



Published on *Multilingual Online Translation* (<http://www.molto-project.eu>)

D3.2 MOLTO translation tools prototype

Contract No.:	FP7-ICT-247914
Project full title:	MOLTO - Multilingual Online Translation
Deliverable:	D3.2 MOLTO translation tools prototype
Security (distribution level):	Public
Contractual date of delivery:	M24
Actual date of delivery:	March 2012
Type:	Prototype
Status & version:	Final
Author(s):	Lauri Carlson
Task responsible:	UHEL
Other contributors:	Thomas Hallgren, Krasimir Angelov, Seppo Nyrkkö, Lauri Alanko, Chunxiang Li, Inari Listenmaa

Abstract

Deliverable D3.2 consists of a prototype of the MOLTO translation tools, documentation of the translation scenario and instructions on the download and installation of the prototype.

1. Introduction

MOLTO promises a translation tool based on Grammatical Framework, a programming language for multilingual grammars. The Grammatical Framework (GF) is used in the MOLTO project to build translation systems for EU languages. The user of the MOLTO translation tool need not know how to write GF grammars. She is supposed to use domain specific grammars developed by others to translate documents in the domains covered by the grammars. GF resource grammars offer basic grammar coverage in dozens of languages. A domain specialist is supposed to write an abstract grammar for a domain based on an ontology of the domain that provides the key concepts and their relationships. Language specific grammar engineers are supposed to map the common abstract grammar to the different resource grammars. Basic domain language and coverage does not guarantee that all terms and idioms found in a translatable document are covered. To be really usable, the MOLTO translation tool should handle lexical gaps in a way that benefits and benefits from a wider community of translators. It should also provide fallback solutions when a text is not covered by the available grammar(s).

This document builds on the Translation Tools API document, which lays out the translation scenario/s addressed by the prototype and describes the various programming APIs available for building the prototype. This document explains the installation, use, and limitations of the MOLTO translation tool prototype. The prototype integrates some but not yet all aspects of the MOLTO translation tools design in the TT API document. As in the API document, we single out a set of core tools for a standalone translator used by one translator, from an extended set of tools that are designed to support MOLTO translation communities.

The core MOLTO Translation Tool (TT) consists of these parts.

- translation editor
- grammar manager
- document manager
- term manager

The core TT editor can be used standalone. It is being integrated, though in the prototype deliverable, only just embedded in the rest of MOLTO translation and ontology/terminology maintenance tools (the extended prototype), in particular, the GlobalSight TMS and the TermFactory term ontology management. Eventually, they both shall play together with grammar GF maintenance and development machinery. To help orientation, we recapitulate the intended workflow from the Translation Tools API document.

1.1 The MOLTO Translation Workflow

The MOLTO TT (translation tools) editor supports a one-person workflow where the same person is the author(ised editor) of the source and the translator. The adoption of the GlobalSight TMS to MOLTO allows embedding it in a more collaborative scenario where more actors are involved as in the professional workflow described in the API document, by adding traditional CAT and translation project support tools to the toolkit. A more difficult part is to adjust the workflow so that the adaptivity goal is satisfied. In the professional workflow, corrected translations accumulate in the translation memory, which helps

translators avoid the same errors next time. In the MOLTO workflow, GF has an active role in generating translations, so it is GF that should learn from the corrections. Concretely, when a translator or reviser changes a wording, the correction should not go unnoticed, but should find its way to back to GF, preferably through a round of community checks. More generally improvements should be shared by the community, so that the whole community acts adaptively.

We next try a description of one round of the ideal MOLTO translation scenario.

Although it is possible that an author is ready to create and translate in one go (especially in a hurry), it is more normal to have some document(s) to start from. The document/s might be created in a GF constrained language editor in the first place. In that case, the only remaining step is translation. If translation coverage and quality has been checked, nothing more is needed. But frequently, some changes are needed to a previously translated document, or a new one is to be created from existing pieces and some new material. Imaginably, some of the parts come from different domains, and need to be processed with different grammars. Some such complications might be handled with document composition techniques in the manner of Docbook or DITA toolchains.

The strength of GF is that it ought to handle grammatical variation of existing sources well, so as to avoid manual patching of previous translations. Assume there is a previously GF translated document, and we want to produce a variant. Then it ought to be enough to load the document, make desired changes to it under the control of the GF grammar, and let GF generate the modified translations.

Is it necessary to show the translations to the user? Not unless the translator knows the target language(s). We should distinguish two profiles: blind translation, where the author does not know or is not responsible for the target languages herself, but relies on outside revision, and plain translation, in which there is one or two target language known to the author/translator to translate to, who wants to check the translations as she goes.

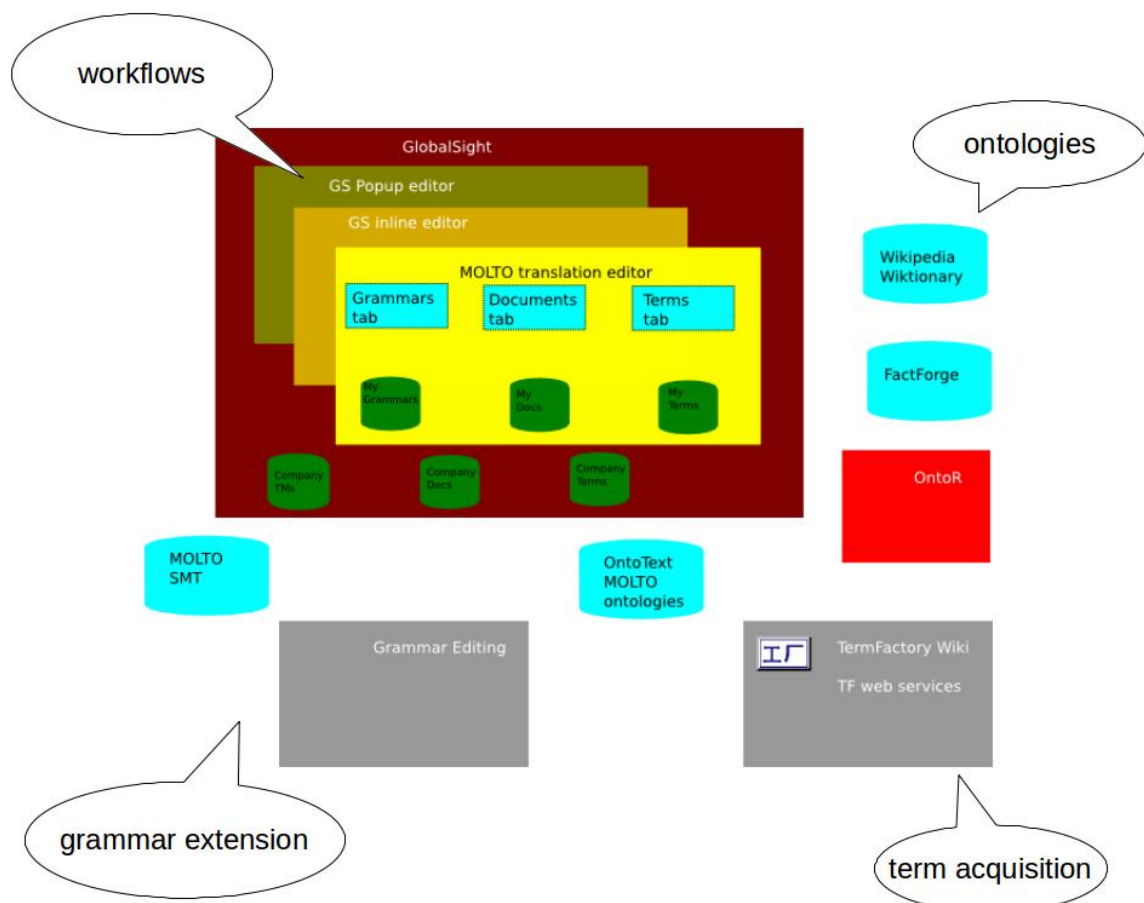
In the blind profile, the author has to rely on revisers, and the revision cycle is slower. The revisers can either notify the author that the source does not translate correctly in their language(s), or they may notify the grammar/lexicon developer(s) directly, or both. If there is a hurry, the reviser/s should provide a correct translation directly for the author/publisher to use as canned text. In addition, they should notify the grammar developer/s of the revisions needed to GF. The notification/s could happen through messages, or conveyed through a shared translation memory, or both. In this slower cycle, it may not be realistic to expect the author to change the source text and repeat the revision process many times over for the same source and possibly a multiplicity of languages to get everything translate right before publication.

