NL2PDDL: A Conversational Interface for Model Generation and Iteration

Kshitij P. Fadnis and **Kartik Talamadupula**
IBM Research
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598
{kpfadnis, krtalamad} @ us.ibm.com

Abstract

Although the automated planning community has seen many advances to planning techniques in the past decade, domain model creation and maintenance has remained the central bottleneck preventing wider adoption of planning technology in the real world. While there has been some work on learning these models in an automated fashion, there has been very little focus on user-friendly interfaces for the creation, querying, and editing of planning models. In this demonstration, we present a novel approach to interfacing with planning models using natural language via a conversation modality. We detail the construction of the system and demonstrate its capabilities in a short attached video.

1 Introduction

In recent years, automated planning techniques and systems have achieved an impressive scale-up in terms of the size of the problems that they can handle. Planners such as Fast Downward (Helmert 2006) and its descendants routinely solve problem instances of real world size in a matter of seconds. Unfortunately, this scale-up has not led to a significant increase in the adoption of planning techniques for real world applications. One of the main reasons for this is the extremely cumbersome task of specifying domain models for planning tasks. The most widely accepted specification language – PDDL (Mcdermott et al. 1998) – and its variants are still fairly complex and have a steep learning curve. This complexity of PDDL is a necessary evil, since the language also needs to be expressive enough to model real domains. Indeed, much planning work over the past two decades has focused on this tension between increased expressivity on the one hand, and planner efficiency on the other. Rather than focus on this known problem, planning practitioners should instead look at the other, much narrower end of this real world bottleneck for planners – the modeling and specification of domains.

In this demonstration, we introduce the NL2PDDL system, whose ultimate form is intended to make the specification of planning domains – agnostic of representation language – easier for subject matter experts (SMEs) who are not familiar with planning technology. Our system introduces, for the first time, the medium of conversation (and dialog) as the backing technology for the creation and maintenance of planning models.

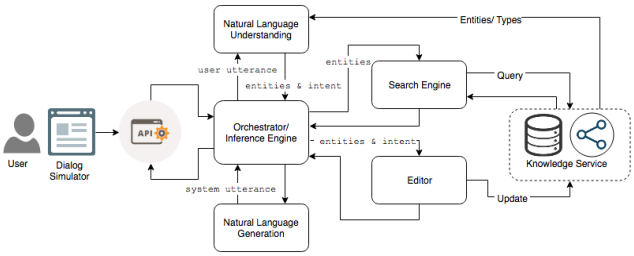

Figure 1: The architecture of the NL2PDDL system.

2 System Architecture

In this section, we briefly describe the architecture of the NL2PDDL system, outlined in Figure 1. We follow a modular design, where each component of the system can be accessed through a simplified set of APIs. There are six major components in the system:

- **Natural Language Understanding (NLU)**: The job of the NLU component is to process a text utterance, in order to identify the entities and intents present in it. For example, on the input "show me some actions in this domain", the output will be the entity "actions" and the intent "inform". We employ lexical and fuzzy matching to explore a given input for valid entities, while intent is identified using the IBM Watson Natural Language Classifier (NLC) (Yates 2015). The NLC is trained with ten different intents, which broadly cover the space of interaction this system is designed to handle.[1] One of the key features of our NLU is that it restricts its entity matching to a set of concepts and their paraphrases from the domain of interest. This reduces the scope of the entity detection, which in turn provides both speed-up and an accuracy boost.
- **Search Engine**: The search engine is primarily responsible for retrieving knowledge about the entities that are present in the text input. This component formulates queries based on the available entities and their sub-fields.
- **Knowledge Service**: This service is the internal memory of the system. It reads in an existing planning domain

[1]The data that we used to train this classifier will be made available, along with the entire source code.

(when available) and builds a graphical representation of the concepts in that domain. Each type, predicate, and action are considered a separate node in this graph; the edges reflect the associations between them. For example, an action `X` with precondition `y` will feature an edge from the `x` node to the `y` node, with the "precondition" relation. This graphical representation allows for easier query and update mechanisms over planning domains. The knowledge service also bootstraps the NLU with concepts associated with the domain of interest.

