The MOLTO Translation Tools API

Lauri Carlson et al.
MOLTO Deliverable 3.1

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. Basic domain language coverage does not guarantee that all terms and idioms in the translatable document are covered. 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 report explains the components of the MOLTO translation tool API. The API is the main value because end user translation environments and tools are many and change quickly with time, while MOLTO tools should have a more long lasting value. The components of the API include at least the following (The Core API):

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

Provisions for interfacing with further typical CAT tool components will be considered (The Extended API).

A model implementation of MOLTO translation tools based on the API will be demonstrated in the MOLTO prototype due as deliverable 3.2 in February 2012.

Translation scenarios

GF is a multilingual interlingual translation system geared toward multilingual generation. As a proof of concept, GF demos display immediate generation into dozens of languages from a tiny grammar. Extended to a more realistic case, this scenario could have a native-language editor producing text for more or less immediate multilingual distribution, for instance, a multilingual website. For this scenario to work, the translations should be acceptable as is without target language native revision.

The GF approach as such suits best an authoring/pre-editing scenario, where an original author or authorised content editor can choose or edit the original message to conform to a domain specific constrained language grammar, which GF is then expected to blind translate reliably to a number of languages. In real life situations, the text to translate is likely to be at least partially new to the system, and no guarantee can be given that the translation is correct in all the generated languages. It is specifically such extensions of the original MOLTO "fridge magnet" demo scenario that this document tries to address.

The current professional human translation scenario is quite different. It is a post-editing scenario. The roles of author (client) and translator are separated. The translator has quite restricted authority over the target text and almost none over the original, aside from obvious errors. The translation process is normally bilingual. This is because the translation is created or at least chosen by human, and human translators rarely have professional competence in more than one or two languages besides their native language.

The preferred professional translation direction is from a second language to native language, because for humans language generation is more demanding than language understanding. In this direction, the translator can exploit external resources to understand the source text and use her native competence to check the quality of the target text. Even in this case, a native subject expert is usually needed to check the translation. The reviser need not know the source language or have the source text at hand.

The extended MOLTO translation scenarios considered here spread between these two extremes. We may still assume that the translator has some authority over the text to produce, i.e. she is the author or is authorized to adapt the text to better satisfy the constraints of the translation grammar. The MOLTO pre-editor/translator should be native or at least fluent in the source language, and familiar with the domain or at least its special language in order to know how the message can be (para)phrased. Thus the extended MOLTO scenario retains an element of constrained-language authoring or pre-editing.

But we may need to relax the blind generation assumption. Although the GF engine may give warnings or suggestions when it is unsure or knows the translation fails, there are likely to be cases where the translation is technically correct, but inadequate for human consumption. A native revision is then needed for one or more target language(s). As in the human translation case, the author/translator can at best serve as informant for one or a few target languages. For the rest, the translation needs to be distributed to a pool of revisers. In real life, a translation has to go out even if GF fails. There must be a way to override GF with human translations. If the translation were a one-off affair, that could end the process. However, in many real life scenarios, the same or very similar texts will come up for (re)translation, and in that case, the results of the revisions should get fed back to the translation cycle, to avoid making the same errors twice. In other words, we should make the MOLTO translation system consisting of GF and the human users an adaptive whole. This is the most demanding part to conceive here. Pre-editing MT has not been very successful in the past, probably partly just because not enough attention has been given to practical concerns like these.

Translation industry standards

As document automation progresses, professional translation is merging into localization, or the adaptation of software to a new locale (language and culture). Translation used to differ from localization in that translators were not expected to worry about formats or the document lifecycle. Translations were shipped to translators as raw text and returned as such. In an intermediate phase, a specialized localization industry developed to multilingualize software, preserving the source format. More recently, with multi channel publishing and document toolchains, there is again a push to separate form from content. The localization industry solution to these conflicting pressures is to separate content from form in a reversible fashion. Localization formats and tools like Gnu gettext and XLIFF make provisions for extracting the translatable text from a document in a way that allows embedding the target text in the same document form.

