CLAM: Computational Linguistics Application Mediator

Maarten van Gompel

23-03-2011

ILK Research Group
Induction of Linguistic Knowledge.

Tilburg University

Introduction

Observation

There are a lot of specialised command-line NLP tools available.

Problems

1. Tools often available only locally, installation and configuration can be time and resource consuming
2. **Human aspect:** Not very user-friendly for the untrained general public or technically-challenged researchers (aka Linguists)
3. **Machine aspect:** How to connect one tool to another? How to communicate with a tool in a uniform fashion?

Solutions

Human aspect: Make NLP tools available as a web application.
Machine aspect: Make NLP tools available as a full-fledged webservice.

Advantages

1. Services are available over the web.
2. User-friendly web application provided for human end-users
3. Uniform interface for users (webapp) and machines (webservice)
4. Great for demo purposes
5. Multiple webservices can be chained in a workflow

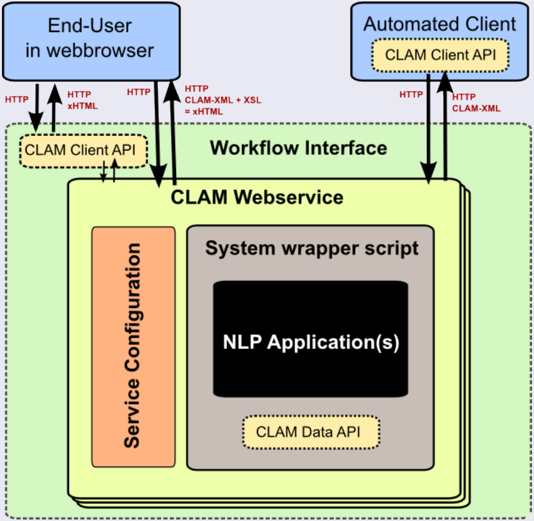
Challenges

1. NLP tasks time consuming: service may run for days before yielding result
2. NLP tasks on large data collections
3. Handling of metadata descriptions
4. Webervices have to be fully deterministic
5. Establishing general interfaces for both humans and machines

Our Focus

1. A *simple* and *universal* approach: *wrapping*
   - Turn almost *any* NLP tool into a webservice with *minimal effort*
   - NLP tool = Given input files and a custom set of parameters, produce output files
   - No need to alter the tool itself, just describe its behaviour
   - Simple, yet powerful enough to deal with complex setups
   - Maximum flexibility
2. Machine-parsable interface & Human-friendly interface

Wrapping Approach



Resource oriented

- Project
  - Input files
    - Per-file parameter selection (=metadata)
  - Global system-wide parameter selection
  - Output files

Example

Project example: User wants to PoS-tag a corpus and starts a project for it

Input: The untagged corpus

Output: The tagged corpus

Technical Details

RESTful Webservice

RESTful Webservice (as opposed to SOAP, XML-RPC)

1. Resource-oriented: "Representations" of "resources" (projects)
2. Using HTTP verbs
3. Lightweight
4. Returns human-readable, machine-parseable XML adhering to a CLAM XML Scheme Definition
5. User authentication in the form of HTTP Digest Authentication

Python

Written entirely in Python 2.5

1. NLP tools, wrapper scripts, and clients may be in any language
2. But: Readily available API when writing wrapper scripts and clients in Python.
3. Built on web.py, runs standalone and out-of-the box with built-in CherryPy webserver

Built-in User Interface

User interface automatically generated from XML using XSLT (in browser)

1. Webservice *directly* accessible from webserver
2. Web 2.0 interface: xHTML Strict, jquery (javascript), AJAX, CSS

Setup

CLAM Setup

Projects are the main resources, users start a new project for each experiment/batch.

Three states:

- **Status 0)** Parameter selection and file upload
- **Status 1)** System in progress
  - Actual NLP tool invoked at this stage only
  - Users may safely close browser, shut down computer, and come back later in this stage
- **Status 2)** System done, view/download output files

Providing a Service (1/2)

In order to make a webservice:

1) Write a service configuration file

- General meta information about your system (name, description, etc..)
- Definition of global parameters accepted by your system (i.e. the wrapper script around your NLP tool)
- Definition of *profiles*
  - A profile defines in detail what output a system produces given a certain input.

Providing a Service (2/2)

In order to make a webservice:

2) Write a wrapper script for your system

- Wrapper script is invoked by CLAM, and should in turn invoke the actual system
- Acts as glue between CLAM and your NLP Application.
- Can be written in any language (python users may benefit from the CLAM API)
- Not always necessary, NLP applications can be invoked directly by CLAM as well.

Profiles

Profiles define...

- **... what output files are produced given which input files**
- ... what metadata parameters are required or possible on input files
- ... how metadata fields are propagated from input files to output files
- ... what viewers are associated with output files (for webapplication)
- ... which converters can act upon input/output files (for webapplication)

Profiles

Profiles define what output files are produced given which input files

- Input Templates
- Output Templates
    - An output template may be conditional on global parameters

Metaphore:

Example

Profile examples:

1. A machine translation system:
   - **Input Template:** The input text in source language X which is to be translated
   - **Output Template:** The translated text target language Y
2. A simple lexicon-based spelling correction system:
   - **Input Template:** The input text which is to be corrected
   - **Input Template:** A lexicon
   - **Output Template:** The corrected text

Writing a Wrapper Script

T

ypical layout of a wrapper script:

1. Read command line arguments (argv) set by CLAM
    - Typical arguments are: Input Directory, Output Directory, Clam XML file
2. Parse Clam XML file (easy using CLAM Data API)
3. Read user-set parameters and iterate over input files, do whatever you need to do
4. Invoke your NLP tool (system call)

Writing a Client to connect to an existing service

1. Communicate with service over HTTP, using HTTP verbs on projects and files to effectuate state transfers
   - `GET /` - List all projects
   - `GET /{project}/` - Get a project's current state (CLAM XML)
   - `PUT /{project}/` - Create a new empty project
   - `POST /{project}/` - Start a project with POSTed data as parameters
   - `DELETE /{project}/` - Delete or abort a project
   - `POST /{project}/input/{filename}` - Upload input file
   - `GET /{project}/output/` - Download all output files as archive
   - `GET /{project}/output/{filename}` - Download output file
2. Check HTTP return codes and parse XML responses
3. **Python users benefit from CLAM Client API, taking**