In the plain profile, a faster cycle of revision is called for. The author/translator can try a few variations of the input. If no variant seems to work, then she probably wants to use her own translation, but also to make sure that GF learns of and from the failure. The failure can be a personal preference, or a general fix that the community should profit from. If it is a personal preference, the user may want to save the corrected translation in her translation memory and/or glossary, but also she may want to tweak her GF grammar to handle this and similar cases to her liking next time. If it is just a lexical gap or missing fixed idiom, then there should be in GF translation API a service to modify the grammar without knowing GF. The

modifications could happen at different levels of commitment. The most direct one would be to provide a modular PGF format which would allow advising the compiled user grammar on the fly. Such a runtime fix would make sure that the same error will not happen during the same translation session or subsequent ones at least until the domain grammar is recompiled.

The next level of commitment to a change would be to generate new GF source, possibly from example translations provided by the author/translator, compile them, and add the changed or extra modules to the user's GF grammar. The cycle involved here might be too slow to do during translation, but it could happen between translation sessions. If fully automatic grammar revision is too error prone, the author/translator could just go on with canned translations in this session, and commit change requests to the grammar developer community. In this case, the changes would be carried out in good time, with regression tests and multilingual revision cycles, especially if the changes affect the domain semantics (abstract grammar) and thereby all translation directions.

Here is a figure of the overall design.



2. The MOLTO Translation Tools Architecture

The MOLTO Translation Tools architecture is recapitulated here briefly. It consists of many largely independent components. There is a core basically answering the needs of a single author/translator, and an Extended API addressing the needs of a community of authors, translators, and grammar engineers.

The components of the MOLTO TT editor prototype currently include the following:

1. sign in
2. grammar manager
3. document manager
4. term manager
5. translation editor

The components of the MOLTO TT extended prototype include the following:

1. user management
2. grammar management
3. document management
4. lexical resources
5. translation editing
6. translation memory
7. reviewing/feedback
8. grammar engineering

The first five are extensions of the corresponding facilities in the core. The lexical resources API borrows from TermFactory. The translation memory and the reviewing/commenting facilities are adapted from GlobalSight. The last item is based on the GF grammar development tools API.

3. The MOLTO Translation Tools Prototype

This section describes the code constituting the prototype. The code base of the translation tools extended prototype currently consists of the following parts.

- MOLTO TT editor
- GlobalSight (adapted for MOLTO)
- TermFactory (adapted for MOLTO)
- OntoText API
- MOLTO Grammar Tools API

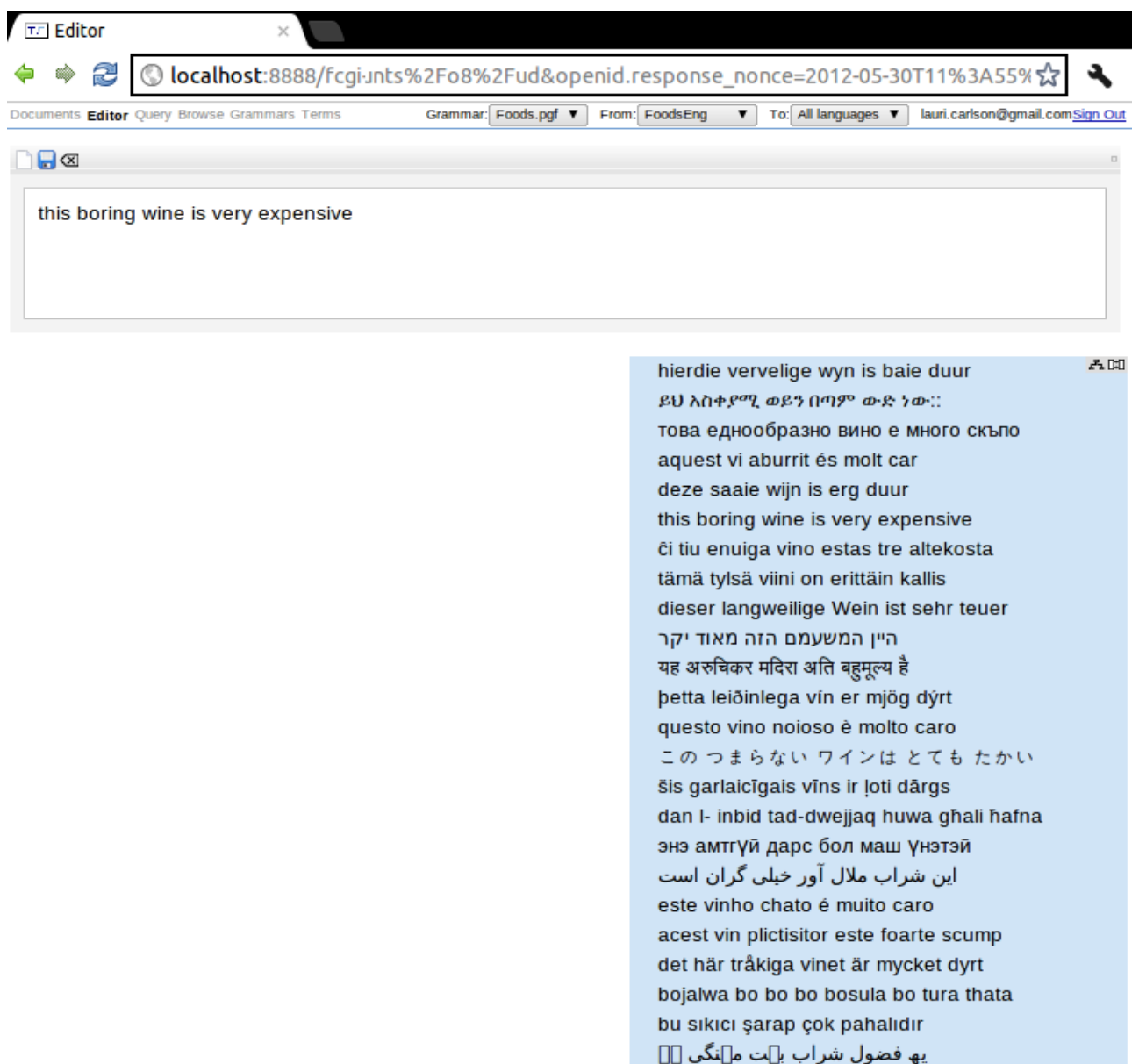
This document describes the prototype's software packages, their installation, use and current limitations. The last two components are not discussed further in this document, because they are described in other MOLTO deliverables. The services currently provided by the GF server are outlined in the MOLTO Grammar Tools API document. The GlobalSight WS API was described in the MOLTO Translation Tools API document. TermFactory is documented at length in the TermFactory manual at

http://www.helsinki.fi/~lcarlson/CF/TF/doc/TFManual_en.xhtml .

3.1 The MOLTO Translation Tools (TT) Editor

This section describes the GF translation editor originally developed by Bringert and Angelov at UGOT and reworked at UHEL.

To guide the development of a suitable translation editor API to support MOLTO translation needs, UGOT created a prototype web-based translation editor. It is implemented using the Google Web Toolkit and usable for authoring with small multilingual grammars. To use it from the web, all that is needed is a reasonably modern web browser. To install it locally, one needs in addition a web server, MySQL database and GF services.



The editor runs entirely in the web browser, so once you have opened the web page and have documents and grammars loaded, you can continue translation editing while you are offline.