- **Editor**: The Editor is responsible for edits to the model that are requested by a user. These include things like the addition or removal of type definitions, the addition or removal of predicates from the add or delete effects of an action, and so on. The Editor retrieves the entities identified by the NLU and sends the appropriate update request to the Knowledge Service. The Editor is currently intended to support simple edit capabilities like the removal of types, predicates, or actions by name; and the addition of new types. We are in the process of extending the Editor's functionality to handle more complex operations like action editing.

- **Natural Language Generation (NLG)**: The NLG is a critical component that differentiates `NL2PDDL` from previous model editors, with its ability to produce conversation. It enables feedback in natural language, which makes interaction with the system more natural. We currently use a rule-based system with predefined response classes; however, our aim is to replace it with a generative model that delivers more human-like responses.

- **Orechestrator/Inference Engine:** This component is the brain of the system – all communication flows through it. Messages in the system are tagged with their source at the point of their origin. The job of the orchestrator is to determine where the message should be sent next, based on where it came from and what it contains. For example, a message originating from the Search Engine is tagged with that source; when it arrives at the orchestrator, it is redirected to the NLG in order to generate an appropriate response based on the search results. Our orchestrator is similar to the one used by (Feng et al. 2018).

On the platform side, the knowledge service is a standalone RESTful service deployed as a Docker container in a Kubernetes cluster. All the information serviced by the system is backed up using a persistent storage mechanism.

3 Demonstration

The `NL2PDDL` system currently includes a web-based interface, which can be used to interact with the system and explore a planning model. A live demonstration of our system and this interface can be viewed at the following URL:

`ibm.biz/nl2pddl`

Please see Section 4 (Appendix) for a description of the interactions that are currently supported by the interface.

References

Feng, S.; Gunasekara, R. C.; Shashidhara, S.; Fadnis, K. P.; and Polymenakos, L. C. 2018. A unified implicit dialog framework for conversational search. *AAAI 2018 Systems Demonstrations*.

Helmert, M. 2006. The fast downward planning system. *J. Artif. Int. Res.* 26(1):191–246.

Mcdermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL - the planning domain definition language. Technical Report TR-98-003, Yale Center for Computational Vision and Control,.

Yates, R. 2015. Introducing the IBM Watson Natural Language Classifier. *IBM developerWorks/Developer Centers, posted Jul* 10.

4 Appendix

We use this Appendix to advise readers on the current (limited) expressiveness of the NLU component. The NLU handles spelling mistakes in the name of concepts (types, predicates, actions) as well as certain grammatical mistakes; this flexibility exists as long as syntactic parsing and part-of-speech tagging are still possible on the input sentence. Please also note that words like `named`, `addeffects` and `argument` are necessary for conditional statements in order to uniquely identify appropriate nested concepts. It is also crucial that proper nouns like action, type or predicate names be capitalized, due to the current limitations of the syntactic parser and the part-of-speech tagger. To assist readers, we list a few typical queries that are currently supported by the system:

- **What-type questions** regarding top-level concepts:
  - `What are types available in this domain?`
  - `What are actions in this domain?`
  - `What are available predicates?`
- **What-type** and **To be-type questions** with a condition:
  - `Is there action named <NAME>?`
  - `What are actions with addeffects <PREDICATE-NAME>?`
  - `Is there a predicate with argument <TYPE-NAME>?`
  - `What are all actions with preconditions <PREDICATE-NAME>?`
- **Greeting statements**: Simple greetings to initiate the conversation.
  - `hello`
  - `hi`
  - `good morning`
  - `good evening`
- **Closing statements**: Simple utterances to end the current conversation.
  - `thank you`
  - `no. that will be it.`
  - `i am all set`
  - `nm. i am done.`
  - `thanks`

Video A link to the storyboard for the demo is available at the following URL: `ibm.biz/nl2pddl-story`