MT Server Land
DFKI LT’s open-source MT network architecture
Christian Federmann, Andreas Eisele

Overview

- Python-based network architecture for MT
- central “broker server” dispatches requests
- distributed “worker servers” handle MT tasks
- Browser-based access for end users
- API access for integration into custom apps
- Open-source project hosted at GitHub
Our Vision

Motivation

- Make MT from ongoing research accessible to everyone
- Build up a shared MT infrastructure for our projects at DFKI’s LT lab
- Allow easy translation using multiple MT engines and/or configurations
- Connect to external applications
Core Requirements

- Single entry point to multiple MT engines for multiple users
- Many language pairs, multiple engines per pair
- Simple web-based access and APIs

Important Features

- Scalability via distributed implementation
- Robustness wrt. failures in all modules
- Keep administrative effort low
- Management of user roles and privileges
Advanced Functionality

• Give access to intermediate results
• Allow fine-grained influence on behaviour of MT engines
• Make auxiliary processing steps (segmentation, normalisation) accessible via uniform interface
• Support needs of interactive translation, incremental training, and other hot topics of ongoing research

System Architecture
System Architecture

API access

end users → broker server → worker servers

End User Access

- Browser-based interface
- Password protected
- Allows to create new, view finished or delete translation requests
- Results downloadable
- Implemented in django
- Hosted using lighttpd
API Access

- Token-based authentication for security
- Uses HTTP connections (GET, POST, DELETE)
- Several export formats (JSON, YAML, XML)
- Can be used with non-Python frameworks
- It is possible to throttle access to functions
- Uses Google protocol buffer serialization

API Access, cont’d

- API methods either available directly from the django application via HTTP calls
- Or via an XML-RPC service wrapper
- We also plan to extend the export formats to include protocol buffer messages (as these are used anyway by the application)
- Implemented in dashboard/api
Object Models

- Defined in dashboard/models.py and dashboard/api/models.py
- 2 central models:
  - WorkerServer, models an external worker server that exports MT functionality via XML-RPC
  - TranslationRequest, models a translation request, including related information
Worker Servers

- WorkerServer implementation includes information on supported language pairs and status methods (is_alive, is_busy...)

- Translation requests are serialized into a Google protocol buffer “message” which allows for easy serialization of data

- Our .proto definition contains request id, source/target language, source/target text and additional “packet data”

Translation Requests

- TranslationRequest implementation allows to create a translation “job” on a suitable worker server

- We first generate an “underspecified” protocol buffer and send the serialized data to the worker server

- All communication relies on base64 encoded, serialized protocol buffers

- .message files: “backups” in case of crashes
Translation Request Messages

- Each TranslationRequest first generates a so called “Translation Request Message”
- TRMs encode request id, source/target languages, source text and (once ready) the final translation
- Each TRM can also have (optional) “packet data”, a list of key→value pairs which may encode additional data obtained from the translation worker server

Current State of Things
Supported MT Systems

We have implemented worker servers for:

- Google Translate (all language pairs!)
- Microsoft Translator
- Yahoo! Babelfish
- Lucy RBMT (output includes parse trees!)
- **Moses SMT →** we have a related project

Get the source code!

Source code is freely available from [github](http://github.com/cfedermann/mt-serverland)

- [http://github.com/cfedermann/mt-serverland](http://github.com/cfedermann/mt-serverland)

Includes bug tracker, wiki, documentation. We will be happy to include your code extensions!

Happy branching!
Conclusion

We have implemented a MT server network:

- with central access for users and API calls
- worker servers for many different systems
- flexible object models allow easy extension
- system plays nicely with other frameworks
- open source development envisaged!

Thank you!

Any questions or comments?!
Publication