3.1.1 Software requirements

In order to install the editor, you need to have the following components:

1. The editor code itself (in the [eclipse package](#))
2. *For developer version only:*
 - Eclipse Helios JEE (3.6)
 - Google Web Toolkit plugin (tested with version 2.3.1)
3. Web server
 - Apache (tested with 2.2.14 on Ubuntu)
 - FastCGI (libapache2-mod-fastcgi)
4. Database
 - HSQL (tested with version 1.8.1)
 - HSQL-MySQL (1.8.1) -- a slightly modified version: [hsqldb-mysql-1.8.1-molto.zip](#)
 - MySQL server (tested with 5.1.54 and 5.1.62)
5. GF server
 - GF (tested with 3.3.3)
 - Haskell (tested with ghc 7.0.4, cabal-install 0.10.2)

In this section we assume that the user has Apache, MySQL and GF server configurations done. Please see [Appendix](#) for instructions on background settings.

3.1.2 Installation

3.1.2.1 Developer version

The prototype TT editor code is packaged as an Eclipse project archive <http://tfs.cc/molto/molto-tt-0.9-linux-eclipse-20120529.zip> ready for import in Eclipse (Helios).

Import the project in Eclipse. You should have Google Web Toolkit plugin (tested with version 2.3.1). The runtime editor files are found in `TT-0.9/www/editor/`. To install the runtime, the following files are placed under Apache2 server root (here `/var/www`) as shown.

```
/var/www/editor$ ls
grammars  index.html  org.grammaticalframework.ui.gwt.EditorApp  WEB-INF
```

When you have placed the files under `/var/www`, then you can launch the project in Eclipse. Choose from the menu `Run -> Run configurations -> Web Application -> (new configuration)`. In the tab `Server` untick `Run built-in server`. If you have put the files in directory `/var/www/editor`, then the launch address will be `127.0.0.1:8888/editor/index.html?gwt.codesvr=127.0.0.1:9997`.

Web server: Apache2 fastcgi and action modules must be enabled for the services. See installation notes at the end for a sample Apache2 virtual host below to handle the services from port 8888 (the default).

GF server: The editor requires also an installation of GF server. The server binaries are `content-service` (for authentication and simple mysql database management) and `pgf-`

service (for gf grammars). When compiling, the cabal option `--global` should be used; then the GF service binaries get installed in `/usr/local/bin`. They can be copied/linked under webserver (by default Apache2) `fcgi-bin` directory as follows.

```
/var/www/fcgi-bin$ ls -l
content-service -> /usr/local/bin/content-service
pgf-service -> /usr/local/bin/pgf-service
```

Database: The TT editor back end requires an installation of MySQL, HSQL and a Haskell library `hsql-mysql` by Krasimir Angelov. Further instructions how to create a database for MOLTO TT tools are in the installation notes.

The content service needs to read mysql database connection parameters from file `/usr/local/bin/fpath`. It should be in the same directory as `content-service` and contain four tokens, the mysql host and database names and the database owner credentials.

```
/usr/local/bin$ cat fpath
localhost moltodb moltouser moltopass
```

Then, the database is created by typing the following:

```
/usr/local/bin$ ./content-service fpath
```

Sign in: The prototype editor currently uses the Google authentication API for sign in. Authentication and authorization for Google APIs allow third-party applications to get limited access to a user's Google accounts for certain types of activities. A user needs to have a Google account to sign in to the application.

3.1.2.2 User version

All back-end requirements are needed also for the user version. Now, instead of opening the package in Eclipse, the only thing needed is to place the following files under Apache2 server root (here `/var/www`) as shown.

```
/var/www/editor$ ls
grammars  index.html  org.grammaticalframework.ui.gwt.EditorApp  WEB-INF
```

Then, to run the editor, just type the address

`127.0.0.1:8888/editor/index.html?gwt.codesvr=127.0.0.1:9997` into browser.

3.1.2.3 Limitations

Ideally, the same login should work throughout the different parts of the distributed toolkit. There should be some group scheme to set group level access restrictions. Eventually, we may want to provide MOLTO single-sign-on as a replacement for Google authentication.

3.1.3 Grammar manager

The prototype editor has a simple grammar manager that is supposed to allow a user to upload her grammars to the editor's grammar cache under her name. The cache kept is on the

editor server for reasons of speed and xss restrictions. The user chooses the current grammar from among the cached grammars using a drop-down list.

3.1.3.1 Limitations

The grammar manager is not yet completed.

3.1.4 Document manager

The prototype editor has a simple document manager that saves a translated document in and retrieves one from the mysql database using ContentService. The current document is saved in the database using a diskette icon on the editor page. The Documents tab shows the currently saved documents and allows the user to load a selected document for continued translation.

3.1.4.1 Limitations

Naming of documents is not yet supported. Both the grammar manager and document manager remain to be linked to the TMS.

3.1.5 Term manager

The TT editor includes a simple tabular equivalents editor for searching and editing translation correspondences from the web of data, including TermFactory services. The equivalents editor is an independent web application that may also be used standalone or as a plugin to other applications. When complete, the equivalents editor lets the user extend their GF grammars with terms entered in the term editor and/or upload them as term proposals to TermFactory.

3.1.5.1 Installation

The equivalents editor was built with the ExtJS javascript library. It can be downloaded from <http://tfs.cc/molto/molto-term-editor.tgz>. Unpack it and put the whole molto_term_editor directory under /var/www/ (or wherever your web server wants them, for example in Windows C:\Program Files\Apache\htdocs). Open the file editor_sparql.html in a browser.

Note that this is also included in the [complete editor](#) as one of the tabs. As for function, the versions are identical. The screenshot below is from the standalone version.

term page - Mozilla Firefox

GlobalSight Index of /molto The MOLTO Translation Too... https://kitwiki.../gs-install.txt term page

localhost/molto_term_editor/editor_sparql.html

Most Visited Latest Headlines Getting Started Editor jQueryify 带上你的女朋友去乌... Oxenstiernaregistret ... Rui Jin Hotel Shanghai... Kingston Ho

The term page

default:narrower KeyWords: Fish query

items				
Add Record Delete Record				
name	uri	en	es	de
Aquarium fish	http://dbpedia.org/resourc...			Zierfisch
Bait fish	http://dbpedia.org/resourc...	Bait fish		
Finnan Haddie	http://dbpedia.org/resourc...	Finnan Haddie		
Fish	http://dbpedia.org/resourc...	Fish Piscines Piscine Fishhes Fishes		
Fish hydrolysate	http://dbpedia.org/resourc...	Fish hydrolysate		
Fish kill	http://dbpedia.org/resourc...			Fischsterben
Florida black bass	http://dbpedia.org/resourc...	Florida black bass Florida Black Bass		
Operculum (fish)	http://dbpedia.org/resourc...	Operculum (fish)		
Orange clownfish	http://dbpedia.org/resourc...	Orange clownfish Orange Clownfish		
Orange kipper	http://dbpedia.org/resourc...	Orange kipper		
Pristipomoides filam...	http://dbpedia.org/resourc...	Pristipomoides filamentosus		
Whiting	http://dbpedia.org/resourc...	Whiting		
Pelagic fish	http://dbpedia.org/resourc...	Pelagic fish		
List of collective no...	http://dbpedia.org/resourc...	List of collective nouns for fish, invertebrates, and plants		
Cryothernia amphitreta	http://dbpedia.org/resourc...	Cryothernia amphitreta		
Chicolar	http://dbpedia.org/resourc...	Chicolar		
Forage fish	http://dbpedia.org/resourc...	Forage fish		
False trevally	http://dbpedia.org/resourc...			Kuhbarsch
Melafix	http://dbpedia.org/resourc...	Melafix		
Halecostomi	http://dbpedia.org/resourc...	Halecostomi		

Undo Submit

Page 1 of 3

Displaying 1 - 20 of 42

3.1.5.2 Use

The term editor consists of two tabular grids. In the first (left side) grid, enter a term in the text input and opt for wider or narrower concepts. In the latter case (the default) the editor shows on the right another grid of concepts that are classed narrower than the search term in the data source (by default, OntoText FactForge) and their designations in a predefined selection of languages. In the former case, the editor fills out the left side grid with concepts that are classed in the data source as wider than the search term. Clicking on one of them does a search for its subconcepts and terms, shown in the right side grid.