The current GF translation engine as such is neutral about the format of the text it receives, but the existing resource grammars expect text to come in raw form. It should be technically possible to include document formatting in GF parsing and generation, and if suitably restricted, that might be the most efficient solution for the translation of inline tags. However, for the rest, it seems best to take advantage of existing content extraction technologies in translation industry. We propose to use XLIFF for MOLTO translatable document format in the extended API.

XLIFF is one of the OASIS LISA OAXAL standards. As of 2011 February 28, the Localization Industry Standards Association (LISA) is insolvent. The LISA standards continue to be used by the industry. The OASIS Open Architecture for XML Authoring and Localization (OAXAL) reference model, comprises the following open standards:

- Unicode
- XLIFF - OASIS XML Localization Interchange File Format
- SRX – LISA Segmentation Rules Exchange
- TMX – LISA Translation Memory Exchange
- GMX/V – LISA Word and Character Count Standard
- W3C ITS – Internationalization Tag Set
- xml:tm – LISA XML based text memory

Translation tools survey

For the extended scenario, we may add other industry standard CAT tools for MOLTO translators to use besides the core list above. There is a plethora of packages for CAT and translation project management/automation both commercial and open source. It seems best to borrow from existing open source packages that comply with translation industry standards, instead of reinventing the wheel. Examples of CAT packages are

- OmegaT http://www.omegat.org/
- OpenTM2 http://www.opentm2.org/
- OpenTMS http://www.opentms.de/?q=en/node/27
- TinyTM http://tinytm.sourceforge.net/
- SwordFish http://www.maxprograms.com/products/swordfish.html
- Heartsome http://www.heartsome.net/EN/home.html

SwordFish and HeartSome are commercial. Examples of translation project management and workflow automation packages are

- ProjectOpen http://www.project-open.com/en/modules/index.php
- GlobalSight http://www.kmworld.com/Articles/News/News/Open-source-translation-management-52166.aspx
- SDL WorldServer http://www.sdl.com/en/language-technology/products/translation-management/worldserver/default.asp
- Across v5 http://www.across.net/en/index.aspx
- XTM http://www.xtm-intl.com/

Of the systems listed above, ProjectOpen and GlobalSight are open source, the rest are commercial.

From existing open source projects we can shop for properties generally expected from TM (http://en.wikipedia.org/wiki/Translation_memory), CAT (http://en.wikipedia.org/wiki/Computer-assisted_translation), and translation project management software. Some commercial systems also have open interfaces, e.g. Across (http://en.wikipedia.org/wiki/Across_Systems). Here are some translation tools listings from the Web.

- http://socialsourcecommons.org/toolbox/show/1107
- http://translatewiki.net/wiki/Main_Page
- http://www.translatewrite.com/foss/index.php?s=foss
- http://www.i18nguy.com/TranslationTools.html
- http://www.translatorstraining.com/sito/

For comparisons, see e.g. Wikipedia.

The translation process

Here we study variants of the machine assisted translation process, to develop a version that suits MOLTO.

The translation industry workflow

To have a point of comparison, we review the practices in the professional translation industry today. Going beyond the 90's single-user computer-assisted translation (CAT) setup with a translation editor, translation memory, and termbase, current translation management system (TMS) packages provide tools for managing complex translation industry projects involving clients, project managers, and a distributed pool of translators, reviewers, and subject experts. Many aspects of the workflow and the associated communication (notification, document transfer) can be automated in these systems. For an example of a translation industry workflow, we take the]project-open[Translation Workflow. Tasks typically covered by translation project and workflow management packages include

- Analyzing source text using a translation memory
- Generating quotes for the customer
- Generating purchase orders for providers and freelancers
- Executing the project
- Monitoring the project
- Generating invoices to the customer
- Generating invoices for providers and freelancers

Roles within the translation workflow

In]project-translation[five user roles can be defined.

- Translator: A person responsible to execute the translation and/or localization projects.
- Project Manager: A person responsible for the successful execution of projects. Project Managers frequently act as Key Account managers in small translation agencies.
- Senior Manager: Is responsible for the financial viability of the business and is the ultimate responsible for the relationships to customers
- Key Account Manager: A person responsible for the relationship to a number of customers. We assume that customer project requests are handled by a Key Account Manager and then passed on to a project Manager for execution.
- Resource Manager: A person responsible for the relationship with translation resources and providers.

