Architecture of a Web-Based Predictive Editor for Controlled Natural Language Processing

Stephen Guy and Rolf Schwitter

Department of Computing,
Macquarie University,
Sydney, 2109 NSW, Australia
{Stephen.Guy,Rolf.Schwitter}@mq.edu.au

Abstract. In this paper, we describe the architecture of a web-based predictive text editor being developed for the controlled natural language PENG^{ASP}. This controlled language can be used to write non-monotonic specifications that have the same expressive power as Answer Set Programs. In order to support the writing process of these specifications, the predictive text editor communicates asynchronously with the controlled natural language processor that generates lookahead categories and additional auxiliary information for the author of a specification text. The text editor can display multiple sets of lookahead categories simultaneously for different possible sentence completions, anaphoric expressions, and supports the addition of new content words to the lexicon.

Keywords: controlled natural language processing, predictive editor, web-based authoring tools, answer set programming.

1 Introduction

Writing a specification in a controlled natural language without any tool support is a difficult task since the author needs to learn and remember the restrictions of the controlled language. Over the last decade, a number of different techniques and tools [3,5,12,13] have been proposed and implemented to minimise the learning effort and to support the writing process of controlled natural languages. The most promising approach to alleviate these habitability problems is the use of a predictive text editor [13,17] that constrains what the author can write and provides predictive feedback that guides the writing process of the author. In this paper, we present the architecture of a web-based predictive text editor being developed for the controlled natural language PENG^{ASP}[15]. The text editor uses an event-driven Model-View-Controller based architecture to satisfy a number of user entry and display requirements. These requirements include the display of multiple sets of lookahead categories for different sentence completions, the deletion of typed words, the addition of new content words to the lexicon and the handling of anaphoric expressions. Additionally, the text editor displays a paraphrase for each input sentence and displays the evolving Answer Set Program [11].

B. Davis et al. (Eds.): CNL 2014, LNAI 8625, pp. 167–178, 2014.

2 Overview of the PENG^{ASP} System

2.1 Client-Server Architecture

The PENG^{ASP} system is based on a client-server architecture where the predictive editor runs in a web browser and communicates via an HTTP server with the controlled natural language processor; the language processor uses in our case an Answer Set Programming (ASP) tool as reasoning service (Fig. 1):

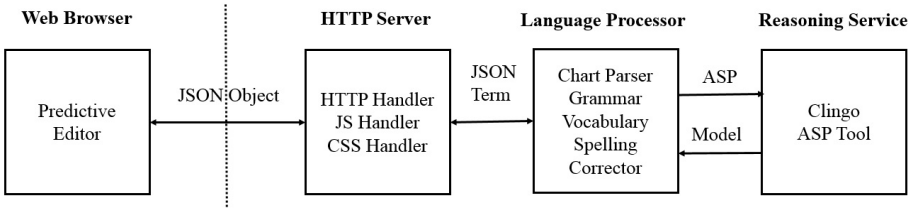

Fig. 1. Client-Server Architecture of the PENG^{ASP} System

The communication between the predictive editor and the HTTP server occurs asynchronously with the help of AJAX technologies and by means of JSON[1] objects. The predictive editor is implemented in JavaScript[2] and JQuery[3]. The HTTP server as well as the controlled natural language processor are implemented in SWI Prolog[4]. The Prolog server translates JSON objects into JSON terms and vice versa so that these terms can be processed directly by the language processor. The language processor incrementally translates the controlled language input via discourse representation structures [8] into an ASP program and sends this ASP program to the ASP tool *clingo* [6,7] that tries to generate one or more satisfiable answer sets for the program.

2.2 HTTP Server

SWI-Prolog provides a series of libraries for implementing HTTP server capabilities. Our server is based on this technology and can be operated as a stand-alone server on all platforms that are supported by SWI-Prolog. The following code fragment illustrates how an HTTP server is created, a port (`8085`) specified, and a request (`Request`) dispatched using a handler registration (`http_handler/3`):

```
server(Port) :- http_server(http_dispatch, [port(Port)]).
:- http_handler('/peng/', handle, []).
handle(Request) :- ...
:- server(8085).
```

[1] `http://json.org/`
[2] `http://www.ecmascript.org/`
[3] `http://jquery.com/`
[4] `http://www.swi-prolog.org/`