The term grid is editable and the editor remembers the user's edits to the cells in the grid.

3.1.5.3 Limitations

The data source and choice of languages are not yet user definable. The editor is not yet connected to the TermFactory or GF grammar back ends.

3.1.6 Editor

In the current version, there is a sign-in box and tabs for grammars, documents, editor, and terms, plus two to query and browse the loaded grammar. The latter services are familiar from other GF front ends and based on the GF grammar Web API.

3.1.6.1 Use

After sign in, the editor calls content-service to show the logged in user's grammars from the grammarusers mysql table in the grammar list. The user chooses a domain grammar. This brings to view the initial vocabulary known by the grammar as fridge magnets to choose from. Alternatively, the user can type or paste text in the editor window. At every new input, the active translation unit is sent to the back end for translation, and the set of fridge magnets is updated. When a translation unit is complete and translatable, it is simultaneously translated to all the available languages and the translations are shown on the screen (in blue). If an input is not parsable, the editor underlines the unparsable part. The user can back off to the point of deviation using backspace. In addition, There is a button for clearing the input.

The editor guides the text author by showing a set of fridge magnets and offers autocompletion to hint how a text can be continued within the limits of the current grammar.

3.1.6.2 Limitations

The prototype gives a first rough idea of how a web based GF translation editor could work. At present, however, it remains oriented to a very small vocabulary (fridge magnets are not apt to work well with thousands of words). It is also doubtful that the setup is fast enough for the amount of interactivity caused at speeds involved in professional translation. A reconsideration how the editor and the back end best play together is indicated. A related limitation is the strict left-to-right orientation of the parsing. UGOT seems to be working on a robust parser which allows other manners of combining parsing and editing. The proper disposition of the translation result is not worked out yet.

3.2 The extended translation tools prototype

We now move on to the extended prototype. We first recapitulate how the extended translation tools extend the one-translation scenario to a community of translators collaboratively using and maintaining MOLTO translation tools.

3.2.1 User management

For more flexibility (as well as vendor independence), the open source LDAP (The Lightweight Directory Access Protocol) based user management implementation from GlobalSight has been adapted for MOLTO. It allows distinguishing different roles and user groups, and controlling access to resources by roles. The GlobalSight user management solution has been conservatively extended for the needs of MOLTO TermFactory users. The following screenshot displays a user's roles as an ontology editor.

View User - Details - Google Chrome

Editor

View User - Details

localhost:9090/globalsight/ControlServlet?linkName=details&pageName=USR&action=details

MOLTO Multilingual Online Translation

Setup Data Sources My Jobs My Activities Reports

View User - Details

Basic Information

User Name:	KOE
First Name:	KOE
Last Name:	KOE
Title:	
Company Name:	TermFactory
Access Level:	TermFactoryEditor

Contact Information

Address:	
Home Phone:	
Work Phone:	
Cell Phone:	
Fax:	
Email Address:	lcarlson@localhost.localdomain
CC Email Address:	
BCC Email Address:	
Email Language:	English

Roles

Activity	Source Locale	Target Locale	Domain
EditTFS	Afar (Afghanistan) [aa_AF]	Afar (Afghanistan) [aa_AF]	http://foo/bar
EditTFS	Finnish (Finland) [fi_FI]	Finnish (Finland) [fi_FI]	http://suomi.fi
QueryTF	Finnish (Finland) [fi_FI]	Finnish (Finland) [fi_FI]	http://tfs.cc/*

Projects

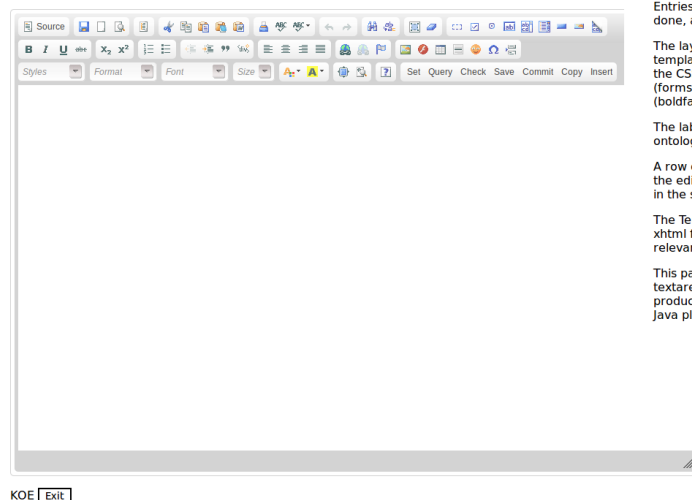
Name	Description
Template	

OK

Term ontology management roles are defined per domain, where a domain is represented by a regular expression on ontology URIs. The MOLTO GlobalSight user management system lets a company project administrator create users and grant them MOLTO TermFactory ontology read and write permissions. The TermFactory back end GateService reads the permissions off the GlobalSight LDAP directory and database and controls access to TermFactory content accordingly. If a user's credentials are not sufficient, TermFactory Gate will not permit term ontology queries or commits. The MOLTO permissions come over and above any constraints that ontology endpoints may impose on the content they manage. They enable fine grained project level control on who is allowed to do what to shared or restricted TermFactory resources.

TermFactory CKEditor

QueryService gate: no read permit for user KOE



CKEditor - The text editor for Internet - <http://ckeditor.com>

Copyright © 2003-2011, [CKSource](#) - Frederico Knabben. All rights reserved.

Legend

The content shown in the edit area on the left is WYSIWYG editable with JavaScript textarea editor CKEditor. FCKEditor has been embedded to many platforms, including mediawiki and Drupal.

Entries are fetched to the page using the TF query service and edited in place using the editor. When done, an edited entry can be or cached in a relational database or copied to a server as a document.

The layout of the terms is kept minimal in this sample. Entry orientation can be changed with TF entry templates (the Template parameter in the Set dialog). Decorations ('skins') can be changed by changing the CSS stylesheet. Concepts (meanings) are shown here in blue, terms (signs) in green and expressions (forms) in orange. Lighter shades of green and yellow indicate text fields. Editable content is strong (boldface), read-only content greyed (transparent).

The labels shown to the user are URIs or CURIEs by default. They can be localized by supplying a TF ontology as parameter locals or with a system json file generated from an ontology.

A row of TermFactory dialog buttons has been added to CKEditor. Button Set opens a css popup dialog to the editor form settings. Buttons Query, Edit and Copy are shortcuts to the corresponding commit buttons in the settings dialog. Button Menu opens a menu for inserting TF templates into the entry.

The TermFactory Menu dialog has tabs for each type of resource. The tabs get populated with json and xhtml files generated from some ontology using the query back end. They allow users to pick templates for relevant properties and values for each type instead of typing them in.

This page is an example of a TF editor mashup. It is a java server page which embeds a copy of the TF textarea editor. To embed a TermFactory editor to new platform, an appropriate plugin or extension is produced using the platform's own extension facilities. The EditForm java servlet can be used as such on java platforms. A TF MediaWiki Special:EditTerm extension is also available.

3.2.2 Document management

The simple document manager of the prototype editor remains to be upgraded to a more sophisticated XLIFF based document manager built using the GlobalSight document management API. See the MOLTO TT API document for more detail.

3.2.3 Lexical resources

A key consideration for the usability of MOLTO translation is the ease with which its text coverage can be extended by a user community. We need to pay great attention to adaptability. The most important factor in extensibility is lexical coverage. Grammatical coverage can be developed and maintained with language engineering, and grammatical gaps can often be circumvented by paraphrasing. In contrast, paraphrasing is not a real option for special domain terms. There are two cases to consider: either the abstract grammar misses concepts, or concrete grammars for some language/s are missing equivalents. In the first case, we need to extend the domain ontology and its abstract grammar. In the second case, we need to add terms.