GlobalSight has yet more default roles:

- Administrator
- Customer
- LocaleManager
- LocalizationParticipant
- ProjectManager
- SuperAdministrator
- SuperProjectManager
- VendorAdmin
- VendorManager

New roles can be invented at will in Globalsight. As discussed in the MOLTO requirements document (Deliverable 9.1), The role cast in MOLTO can have at least these roles:

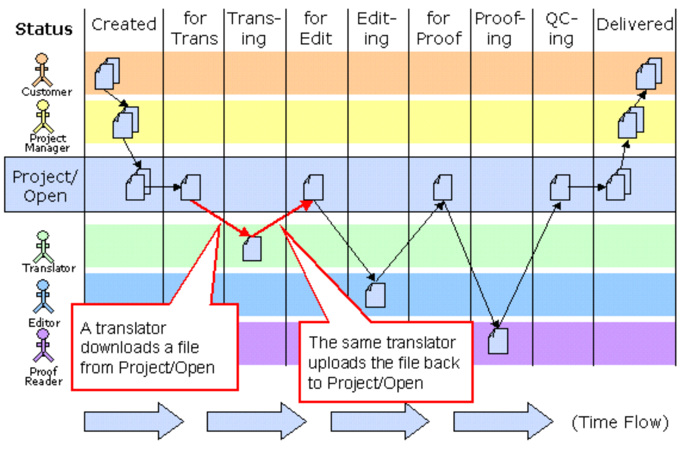
- Author
- Editor
- Translator
- Reviewer (domain/language expert)
- Ontologist
- Terminologist
- Grammarian

The Project Cycle

The figure below shows in a schematic way in which the workflow proceeds:

- A document from the customer is passed to the project manager for translation (This action is represented by the arrow from documents in the upper left corner from the client to the project manager).
- The Project Manager uploads the document into the]project-open[System
- A translators downloads the file
- The same translators uploads the translated files

Similarly for editors and proofreaders. Finally, the project manager retrieves the document and sends it to the customer. Alternatively, the project manager can allow the customer to download the files directly. In addition to tracking the status of a project at every stage, the system allows the project manager to allocate projects to the most suitable team and streamline the freelancers' job.



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 the software 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 Contect 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

GlobalSight Web Services API

GlobalSight provides a web services API (http://www.globalsight.com/wiki/index.php/GlobalSight_Web_Services_API). It is used to integrate external systems to GlobalSight in order to submit content to the localization/translation work-flow, and monitor its status. The Web services API allows any client to connect and exchange data with GlobalSight, regardless of its implementation technology or operating system. The web service provides methods for

- Authentication
- Content Import
- Project management
- Activity and task management
- User management
- Locale management
- Job management
- Content export
- Documentum support
- Translation Memory management
- Term Base management
- CVS support

For convenience, we shall borrow parts of the MOLTO extended API from the GlobalSight translation management system.

The web localization workflow

The translation industry workflow is top-down controlled, built on email and file transfers. For a more collaborative bottom-up approach, we can look at web localization. Web platforms are getting localized by a collaborative translation workflow. Here, translation is typically crowdsourced to a pool of volunteers, who either translate manually online or download po files to work on with local tools. The website coordinates the effort. Different projects may have assigned managers that monitor the collaboration. It is exemplified by the Translate toolkit (http://en.wikipedia.org/wiki/Translate_Toolkit) used to collaboratively localize open source software packages.

An instance of the Translate toolkit is Pootle http://en.wikipedia.org/wiki/Pootle, an online translation management tool with translation interface. It is written in the Python programming language using the Django framework and is free software originally developed and released by Translate.org.za in 2004. It was further developed as part of the WordForge project and the African Network for Localisation and is now maintained by Translate.org.za.

Pootle is intended for use by free software translators but is usable in other situations. Its main focus is on localization of applications' graphical user interfaces as opposed to document translation. Pootle makes use of the Translate Toolkit for manipulating translation files. The Translate Toolkit also offers offline features that can be used to manage the translation of Mozilla Firefox and OpenOffice.org in Pootle. Some of Pootle's features include terminology extraction, translation memory, glossary management and matching, goal creation and user management.

