MT Server Land Translation Services

Christian Federmann
DFKI – Language Technology Lab
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, GERMANY
E-mail: cfedermann@dfki.de

Abstract

We demonstrate MT Server Land, an open-source architecture for machine translation services that is developed by the MT group at DFKI. The system can flexibly be extended and allows lay users to make use of MT technology within a web browser or by using simple HTTP POST requests from custom applications. A central broker server collects and distributes translation requests to several worker servers that create the actual translations. User access is realized via a fast and easy-to-use web interface or through an XML-RPC-based API that allows integrating translation services into external applications. We have implemented worker servers for several existing translation systems such as the Moses SMT decoder or the Lucy RBMT engine. We also show how other, web-based translation tools such as Google Translate can be integrated into the MT Server Land application. The source code is published under an open BSD-style license and is freely available from GitHub.

Keywords: Machine Translation, Web Service, Translation Framework, Open-Source Tool

1. Introduction

Machine translation (MT) is a field of active research with lots of different MT systems being built for shared tasks and experiments. The step from the research community towards real-world application of available technology requires easy-to-use MT services that are available via the Internet and allow collecting feedback and criticism from real users. Such applications are important means to increase visibility of MT research and to help shaping the multi-lingual web. Applications such as Google Translate[1] allow lay users to quickly and effortlessly create translations of texts or even complete web pages; the continued success of such services shows the potential that lies in usable machine translation, something both developers and researchers should be targeting.

In the context of ongoing MT research projects at DFKI's language technology lab, we have decided to design and implement such a translation application. We have released the source code under a permissive open-source license and hope that it becomes a useful tool for the MT community. A screenshot of the MT Server Land application is shown in Figure 1.

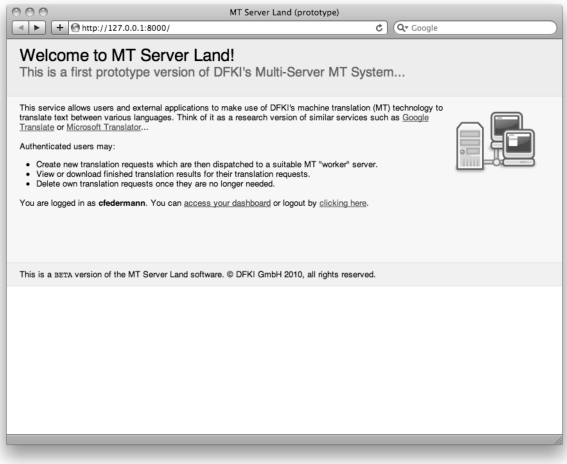

Figure 1: Screenshot of MT Server Land

2. System Architecture

The system consists of two different layers: first, we have the so-called **broker** server that handles all direct requests from end users or via API calls alike. Second, we have a layer of **worker** servers, each implementing some sort of machine translation functionality. All communication between users and workers is channeled through the broker server that acts as a central "proxy" server. An overview of the system architecture is given in Figure 2.

For users, both broker and workers "constitute" the MT

[1] http://translate.google.com

Server Land system; the broker server is the "visible" part of the application while the various worker servers perform the "invisible" translation work. The system has been designed to make it easier for lay users to access and use machine translation technology without the need to fully dive into the complexities of current MT research. Within MT Server Land, translation functionality is available by starting up suitable worker server instances for a specific MT engine. The startup process for workers is standardized using some easy-to-understand parameters for, e.g., the hostname/IP address or port number of the worker server process. All "low-level" work (de-/serialization, transfer of requests/results, etc.) is handled by the worker server instances. Of course, it is possible to design and create new worker server instances, e.g., to demonstrate new features in a research translation system or to integrate other MT systems.

Human users connect to the system using any modern web browser; API access can be implemented using HTTP POST and/or XML-RPC requests. It would be relatively easy to extend the current API interface to support other protocols such as SOAP or REST. By design, all internal method calls that connect to the worker layer have to be implemented with XML-RPC. In order to prevent encoding problems with the input text, we send and receive all data encoded as Base64 Strings between broker and workers; the broker server takes care of the necessary conversion steps. Translation requests are converted into serialized, binary Strings using Google protocol buffer compilation.

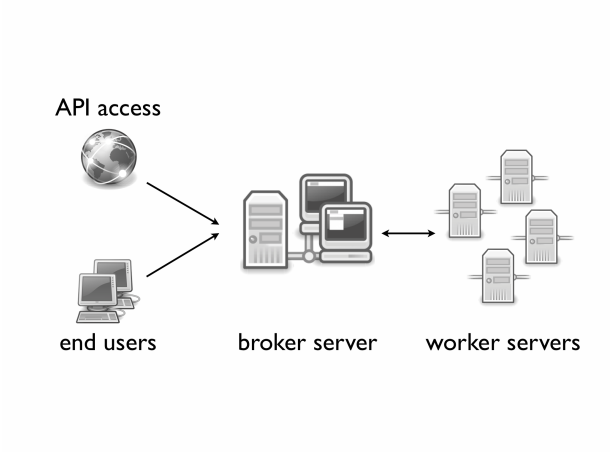

Figure 2: Architecture overview of MT Server Land

2.1. Broker Server

The broker server has been implemented using the **django web framework**[2], which takes care of low-level tasks and allows for rapid development and clean design of components. We have used the framework for other project work before and think it is well suited to the task. The framework itself is available under an open-source BSD-license.

