed to process text of one language and one domain at a time (see 5.2.2.7).

Chunkers use the *Language Information*, *Domain*, *Elements Within Text* and *Locale Filter* data categories, however ignore the *Terminology* data category, which is taken into account by annotators.

"*Split*" and "*splitting*" refer to splitting the document into chunks, whereas "*join*" and "*joining*" refer to the opposite of splitting.

Chunkers operate according to the following algorithm:

- Find all unique languages in the document (e.g., *<none>*, *en*, *en-US*, *lv*).
- Find all unique domains in the document (e.g., *<none>*, *auto*, *finance*, *law*).
- For each language/domain pair, find elements with independent text flow (*Elements Within Text* is not equal to "*yes*"), ignoring some elements (e.g., *<script>* in HTML5, everything but *<source>*, *<target>* in XLIFF). The content of these elements is in the selected language and the content is also in the selected domain.
- For each independent element, extract text from it:
    - If it is an ignored element, return nothing at all;
    - If it is a whitespace element (e.g., *
* in HTML5) and it is not the first node, represent it with a space;
    - If it is in another language or domain, ignore the text;
    - If it is excluded by the *Locale Filter* data category, ignore the text;
    - Otherwise include the node text;
    - If the text is followed by another independent text flow, add a break;
    - Repeat the process with all the children.

Example in HTML5:

```
<body lang='en'>
  <p lang='lv'>Sveika, pasaule</p>
  <div lang='en'>Hello, <strong>world</strong> <div>:)</div>!</div>
  <p lang='en-US' domains='law, finance'>money laundering</p>
  <p its-locale-filter-list='lv'>High-five!</p>
  <script>...</script>
</body>

Languages: en, lv, en-us
Domains: <none>, law, finance

Language: en, Domain: <none>
Independent text flows:
   <body lang='en'>
   (<p lang='lv'> is in another language)
   <div lang='en'>
   <div>
   (<p lang='en-US' is in another language and domain)
   <p its-locale-filter-list='lv'>
```

```
Extracted text:
   <body lang='en'>: (nothing because it contains only child elements)
   <div lang='en'>: Hello, world <break>! (<break> because of another
independent element)
   <div>: :)
   <p its-locale-filter-list='lv'>: (nothing because the content applies
only to 'lv')

Language: lv, Domain: <none>
Independent text flows: <p lang='lv'> (not <p its-locale-filter-list='lv'>
because the language is 'en', inherited from <body>)
Extracted text: <p lang='lv'>: Sveika, pasaule

Language: lv, Domain: finance
Independent text flows: none

Language: en, Domain: finance
Independent text flows: none

Language: en-us, Domain: finance
Independent text flows: <p lang='en-US' domains='law, finance'>
Extracted text: money laundering

Language: en-us, Domain: law
Independent text flows: <p lang='en-US' domains='law, finance'>
Extracted text: money laundering
```

The extracted text is then sent to the TaaS terminology annotation service, which adds terminology annotation to the text but does not change anything else.

5.2.2.7. The TaaS terminology annotation service

Terminology as a Service (TaaS) exposes a RESTful API over HTTP that takes a text, its language and domain and which terminology annotation method to use as parameters, and annotates terminology in the text according to the parameters. At the time of writing, the TaaS terminology annotation service can annotate texts in English, Latvian and Lithuanian.

The TaaS terminology annotation service can annotate terminology using three methods:

- Method1: statistical;
- Method2: term bank based;
- Method3: both statistical and term bank based.

The *Domain* values have to be valid *EuroVoc* codes (two or four letter digit strings). Parent domains (two letter codes) include all their child domains (four letter codes). An empty domain means that terms from all domains should be annotated. Domains are only taken into account when the *Term bank based terminology annotation* method is used.