It can play various roles in the translation process. The simplest displays statistics for the body of translations hosted by the server. Its suggestion mode allows users to make translation suggestions and corrections for later review, thus it can act as a translation specific bug reporting system. It allows online translation with various translators and lastly it can operate as a management system where translators translate using an offline tool and use Pootle to manage the workflow of the translation.

The Translate Toolkit API is documented at http://translate.sourceforge.net/doc/api/. It is open source subject to the GPL licence.

The Google Translator Toolkit

Google provides a free service for translating webpages by post-editing Google MT results. The toolkit allows users to

- Upload and translate documents
- Use documents from your desktop or the web.
- Download and publish translations
- Publish translations to Wikipedia™ or Knol.
- Chat and share translations online
- Collaborate online with other translators.
- Use advanced tools like translation memories and multilingual glossaries.

The Google Translator Toolkit Data API allows client applications to access and update translation-related data programmatically. This includes translation document, translation memory, and glossary data stored with Google Translator Toolkit. The Google Translator Toolkit API is now a restricted API (http://code.google.com/apis/gtt/).

The MOLTO translation workflow

We now consider the MOLTO translation scenario. The MOLTO translation demo editor (see figure further below) supports a one-person workflow where the same person is the author(ised editor) of the source and the translator. Technically we can extend this to a more collaborative scenario where more actors are involved as in the professional workflow above, by adding the usual project support tools to the toolkit. A more difficult part is to adjust the workflow so that the adaptivity goal above 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.

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 neeeded. 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 Docboook 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.

The MOLTO Translation tools API

The MOLTO Translation Tools API exposes the most important operations used in translating with GF in MOLTO. It makes them available for programmers who want to create alternative accesses to GF translation tools, besides the MOLTO web translation demo platform. The API is divided into a Core API basiclly 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 Core API include at least the following:

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

The components of the MOLTO TT Extended API 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 API. 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.