For both ontology and term management, we apport to MOLTO the TermFactory ontology based terminology management concept. TermFactory is a system of distributed multilingual term ontology repositories maintained by a network of collaborative management platforms. It has been described at length in the TermFactory Manual at http://www.helsinki.fi/~lcarlson/CF/TF/doc/TFManual_en.xhtml.

The user of the MOLTO translation editor has direct access through the equivalents editor to querying and editing term equivalents for concepts already in available ontologies, either already in TermFactory or 'raw' from the Web of Data, in particular, the OntoText services serving data from FactForge repository.

3.2.3.1 Term management

Say for instance there is no equivalent listed for cheese in some language's concrete grammar FooLang. The author/translator can use the equivalents editor to query for terms for the concept food:Cheese in TermFactory or do a search through OntoText services for candidate equivalents, or, if she knows the answer herself, submit equivalents through the equivalents editor. The new equivalent/s are saved in the user's own MOLTO lexicon, and submitted to TermFactory as term proposals for the community to evaluate.

3.2.3.2 Ontologies

If there is a conceptual gap not easily filled in through the equivalents editor, there is the option of forwarding the problem to an appropriate TermFactory collaborative platform. This route is slower, but the quality has a better guarantee in the longer run, as inconsistency or duplication of work may be avoided. Say there is no concept in the domain ontology for the new notion that occurs in the source text. In easy cases, new concepts can be added through the equivalents editor, subclassing some existing concept in the ontology. In more complex cases, where negotiations are needed in the community, an ontology extension proposal is submitted through a TermFactory wiki. TermFactory offers facilities for discussing and editing ontologies and their terms. In due time, them modified ontology gets implemented in a new release of the GF domain abstract grammar.

3.2.3.3 Ontology-grammar interface

TermFactory ontologies are extensible and support reasoning. Instead of implementing domain ontology-to-grammar bridges over and over again for every new domain and application, it seems more promising to take advantage of the semantic network structure of (term) ontologies. Suppose verbalizations are already defined for a selection of upper or middle level ontologies. Special domain ontologies can subclass them and thereby also inherit the verbalizations that go with the superclasses and properties. UHEL is currently looking at the generalization of the MOLTO museum case ontology-to-grammar mapping in this direction.

3.2.4 Translation editing

The TT translation editor is just a prototype. Different scenarios and platforms may call for different combinations of its features. One way to go is to extend the prototype with further tabs and facilities for CAT tool support. But there is also the opposite alternative to consider of calling MOLTO translation tool services from a third party editor. GlobalSight has two built in translation editors, called popup editor and inline editor. The popup editor is a Trados TagEditor lookalike, while the inline editor has something of the look and feel of old Trados versions running WYSIWYG on Microsoft Word. The inline editor has been implemented in javascript using the FCKEditor library. It might just be feasible to embed MOLTO prototype editor functionalities into the GlobalSight editor(s). In the Globalsight setup, there is already support for importing cut-and-dried MT translations from a MT service, but here we are talking about something rather more intricate.

It is not immediately obvious which route would provide least resistance. From the point of view of GF usability, finding a neat way of embedding GF editing functions in third party translation editors could be a better sales position than trying to maintain a whole new

MOLTO translation environment. (Unless of course, the new environment is clearly more attractive to targeted users than existing ones.) We may also try to have it both ways.

3.2.5 Reviewing/feedback

It was noted above that blind translation in the case of incomplete or inadequate coverage in resource grammars can occasion a round of reviewing and giving feedback on the translations before publication. This part of the process is in its main outlines familiar from the translation industry workflow, and can be implemented as a variation of it. In the MOLTO workflow, reviewer comments are not returned (just) to the human author/translator(s), but they should have repercussions in the ontology and grammar management workflows. This part requires modifying and extending the existing GlobalSight revisioning tools to communicate with the MOLTO lexical resources and grammar services. The GlobalSight revisioning tools now use email as the human-to-human communication channel. We probably want to use a webservice channel for machine-to-machine communication, and possibly some web commenting system as an alternative to email.

3.2.6 Grammar engineering

To the extent grammar engineering can be delegated to translation tool users, it must happen transparently without requiring knowledge of GF. One way to do this is through what is known as example-based grammar writing in GF. Example-based grammar writing is a new GF technique for backward-engineering GF source from example translations. It can play a significant role in the translation-to-grammar feedback cycle. This part of the TT API will be borrowed from the MOLTO Grammar Developer Tools API.

The following sections describe what parts of the above list are already in place in the prototype and what remains to do.

3.3 GlobalSight

GlobalSight (<http://www.globalsight.com/>) is an open source Translation Management System (TMS) released under the Apache License 2.0. Version 8.2. was released on Sept 15, 2011. As of version 7.1 it supports the TMX and SRX 2.0 Localization Industry Standards Association standards.[2] It was developed in the Java programming language and uses MySQL database and OpenLDAP directory software. GlobalSight also supports computer-assisted translation and machine translation.

According to the documentation, GlobalSight has the following features:

- Customizable workflows, created and edited using graphical workflow editor
- Support for both human translation and fully integrated machine translation (MT)
- Automation of many traditionally manual steps in the localization process, including: filtering and segmentation, TM leveraging, analysis, costing, file handoffs, email notifications, TM update, target file generation
- Translation Memory (TM) management and leveraging, including multilingual TMs, and the ability to leverage from multiple TMs
- In Context Exact matching, as well as exact and fuzzy matching
- Terminology management and leveraging

- Centralized and simplified Translation memory and terminology management
- Full support for translation processes that utilize multiple Language Service Providers (LSPs)
- Two online translation editors
- Support for desktop Computer Aided Translation (CAT) tools such as Trados
- Cost calculation based on configurable rates for each step of the localization process
- Filters for dozens of filetypes, including Word, RTF, PowerPoint, Excel, XML, HTML, Javascript, PHP, ASP, JSP, Java Properties, Frame, InDesign, etc.
- Concordance search
- Alignment mechanism for generating Translation memory from previously translated documents
- Reporting
- Web services API for programmatic access to GlobalSight functionality and data
- Integrated with Asia Online APIs for automated translation

3.3.1 Installation

The latest full Linux install version of GlobalSight is 7.1.0.x . It can be updated to the current version 8.2.2.0 using publicly available upgrade packages. The GlobalSight 7.2.0.0 base version and the upgrade packages are available from SourceForge. (Copies are available from tfs.cc under /srv/GlobalSight_backup/upgrade. More detailed install instructions, including scripts to install LDAP for GlobalSight can be found at <http://tfs.cc/globalsight-molto-install/>. A fully functional GlobalSight site also needs access to email services.

To upgrade from a working install of GlobalSight 8.2.2.0 to MOLTO GlobalSight, download, unpack and run http://tfs.cc/molto/GlobalSight_Installer_8.2.2.1.zip.

There is also a complete MOLTO GlobalSight eclipse project archive at <http://tfs.cc/molto/molto-globalsight-8.2.2.1-linux-eclipse-20120529.zip> containing the source as well as the runtime.

3.3.2 MOLTO GlobalSight

MOLTO GlobalSight differs from GlobalSight out of the box in two ways. First, MOLTO GlobalSight extends MOLTO user roles to terminology editing. It will be discussed in more detail below in connection with TermFactory. Second, GlobalSight has two built in translation editors, called popup editor and inline editor. The popup editor is a Trados TagEditor lookalike, while the inline editor has something of the look and feel of old Trados versions running WYSIWYG on Microsoft Word. The inline editor has been implemented in javascript using the FCKEditor library. MOLTO GlobalSight extends the selection by embedding the MOLTO TT editor as a third option on the editor menu:

The screenshot shows the 'Activity Details' page for a job named '20120319-2314_demosample'. The page has a navigation bar with 'My Activities' and a breadcrumb trail. Below the job name, there is a description of the job and a set of tabs: 'Details', 'Comments', and 'Work Offline'. The 'Details' tab is active, showing a list of job attributes such as Job Name, Job ID, Activity, Company, Project, Project Manager, Source Word Count Total, Priority, Source Locale, Target Locale, Due By, Overdue, and Status. To the right of the details, there is a search bar for 'Target File name contains:' and a table of 'Primary Target Files'. The table has columns for 'Word Count', 'Source', and 'Translated Text'. A row is visible with a word count of 105 and a source link. A context menu is open over the source link, offering options: 'Open in MOLTO editor', 'Open in inline editor', and 'Open in popup editor'. Below the table, there is a 'Detailed Word Counts...' button and an 'Activity Comments' section showing a comment from 'Translation1' for the Finnish (Finland) locale. At the bottom, there is a table for 'Comment Creator', 'Date Created', 'Comments', and 'Attached Files', which currently shows no comments.