The TaaS terminology annotation service returns the same text with terms enclosed in XML tags, although the text itself is not an XML document.

```
This is a <TENAME SCORE="1.0" LEMMA="term" MSD="NN">term</TENAME>.
```

If the *Term bank based terminology annotation* method is used, annotated terms may contain a `termID` attribute and a corresponding *termEntry* (in the *TBX* format) containing additional terminological information.

```
This is a <TENAME termID="etb-1" SCORE="1.0" LEMMA="term"
MSD="NN">term</TENAME>.

<martif type="TBX"
xmlns='http://www.ttt.org/oscarstandards/tbx/TBXcoreStructV02.dtd'>
<martifHeader>
  <fileDesc>
      <sourceDesc>
        <p>Tilde Terminology Annotation Service</p>
      </sourceDesc>
  </fileDesc>
  <encodingDesc>
      <p type="XCSURI">http://www.ttt.org/oscarstandards/tbx/TBXXCS.xcs</p>
  </encodingDesc>
</martifHeader>
<text>
  <body>

<termEntry id="etb-1">
  <admin type="sourceLanguage">en</admin>
  <descrip type="subjectField">04</descrip>
  <langSet xml:lang="en">
  <ntig>
      <termGrp>
        <term>Translation</term>
        <termCompList type="lemma">
            <termCompGrp>
              <termComp>translation</termComp>
              <termNote type="partOfSpeech">noun</termNote>
              <termNote type="grammaticalNumber">singular</termNote>
            </termCompGrp>
        </termCompList>
      </termGrp>
      <descrip type="reliabilityCode">1</descrip>
      <admin type="score">0.24</admin>
      <xref
target="http://www.eurotermbank.com/Collection.aspx?collectionid=382"
type="xSource">Eesti Õigustõlke Keskuse terminibaas ESTERM</xref>
  </ntig>
  </langSet>
</termEntry>

  </body>
 </text>
</martif>
```

The class `TaaS` is the TaaS terminology annotation service client.

It can be configured with two properties:

- `UseStatisticalExtraction` (boolean),
- `UseTermBankExtraction` (boolean).

To perform annotation, the method `async Task<string[][]> Annotate(string language, string domain, string[][] text)` is used.

The first dimension of `string[][] text` defines paragraphs. They are independent units of text. The second dimension of the array defines the text fragments.

The following HTML5 segment

```
<p>Hello <span>world</span>!</p>
<div>Beginning <div>middle</div> end</div>
```

is represented as:

```
new[] {
  // span's Elements Within Text data category value is Yes
  new[] { "Hello world!" },
  // <div> is an independent element, so "Beginning end" cannot be a term
  new[] { "Beginning ", " end" },
  // <div> like <p> has an independent text flow
  new[] { "middle" }
}
```

The two-dimensional array is transformed into a string:

```
Hello world!
 ... ...
Beginning
 .. ..
 end
 ... ...
middle
```

Where `\n ... ... \n` are paragraph boundaries and `\n .. .. \n` are fragment boundaries.

If the term information along with annotations is returned, it is stored. As a text fragment can be annotated in two different domains, the IDs of the returned terms conflict due to multiple calls to the terminology annotation service. Therefore, the IDs are suffixed with the ID (from 1 to N) of the call to the terminology annotation service.

```
domain: law      etb-1 => taas-1-etb-1
domain: finance  etb-1 => taas-2-etb-1
```

The additional term information can be retrieved with `XElement GetTermEntry(string termID)`.

5.2.2.8. TaaS Configuration

The TaaS terminology annotation service is an external Web service and not in the scope of this project. The TaaS terminology annotation service is developed as part of the TaaS project (see http://taas-project.eu for more details). The credentials to access the TaaS terminology annotation service are not provided in the source code. For more details on how to access the

TaaS terminology annotation service, please refer to the TaaS project or contact Tilde[11]. The access details can be changed in the `Web.config` file.

```
<configuration>
  <appSettings>
    <add key="TaaS_Username" value="..." />
    <add key="TaaS_Password" value="..." />
    <add key="TaaS_Server" value="http://..." />
    <add key="TaaS_Timeout" value="00:10:00" />
    <add key="TaaS_MaxLength" value="50000" /> <!-- Max number of
characters to send to TaaS. If text length exceeds this number, the
remaining part will not be sent to TaaS and will not be annotated. -->
  </appSettings>
</configuration>
```

5.2.2.9. Other dependencies

This section lists software libraries that are used in the project.

Library	**URL**	**Licence**
Json.NET (Newtonsoft.Json)	http://json.codeplex.com/	MIT Licence
Microsoft ASP.NET MVC Framework	http://aspnetwebstack.codeplex.com/	Apache 2.0 Licence
HtmlParserSharp	https://github.com/jamietre/HtmlParserSharp	implied MIT Licence

[11] http://taws.tilde.com/contact