CLAM: Computational Linguistics Application Mediator

Maarten van Gompel

20 Mei 2010

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 tough
2. Not very user-friendly for the untrained general public or technically-challenged researchers (aka Linguists)
3. How to connect one tool to another?

Solution

Making NLP tools available as a full-fledged webservice.

Advantages

1. Services available over the web.
2. User-friendly interface built-in in the webservice
3. Great for demo purposes
4. Multiple webservices can be chained in a workflow

Our Focus

1. A *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
2. Machine-parsable interface & Human-friendly interface

Wrapping Approach

1. NLP application: blackbox
2. Wrapper script
3. CLAM Webservice

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 runs 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

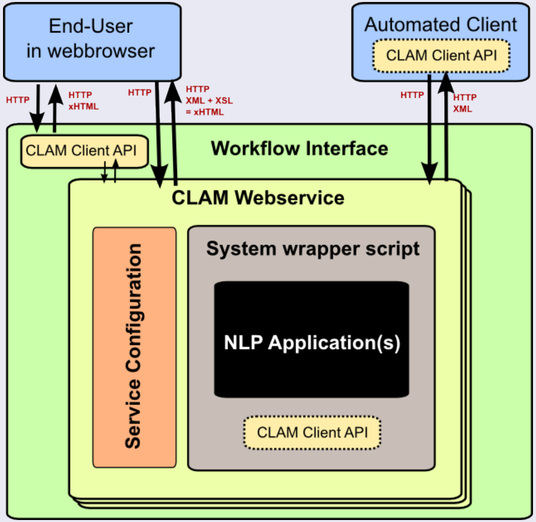
In order to make a webservice:

1. Write a service configuration file (in Python, but no Python experience required).
    - General meta information about your system (name, description, etc..)
    - Definition of parameters accepted by your system/wrapper script
    - Definition of input formats and output formats
    - Definition of users and authentication method
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.

Writing a Client

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 state
   - `PUT /project/` - Create a project
   - `POST /project/` - Start a project with POSTed data as parameters
   - `DELETE /project/` - Delete or abort a project
   - `POST /project/upload/` - Upload a file
   - `GET /project/output/` - Download all output files as archive
   - `GET /project/output/file` - Download output file
2. Parse XML responses
3. **Python users benefit from CLAM Client API, taking care of all above communication and response parsing!**

Architecture


End-User in webbrowser
Automated Client
CLAM Client API
HTTP
HTTP xHTML
HTTP
HTTP XML + XSL = xHTML
HTTP
HTTP XML
CLAM Client API
Workflow Interface
CLAM Webservice
Service Configuration
System wrapper script
NLP Application(s)
CLAM Client API