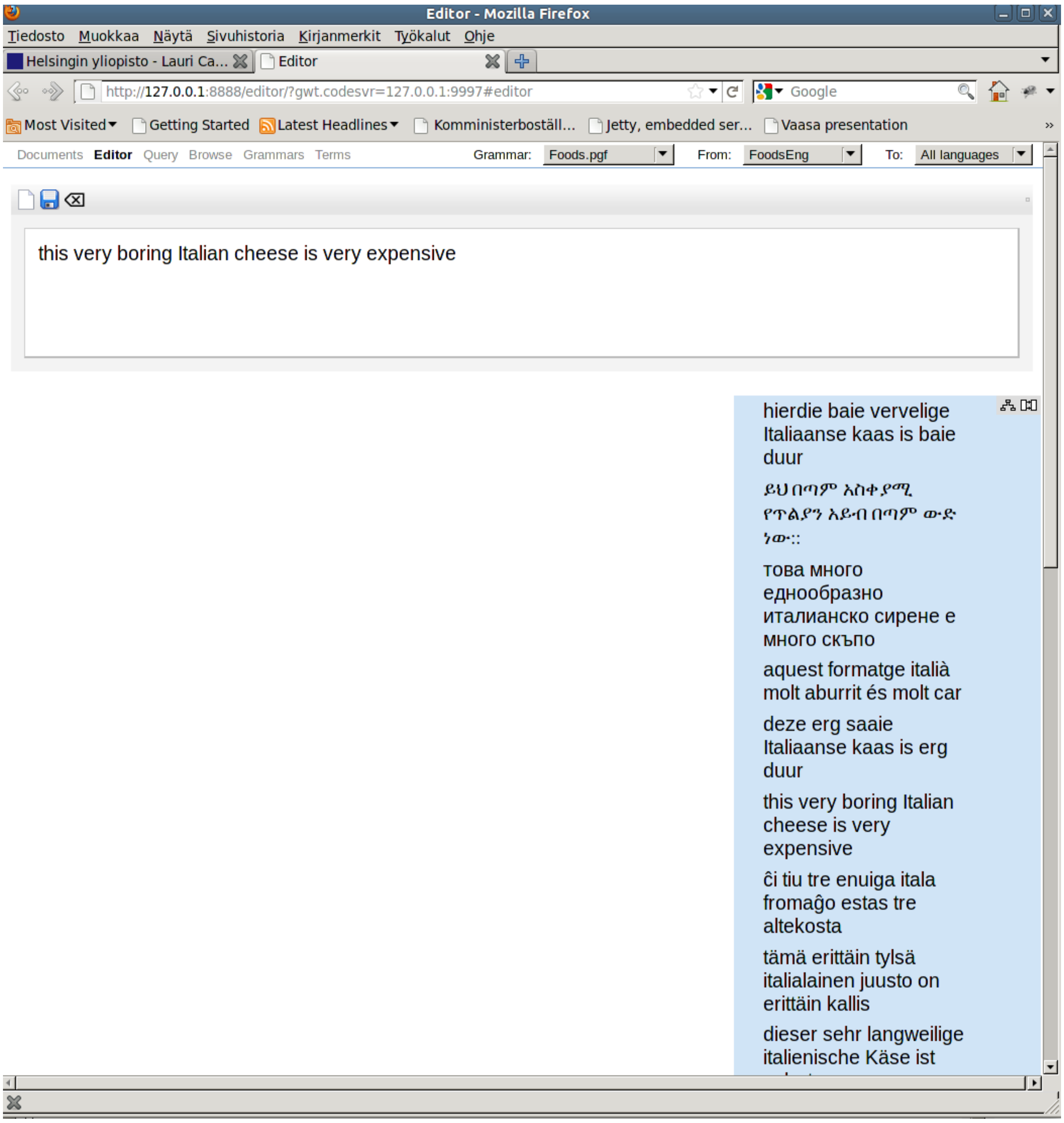
The MOLTO Translation Tools Core API

The core API basically provides for the one-editor/translator scenario, where an editor/translator creates or edits a source document under constraints of a selected GF grammar in PGF form and generates translations for the source. For lexical gaps (out-of-vocabulary items) there is a simple term editor which allows looking up concepts and adding equivalents.

The demo prototype translation editor

This section describes the translation editor developed by K. Angelov at UGOT.

To guide the development of a suitable translation editor API to support MOLTO translation needs, UGOT has created a prototype web-based translation editor. It is implemented in Google Web Toolkit. It is usable for authoring with small multilingual grammars. It doesn't require any downloads or use of command shells. All that is needed is a reasonably modern web browser.



Editor - Mozilla Firefox

http://127.0.0.1:8888/editor/?gwt.codesvr=127.0.0.1:9997#editor

Documents Editor Query Browse Grammars Terms Grammar: Foods.pgf From: FoodsEng To: All languages

this very boring Italian cheese is very expensive

hierdie baie vervelige Italiaanse kaas is baie duur

ይህ በጣም አስቀያሚ የጥልያን አይብ በጣም ውድ ነው።

това много еднообразно италианско сирене е много скъпо

aquest formatge italià molt aburrit és molt car

deze erg saaie Italiaanse kaas is erg duur

this very boring Italian cheese is very expensive

ĉi tiu tre enuiga itala fromaĝo estas tre altekosta

tämä erittäin tylsä italialainen juusto on erittäin kallis

dieser sehr langweilige italienische Käse ist

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.

Sign in

Signing in should allow a user controlled access to her own and some (maybe not all) shared resources. 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.

For basic sign in needs, the demo editor currently uses the Google authentication API.

> Web applications that need to access services protected by a user's Google or Google Apps (hosted) account can do so using the Google Authentication service. This service lets web applications get access without ever handling their users' account login information. Google offers two libraries for handling authentication: one using the OAuth open standard, and a second interface called AuthSub, developed prior to the release of the OAuth standard. 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.

Grammar manager

The demo editor has a simple grammar manager that retrieves the user's grammars from a mysql database via a ContentService implemented in Haskell, subject to a successful login through Google.