2.1.1. Object Models

The broker server implements two main **django models** that we describe subsequently. Please note that we have also developed additional object models, e.g. for quota management. See the source code for more information.

- **WorkerServer** stores all information related to a remote worker server. This includes source and target language, the respective hostname and port address as well as a name and a short description. Available worker servers within MT Server Land can be constrained to function for specific user and/or API accounts only.
- **TranslationRequest** models a translation job and related information such as the chosen worker server, the source text and the assigned request id. Furthermore we store additional metadata information. Once the translation result has been obtained from the translation worker server, it is also stored within the instance so that it can be removed from the worker server's job queue.

2.1.2. User Interface

We developed a browser-based web interface to access and use the MT Server Land application. End users first have to authenticate before they can access their **dashboard** that lists all known translation requests for the current user and also allows creating new requests. When creating a new translation request, the user may choose which translation worker server should be used to generate the translation for the chosen language pair. We use a validation step to ensure that the respective worker server supports the selected language pair and is currently able to receive new translation requests from the broker server; after successful validation, the new translation request is sent to the worker server that starts processing the given source text.

[2] http://www.djangoproject.com/

Once the chosen worker server has completed a translation request, the result is transferred to (and also cached by) the object instance inside the broker server's data storage. The user can view the result within the dashboard or download the file to a local hard disk. Translation requests can be deleted at any time, effectively terminating the corresponding thread within the connected worker server (if the translation is still running). If an error occurs during translation, the system will recognize this and deactivate the respective translation requests.

2.1.3. API Interface

In parallel to the browser interface, we have designed and implemented an API that allows connecting applications to the MT functionality provided by our application using HTTP POST requests. Again, we first require authentication before any machine translation can be used. We provide methods to list all requests for the current "user" (i.e. the application account) and to create, download, or delete translation requests. Extension to REST or SOAP protocols is possible.

2.2. Worker Server Implementations

A layer of so-called worker servers that are connected to the central broker server implements the actual machine translation functionality. For the MT Server Land, we have implemented worker servers for the following MT systems:

- **Moses SMT**: a Moses (Koehn et al., 2007) worker is configured to serve exactly one language pair. We use the Moses Server mode to keep translation and language model in memory, which helps to speed up the translation process. As the limitation to one language pair effectively means that a huge number of Moses worker server instances has to be started in a typical usage scenario, we have also worked on a better implementation which allows to serve any number of language pairs from one worker instance. Future improvements could be achieved by integrating "on-the-fly" configuration switching and remote language models to reduce the amount of resources required by the Moses worker server.
- **Lucy RBMT**: our Lucy (Alonso & Thurmair, 2003) worker is implemented using a Lucy Server mode wrapper. This is a small Python program running on the Windows machine on which Lucy is installed. We have implemented a simple XML-RPC based API interface to send translation requests to the Lucy engine and later fetch the corresponding results. For integration in MT Server Land, we simply had to "tunnel" our Lucy worker server calls to this Lucy server mode implementation.
- **Joshua SMT**: similar to the Moses worker, we have created a Joshua (Li et al., 2010) worker that works by creating a new Joshua instance for each translation request.

We have also created worker servers for popular online translation engines such as **Google Translate**, **Microsoft Translator**, or **Yahoo! BabelFish**. We will demonstrate the worker servers in our presentation.

3. Acknowledgements

This work was supported by the EuroMatrixPlus project (IST-231720) that is funded by the European Community under the Seventh Framework Programme for Research and Technological Development.

4. References

Alonso, J. A. and Thurmair, G. (2003). The Comprendium Translator System. In Proceedings of the Ninth Machine Translation Summit.

Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C. J., Bojar, O., Constantin, A., and Herbst, E. (2007). Moses: Open Source Toolkit for Statistical Machine Translation. In Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics Companion Volume Proceedings of the Demo and Poster Sessions, pages 177–180, Prague, Czech Republic. Association for Computational Linguistics.

Li, Z., Callison-Burch, C., Dyer, C., Ganitkevitch, J., Irvine, A., Khudanpur, S., Schwartz, L., Thornton, W., Wang, Z., Weese, J., and Zaidan, O. (2010). Joshua 2.0: A Toolkit for Parsing-based Machine Translation with Syntax, Semirings, Discriminative Training and other Goodies. In Proceedings of the Joint Fifth Workshop on Statistical Machine Translation and MetricsMATR, pages 133–137, Uppsala, Sweden. Association for Computational Linguistics.