Clicking the option opens the Molto TT Editor in another window.

3.3.2.1 Limitations

As yet, content from the document under translation is not automatically imported into the MOLTO TT editor. Content can be cut and pasted into the MOLTO TT editor.

3.4 TermFactory

The MOLTO TermFactory prototype consists of the generic TermFactory codebase plus MOLTO related ontology content. At present, such content comprises the English-Finnish WordNet ontology. Integration of the TermFactory back-end with the MOLTO KRI over JMS is underway.

The TermFactory codebase consists of

- a term ontology query/editing back end run as an Axis2 Tomcat web service
- a Tomcat webapp that provides standalone term ontology query form and editor
- a MediaWiki installation with a TermFactory editor extension
- a link to the Disqus comment system

TermFactory is an architecture and a workflow for Semantic Web based, multilingual, collaborative terminology work. What this means in practice is that it applies Semantic Web and other document and language technology standards to the representation of multilingual


special language terms and the related concepts, and provides a plan for how such terminologies can be collected, updated, and agreed about by professionals, not only terminology professionals, all over the globe, during their everyday work on virtual work platforms over the web. As a whole, TF could be termed a [semantic web framework](#) for multilingual terminology work.

TF provides

- ontology and terminology formats
- format conversions
- query and edit tools
- repositories
- web services

for people to work on terms jointly or separately, building on the results of the work of others, while maintaining quality and consistency between the different contributions.

As a prototype, there is a MediaWiki platform for human to human collaboration on collecting terminological data plus a TF editor plugin for conveying the results of the collaboration into TermFactory ontology format. Here is a snapshot of a random MOLTO TF concept in the Wiki.




Page
Discussion
Entry Editor
Read
Edit

Navigation
Main page
Community portal
Current events
Recent changes
Random page
Help
Toolbox
What links here
Related changes
Special pages
Printable version
Permanent link

Wn30:synset-oil painting-noun-1

wn30:synset-oil_painting-noun-1 in europeana
oil painting in Princeton WordNet 3.1
oil painting in FinWordNet en
oil painting in FinWordNet fi
oil painting in Sanakirja.org
Öljymaalaus in Sanakirja.org
oil painting in Wiktionary
Öljymaalaus in Wiktionary


Term oil painting in English Wikipedia



Mona Lisa, Leonardo da Vinci, c. 1503-06

Oil painting is the process of painting with pigments that are bound with a medium of drying oil—especially in early modern Europe, linseed oil. Often an oil such as linseed was boiled with a resin

Term Öljymaalaus in Finnish Wikipedia



Mona Lisa on Leonardo da Vincin tunnettu öljyvärimaalaus

Öljymaalaus eli öljyvärimaalaus on öljyväreillä kankaalle, mutta aiemmin puu oli yleisempi pohja

3.4.1 Use

MOLTO TermFactory Mediawiki is used in the usual way a wiki works. In the demo prototype, it has been populated with the Finnish-English Wordnet (ca. 100K concepts, 2 languages, ca. 200K terms per language). The pages are generated automatically on demand. A Wordnet page currently only consists of a set of iframes and links to related lexical resources on the web. In actual use, each category (Wordnet is one) may generate its own boilerplate page design to help users describe and discuss the concepts of a category and their

designations in different languages. A commenting system is in place that can be shared between different platforms and applications. The discussion threads are indexed by the URI of the relevant resource.

The TermFactory ontology content related to a resource can be queried and edited on the Mediawiki platform using a TermFactory ontology editor extension, shown on top of the page as the Entry Editor tab. Below is a snapshot showing the TF editor opened to the TermFactory entry corresponding to the chosen WordNet term.

The screenshot displays the TermFactory ontology editor interface. On the left is a navigation sidebar with links like 'Main page', 'Community portal', and 'Recent changes'. The main area shows the 'Special:EditForm/Wn30:synset-oil painting-noun-1' page. The editor uses a CKEditor-style WYSIWYG text area to display and edit RDF triples. The visible triples are:

- wn30:synset-oil painting-noun-1**
 - rdfs:type**
 - wn20schema:NounSynset
 - wn20schema:Synset
 - rdfs:comment**
 - a picture painted with oil paints
 - wn:hypernym**
 - wn30:synset-painting-noun-1
 - wn:hyponym**
 - wn30:synset-canvas-noun-2
 - wn:synset**
 - wn30:synset-oil painting-noun-1
 - rdfs:label**
 - oil painting
 - oil painting
 - owl:sameAs**
 - wn:WN30-103844349
 - meaning of**
 - **rdfs:type**
 - wn20schema:NounWordSense
 - **sign:hasTranslation**
 - wn30fi-fi-öljymaalaus-N_-_wn30-synset-oil painting-noun-1
 - **wn:synset**
 - wn30:synset-oil painting-noun-1

The interface includes a toolbar with various editing tools and a 'Thread wn30:synset-oil painting-noun-1' link at the bottom.

Instead of going by way of fill-in forms, the TermFactory approach is to support direct WYSIWYG editing of localized ontology triples in a HTML textarea editor. The TermFactory editor application uses the CKEditor javascript textarea editor for this purpose. TF adds to the CKEditor standard release a special purpose plugin that adds TermFactory specific action buttons and a menu to the standard issue.

While staying conceptually close to the original RDF format of the data, the TermFactory editor layout is quite versatile. With suitable parameters, it can be tweaked to show ontology content editable in shapes already familiar to professional terminologists. There is a customisable, schema-aware insertion menu to help inserting relevant content, plus customisable input and output layout templates. The editor is not limited to TermFactory ontologies, as it is built on a general purpose textarea editor using a generic RDF to HTML mapping.

A specialty of TermFactory is that it supports terminological reflexion. The metaterminology used in the editor is not fixed, but can be changed by giving it a TF term ontology as parameter. Using TF localization and bridge ontologies, not only the editor interface, but also the content shown can be localized to a user community's conceptualization, language and terminology. Here is the same editor page fetched after setting Mediawiki language settings set to Finnish. Note how the terminological metalanguage used in the entry is now shown in Finnish. (The localization is not complete, because the current localization ontology's coverage has some gaps.)

3.4.2 Installation

The TermFactory source code is on svn at svn.it.helsinki.fi/repos/termfactory. A username and password on the repository server is needed for checkout. To check out a path, choose installation directory, go to it and do `svn checkout`

<https://<username>@svn.it.helsinki.fi/repos/termfactory/path> .

The compiled web archive files for TF are

io/lib/tf-io.jar	The core library (offline tools)
ws/service/TFServices.aar	The Axis2 webservice archive
ws/servlet/TermFactory.war	The Tomcat webapp archive

These three archives should be enough for deployment of TF in Linux from binaries on Tomcat running Axis2. Installations of mysql and Jena TDB are needed for persistent storage of ontologies on the TermFactory server. File upload services require prior installation of WebDAV. Detailed TF source build and install instructions are available on request.

TermFactory MediaWiki is MediaWiki out of the box plus the TermFactory MediaWiki extension, downloadable from the TermFactory svn path fe/TermFactory. The extension requires installing TF back end, of course.

- Install MediaWiki 1.16 (or newer)
- Put everything under extensions/TermFactory
- Add `require("$IP/extensions/TermFactory/TermFactory.php");` to `LocalSettings.php` in the main directory
- go to page `Special:EditTerm`

3.4.3 Limitations

User management between MediaWiki, TermFactory services, and TermFactory WebDAV is not fully in synch yet.

4. Integration

This section comments on the current status of the integration the different parts.