Available operations in ContentService:

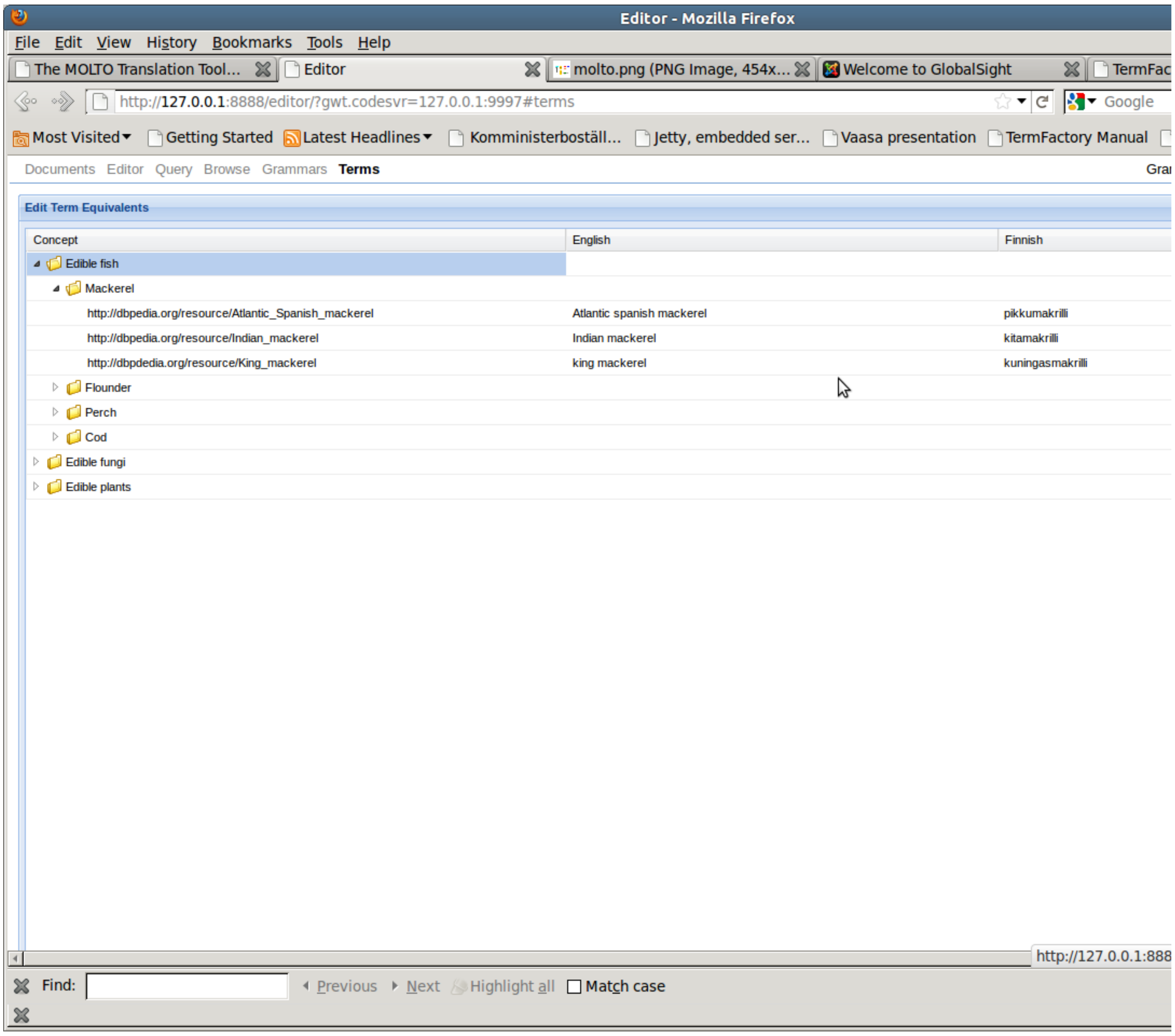
- login
- update_grammar
- delete_grammar
- grammars (listing)
- save
- load (document from mysql db)
- search
- delete

Document manager

The demo editor has a simple file database manager that uploads and requests the user's documents from a mysql database using the same ContentService as the grammar manager.

Term manager

The demo editor has a simple treegrid editor for searching and editing translation correspondences from the web of data, including TermFactory services. It is not yet connected to the GF grammar back end. The management of lexical resources and ontologies is detailed in connection with the extended API below.



Concept	English	Finnish
Edible fish		
Mackerel		
http://dbpedia.org/resource/Atlantic_Spanish_mackerel	Atlantic spanish mackerel	pikkumakrilli
http://dbpedia.org/resource/Indian_mackerel	Indian mackerel	kitamakrilli
http://dbpdedia.org/resource/King_mackerel	king mackerel	kuningasmakrilli
Flounder		
Perch		
Cod		
Edible fungi		
Edible plants		

Editor

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. 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 prototype gives a first rough idea of how a web based GF translation editor could work. While this editor is implemented in JavaScript and runs entirely in the web browser, we do not expect to create a full implementation of the MOLTO translation tools that runs in the web browser, but let the editor communicate with outside servers, including a TMS server (Globalsight) and a GF server.

The MOLTO Translation Tools Extended API

User management

For more flexibility (as well as vendor independence), an open source LDAP (The Lightweight Directory Access Protocol) based user management implementation can be used. There is one in GlobalSight. It allows distinguishing different roles and user groups, and controlling access to resources by roles.

Document management

The simple document manager of the demo editor will be complemented with a more sophisticated XLIFF based document manager built using the GlobalSight document management API. Document format conversions belong to the day's work in the translation business, and they can be assumed to be handled by the extended dcoument manager, using XLIFF as a fixpoint.

XLIFF (XML Localisation Interchange File Format) is an XML-based format created to standardize localization. XLIFF was standardized by OASIS in 2002. Its current specification is v1.2[1] released on Feb-1-2008. The XLIFF Technical Committee is currently at work on XLIFF 2.0. The specification is aimed at the localization industry. It specifies elements and attributes to aid in localization.

XLIFF cognizant open source editors and localization platforms include

- Benten - an open source XLIFF editor written in Java.
- OmegaT - a cross-platform and open source CAT tool.
- Pootle - a web-based localisation platform.
- Heartsome - a suite of cross-platform CAT (Computer-assisted translation) tools founded on open standards: XLIFF, TMX, TBX, SRX, XML, GMX. It also provides a free Lite version.
- Swordfish III - a cross-platform CAT tool that uses XLIFF 1.2 as native format.
- Virtaal - an open source CAT tool.
- XTM - a highly collaborative web server based CAT environment with extensive support for XLIFF (1.0 through to 1.2) as well as an implementation of the OASIS OAXAL architecture.

Examples of XLIFF Documents

```
Example 1: A simple XLIFF file with strings extracted from a Windows RC file. Here the skeleton (the data needed to reconstruct the origina

<?xml version="1.0" encoding="windows-1252" ?>
<xliff version="1.1" xml:lang='en'>
 <file source-language='en' target-language='fr' datatype="winres"
  original="Sample1.rc">
  <header>
   <skl><external-file href="Sample1.rc.skl"/></skl>
  </header>
  <body>
   <group restype="dialog" resname="IDD_DIALOG1">
    <trans-unit id="1" restype="caption">
     <source>Title</source>
    </trans-unit>
    <trans-unit id="2" restype="label" resname="IDC_STATIC">
     <source>&Path:</source>
    </trans-unit>
    <trans-unit id="3" restype="check" resname="IDC_CHECK1">
     <source>&Validate</source>
    </trans-unit>
    <trans-unit id="4" restype="button" resname="IDOK">
     <source>OK</source>
    </trans-unit>
    <trans-unit id="5" restype="button" resname="IDCANCEL">
     <source>Cancel</source>
    </trans-unit>
   </group>
  </body>
 </file>
</xliff>

Example 2: an XLIFF document storing text extracted from a Photoshop file (PSD file) and its translation in Japanese:

<xliff version="1.2">
 <file original="Graphic Example.psd"
  source-language="en-US" target-language="ja-JP"
  tool="Rainbow" datatype="photoshop">
  <header>
   <skl>
    <external-file uid="3BB236513BB24732" href="Graphic Example.psd.skl"/>
   </skl>
   <phase-group>
    <phase phase-name="extract" process-name="extraction"
     tool="Rainbow" date="20010926T152258Z"
     company-name="NeverLand Inc." job-id="123"
     contact-name="Peter Pan" contact-email="ppan@xyzcorp.com">
     <note>Make sure to use the glossary I sent you yesterday.
      Thanks.</note>
    </phase>
   </phase-group>
  </header>
  <body>
   <trans-unit id="1" maxbytes="14">
    <source xml:lang="en-US">Quetzal</source>
    <target xml:lang="ja-JP">Quetzal</target>
   </trans-unit>
   <trans-unit id="3" maxbytes="114">
    <source xml:lang="en-US">An application to manipulate and
     process XLIFF documents</source>
    <target xml:lang="ja-JP">XLIFF 文書を編集、または処理
     するアプリケーションです。</target>
   </trans-unit>
```

```
    <trans-unit id="4" maxbytes="36">
     <source xml:lang="en-US">XLIFF Data Manager</source>
     <target xml:lang="ja-JP">XLIFF データ・マネージャ</target>
    </trans-unit>
   </body>
  </file>
</xliff>
```