- Linking between the MOLTO TT translation editor and GlobalSight document management and translation process This linking is important for usability, but is closely connected to the further development of the TT editor and the MOLTO GF back end. Alternatives range from cut and paste (already supported) to automatic processing of XML/XLIFF document elements marked for MOLTO translation.
- Linking between the equivalents editor and TermFactory, in particular
 - TermFactory back end to answer equivalents editor queries to populate the editor from TF repositories
 - TermFactory back end to allow equivalents editor to communicate user additions to TF

A JSON format i/o API for converting term equivalent tables to/from TermFactory to term ontology format is in place in TermFactory, so this item is almost complete.

- Communication between the equivalents editor, TermFactory and the GF grammar services, in particular,
 - conversion of lexical entries between TermFactory ontology format and GF abstract grammar format
 - updating of GF grammars with user defined entries

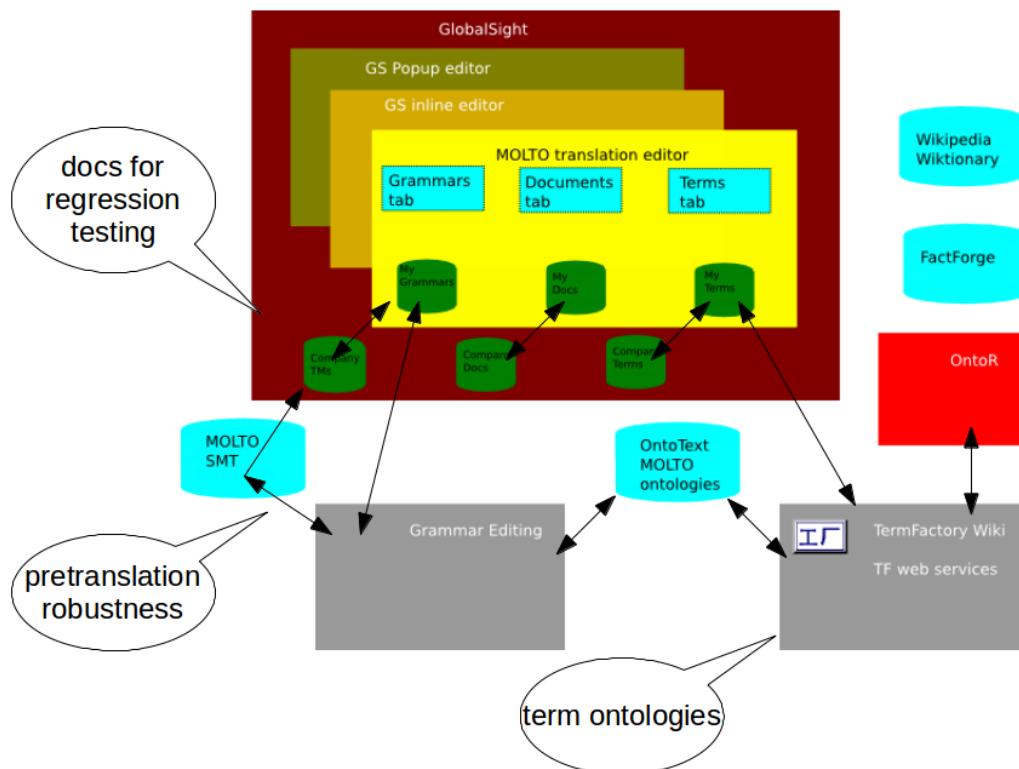
A mapping between MOLTO grammars and ontology formats has been defined by UGOT on the Museum case. UHEL has started generalizing the solution.

- Linking between translation editor/s, translation leveraging tools (TM/termbank) and GF services, in particular

- linking between the MOLTO (GlobalSight) comment/review system and TermFactory
- linking between the MOLTO (GlobalSight) comment/review system and GF grammar editing services

These steps are less urgent, so they will be left last.

Here is a figure showing some of the connections in the design.



5. Requirements on the GF grammar and translation APIs

This section repeats the wishlist of requirements from Translation Tools on the GF grammar and translation API.

Assume the GF translation goes to a reviser, working with or without another copy of the MOLTO translation tool. The corrected translation, in XLIFF form, should be brought to GF's attention. This calls for a new functionality from the GF grammar API: one which corrects the grammar and lexicon software to produce the output required by the corrected translation. This functionality is to be built on the GF example-based grammar writing methodology.

In order for the corrections to converge, revised translations must accumulate so that the newest corrections do not falsify earlier ones. The collection of manual corrections may become ambiguous or inconsistent, which situation should also be recognised and brought to the attention of a grammar engineer. Again, it is important to pay attention to user roles and rights.

We may want to provide ways to override GF translations with canned translations. At the translation tools level, this can happen by preferring TM translations over GF. We should also consider ways to override compositional translations on GF grammar level.

Another requirement is translation time update of grammar, at least the lexicon, so that translator's on the fly lexicon additions are possible.

If we want to support translating formatted documents using XLIFF, the minimum requirement is that the GF translation API handles XLIFF inline tags.

Appendix: Developer notes

The complete MOLTO TT prototype editor code is downloadable as an eclipse (Helios) project archive <http://tfs.cc/molto/molto-tt-0.9-linux-eclipse-20120529.zip>. The TT editor's database back-end Haskell source code is packaged as <http://tfs.cc/molto/molto-tt-server-0.9-linux-20120529.zip>.

Apache web server settings

Install apache and fastcgi with apt-get: `sudo apt-get install apache2 libapache2-mod-fastcgi`

Here is a sample Apache2 virtual host below to handle the MOLTO TT back end services from port 8888 (the default). The back end server is supposed to be in the same domain as the editor to avoid cross-domain scripting violations. Copy the text below to `/etc/apache2/sites-available/default`

```
<VirtualHost *:8888>
    ServerAdmin webmaster@localhost

    DocumentRoot /var/www
    AddDefaultCharset UTF-8

    <Directory />
        Options FollowSymLinks
        AllowOverride None
    </Directory>
    <Directory /var/www/>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride None
        Order allow,deny
        allow from all
    </Directory>

    # Allow fastcgi services from fcgi-bin.
```



```

<Directory "/var/www/fcgi-bin/">
    Options +ExecCGI
    AddDefaultCharset UTF-8
    SetHandler fastcgi-script
</Directory>

    # Identify pgf-service as a fastcgi server
    FastCgiServer /var/www/fcgi-bin/pgf-service

    # Identify content-service as a fastcgi server
    FastCgiServer /var/www/fcgi-bin/content-service

    # Make action pgf-service handle pgf files
    Action pgf-service /fcgi-bin/pgf-service
    AddHandler pgf-service .pgf
    AddCharset UTF-8 .pgf
</VirtualHost>

```

After you have copied the above to `/etc/apache2/sites-available/default`, activate the changes:

```

sudo a2enmod fastcgi
sudo a2enmod actions

```

Finally, restart apache by typing `sudo service apache2 restart`.

Some attested problems

1. Server doesn't start, error message is just "fail". If you copy and paste conf files, make sure they don't override any previous configs you might have.
2. "Permission denied" error messages when trying to access (chmod 775) files in `/var/www`.

Solution was to change the ownership group from root to all users: `sudo chown -R root:www-data /var/www/` (the name of the group might vary, you can see yours by seeing which group has `/var/www`; do `grep "/var/www" /etc/passwd`).

GF server settings

Get the latest sources for GF from darcs repository. The instructions are here:

<http://www.grammaticalframework.org/download/index.html>

Assuming you have the source files, go to `src/server` and type `sudo cabal install -f content --global`. It is important to use the option `--global`, because by default they are installed in the home directory, and that doesn't go well with Apache. The binaries will be installed in `/usr/local/bin`. Because of Apache, you need to set their owner group to the apache group (depending on machine, e.g. `www-data` or `apache`).

The next step is to link the binaries to `/var/www/fcgi-bin`.

```

/var/www$ sudo ln -s /usr/local/bin/content-service fcgi-bin/
/var/www$ sudo ln -s /usr/local/bin/pgf-service fcgi-bin/

```


Some attested problems

1. gf-server-1.0 depends on fastcgi-3001.0.2.3 which failed to install. If you get the following error, you must first have C library fcgi. First install libfcgi-dev (sudo apt-get install libfcgi-dev), then try again to install PGF service and content service: sudo cabal install -f content --global

Developer version settings

This applies only if you want to use the editor from Eclipse. Otherwise you don't need any of this.

To test the TT editor under eclipse using GWT devMode, we found it necessary to recompile content-service to add the gwt code server port parameter to page URLs. To do so activate the following lines in ContentService.hs . We have been using eclipse 3.6 JEE with Google Web Toolkit version 2.3.1.

```
-- devModeScriptName = (liftM2 (++)) (getVarWithDefault "SCRIPT_NAME" "")
-- (return "?gwt.codesvr=127.0.0.1:9997")
-- path <- devModeScriptName
```

A corresponding change is needed in the client code. Activate the following line in TT-0.9/src/org/grammaticalframework/ui/gwt/client/SettingsPanel.java

```
// String defaultUrl = "/fcgi-bin/content-
service?gwt.codesvr=127.0.0.1:9997";
```

Launch settings for building and testing under devMode under eclipse (in

\$HOME/workspace/.metadata/.plugins/org.eclipse.debug.core/.launches):

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<launchConfiguration type="com.google.gdt.eclipse.suite.webapp">
<booleanAttribute key="com.google.gdt.eclipse.core.RUN_SERVER"
value="false"/>
<stringAttribute key="com.google.gdt.eclipse.core.SERVER_PORT" value="80"/>
<stringAttribute key="com.google.gdt.eclipse.suiteMainTypeProcessor.
PREVIOUSLY_SET_MAIN_TYPE_NAME" value="com.google.gwt.dev.DevMode"/>
<booleanAttribute key="com.google.gdt.eclipse.
suiteWarArgumentProcessor.IS_WAR_FROM_PROJECT_PROPERTIES" value="true"/>
<listAttribute key="com.google.gwt.eclipse.core.ENTRY_POINT_MODULES">
<listEntry value="org.grammaticalframework.ui.gwt.EditorApp"/>
</listAttribute>
<stringAttribute key="com.google.gwt.eclipse.core.URL" value="editor"/>
<listAttribute key="org.eclipse.debug.core.MAPPED_RESOURCE_PATHS">
<listEntry value="/TT-0.9-ORIGINAL"/>
</listAttribute>
<listAttribute key="org.eclipse.debug.core.MAPPED_RESOURCE_TYPES">
<listEntry value="4"/>
</listAttribute>
<stringAttribute key="org.eclipse.jdt.launching.CLASSPATH_PROVIDER"
value="com.google.gwt.eclipse.core.moduleClasspathProvider"/>
<stringAttribute key="org.eclipse.jdt.launching.MAIN_TYPE"
value="com.google.gwt.dev.DevMode"/>
<stringAttribute key="org.eclipse.jdt.launching.PROGRAM_ARGUMENTS" value="-
startupUrl editor -war $HOME/workspace/TT-0.9/www/editor -noserver -
```

```
remoteUI "${gwt_remote_ui_server_port}:${unique_id}" -logLevel INFO -
codeServerPort 9997 org.grammaticalframework.ui.gwt.EditorApp"/>
<stringAttribute key="org.eclipse.jdt.launching.PROJECT_ATTR" value="TT-
0.9"/>
<stringAttribute key="org.eclipse.jdt.launching.VM_ARGUMENTS" value="-
Xmx512m"/>
</launchConfiguration>
```

Database settings

HSQL and HSQL-MySQL installation

First you need to install HSQL. It's in [hackage](#), so it can be installed by typing `sudo cabal install hsql --global`.

The public version of the Haskell MySQL package `hsql-mysql-1.8.1` used by the TT content service appears to have a bug that prevents multiple successive mysql procedure calls. A debugged version of the package can be found at <http://tfs.cc/molto/hsql-mysql-1.8.1-molto.zip>.

Install the debugged version:

- Download and extract the files in [hsql-mysql-1.8.1-molto.zip](#)
- Go to the top level directory, where you can find `hsql-mysql.cabal`
- Install by typing `sudo cabal install --global`

Some attested problems

1. HSQL 1.8.2 doesn't install. Solution: try HSQL 1.8.1. You can choose it by typing `sudo cabal install hsql-1.8.1 --global` or downloading the version from [hackage](#).

MySQL server settings

In addition to the above, you need a mysql server. You can install one by typing `sudo apt-get install mysql-server`.

Content-service uses a mysql database to store users, grammars and documents.

To create the database connection you need to do the following steps:

1. create the database for molto content service
2. create the content service database owner user
3. give that user all rights for the database
4. create key file `fpath` somewhere with the unholy quartet `host db user pwd` in format known to `haskell readFile`.
5. call service with `fpath` as the only argument to create user, grammar and document tables:
`content-service fpath`

Now create the database:

```
$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
...
```

```
mysql> CREATE DATABASE moltodb;
CREATE USER moltouser IDENTIFIED BY 'moltopass';
GRANT ALL on moltodb.* to moltouser;
```

Query OK, 1 row affected (0.02 sec)

```
mysql> Query OK, 0 rows affected (0.00 sec)
```

```
mysql> Query OK, 0 rows affected (0.00 sec)
```

```
mysql> show databases;
+-----+
| Database          |
+-----+
| information_schema |
| moltodb            |
+-----+
2 rows in set (0.00 sec)
```

```
mysql> quit
Bye
```

Next, create the database files.

```
/usr/local/bin$ ./content-service fpath
```

And then log in to mysql with the user moltouser.

```
mysql -u moltouser -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
```

```
mysql> use moltodb;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A
```

```
Database changed
mysql> show tables;
+-----+
| Tables_in_moltodb |
+-----+
| Documents          |
| GrammarUsers       |
| Grammars           |
| Users              |
+-----+
4 rows in set (0.00 sec)
```

If so the tables got created ok. There is nothing yet in the tables. After you first sign in with your google account, then your user account will be in the table Users. You can query any of the tables by writing `select * from <table>`.

Tabular editor (stand-alone version)

To install the tabular equivalents editor source, do this:

1. checkout the whole source code directory from `https://svn.it.helsinki.fi/repos/molto/trunk/molto_term_editor/` and put it in any place where the apache server can reach, i.e. `/var/www/`.
2. download the `extjs-4.0.2a` package from `http://extjs.cachefly.net/ext-4.0.2a-gpl.zip`, and uncompress it as `extjs-4.0.3` under the directory `molto_term_editor/`.
3. then, you can open the link `http://localhost/molto_term_editor/editor_sparql.html`, if you put the source code under `/var/www/`.

GlobalSight

The GlobalSight installation values are kept in `$HOME/workspace/GS-8.2.2.1/main6/tools/build/dist/GlobalSight/install/data/installValues.properties`. The following shows settings used for a MOLTO GlobalSight eclipse installation (with `$HOME` replacing the installation directory and `PASSWORD` the password/s) :

```
#Mon May 28 22:52:03 EEST 2012
mailserver=mail.domain.com
system_log_directory_forwardslash=$HOME/workspace/GS-
8.2.2.1/main6/tools/build/dist/GlobalSight/logs
install_data_dir_forwardslash=$HOME/workspace/GS-
8.2.2.1/main6/tools/build/dist/GlobalSight/install/data
server_host=localhost
database_password=PASSWORD
database_server=localhost
database_username=globalsight
ldap_password=PASSWORD
ldap_install_dir=/var/lib/ldap
server_port=9090
ldap_host=localhost
gs_home=$HOME/workspace/GS-8.2.2.1/main6/tools/build/dist/GlobalSight
ldap_username=ldap_connection
admin_email=WelocalizeAdmin@domain.com
system4_admin_username=gsAdmin
ldap_base=globalsight.com
ldap_port=389
GS_HOME=$HOME/workspace/GS-8.2.2.1/main6/tools/build/dist/GlobalSight
cap_login_url=http\://127.0.1.1\:9090/globalsight
```