XLIFF is bilingual (each translation unit offers a <source> and a <target> elements). There are however ways to have multilingual XLIFF documents:

- Each <file> element can have different source/target pairs.
- The language of the data in the <alt-trans> element can be different from the main source/target languages. This allows alternative translations coming from a different language. For example: French (fr-FR) proposed translations could be offered when translating into French Canadian (fr-CA), and so forth.

In the GF interlingual model, the source "language" can be the abstract syntax representation of a translation unit.

The above considerations entail some requirements for translation-time document management in the MOLTO Translation tools API:

- Associated to the MOLTO Translation Tools API, there must be tools for extracting XLIFF content documents out of various types of original skeleta and putting translated content back to the skeleton. (These tools are outside of MOLTO proper on the because many such tools already exist and because it is up to the provider of a new document type to also provide XLIFF support for it.)
- There must be methods in the Molto Translation API for extracting raw text from XLIFF source elements, feeding it into GF and inserting the translation into the XLIFF target element. The GF translation API should also have methods for handling XLIFF coded inline tags. The best solution for that could be a special purpose GF grammar, because the correct placement of inline tags can depend on the translation of the content.

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. 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 ontology and term management, we propose to 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 to the treegrid editor for 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.

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 treegrid 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 treegrid 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.

Ontologies

If there is a conceptual gap not easily filled in through the treegrid 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 treegrid 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.

Translation editing

The translation editor demo is a good prototype, but different scenarios and platforms may call for different combinations of its features. One way to go is to extend the demo with further tabs and facilities for CAT tool support. But there is the 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 demo 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.

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.

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. See the last section of this document.

Web services API for the MOLTO Translation Tools

To develop the above outlined web-based translation environment further, or implement other usage scenarios, a web service interface to the MOLTO editor API will be useful. The interface consists of several parts.

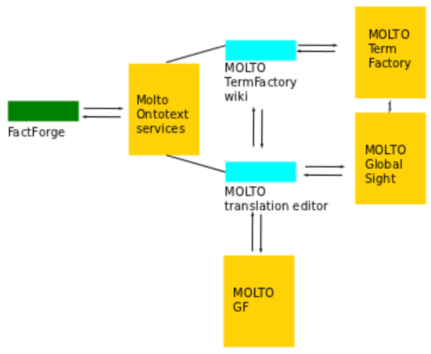
- Translation editor demo
- Globalsight API (adapted for MOLTO)
- TermFactory API (adapted for MOLTO)
- OntoText API
- MOLTO Grammar Tools API
- Translation tools glue

The editor demo is in the MOLTO darcs repository. The services provided by the GF server are outlined in the MOLTO Grammar Tools API document. The GlobalSight WS API was described above. The TermFactory web services are documented in the TF Manual at http://www.helsinki.fi/~lcarlson/CF/TF/doc/TFManual_en.xhtml#Services .

The translation tools glue connects the different parts of the whole. It includes at least:

- treegrid editor back end: answers queries to populate the editor from TF and Ontotext repositories, and communicates user additions to TF and the GF grammar editing services (see next section).
- linking between translation editor/s, translation leveraging tools (TM/termbank) and GF services
- linking between the MOLTO (GlobalSight) TT reviewing system and TermFactory
- linking between the MOLTO (GlobalSight) TT reviewing system and GF grammar editing services
- conversion between TermFactory ontology format and GF abstract grammar format

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



Requirements on the GF grammar and translation APIs

The above design generates a wishlist of requirements 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.