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 work, we present a novel approach to interfacing with planning models using natural language via a conversation modality. We detail the construction of the system, its capabilities, as well as some key opportunities and challenges that are brought up by this unique interface with planning models.

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. Even today, this is more of a dark art than a science, and it is handled by a small group that can afford to spend the time needed to truly straddle two realms. One is the world of planning research; the other is that of the real world application. Only such people are currently able to identify the correct abstraction level needed to convert an application into a decision making problem, and then model that abstraction into PDDL. This has long remained the elephant in the room for the planning community: an inconvenient truth that gets truly discussed and debated only in workshops and systems demonstrations.

In this paper, 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. The reasons for using this medium are straightforward, and we list them here.

Conversation as a Medium for Planning One of the main reasons for using conversation is that expressing new ideas and specifications is easier through natural language, and certainly more natural than filling out cumbersome forms based on the rules of a syntax. Besides, while using conversation, the burden of translating an utterance into the representation language's syntax falls on the authoring system, and not on the SMEs who are using the system. This lends itself to easy automation of the translation from natural language to the desired representation, while ensuring that the cognitive burden of learning yet another new interface is non-existent. Conversation also provides a natural avenue for two-way – and if needed, multi-way – interaction. The system can prompt, clarify, and inform the user when necessary; this can even be done without explicit tasking, as the agent recognizes mistakes being made in the authoring. Such an interactive process – conducted through dialog – is also very educational, and can be used to teach the process and pitfalls of domain modeling to an SME who is a novice when it comes to PDDL.

Finally, the use of conversation gives the system a stored history of the domain authoring process. While version control systems can track model updates just as well, there is a fundamental difference in terms of the information that is stored: version control techniques are restricted to the syntactic information alone, while the conversation history can capture the SME's interaction in its entirety. A good domain modeling assistant can use this rich history from past conversations to ensure that the modeling process is on the right track. The conversation thus serves as a natural, human-readable log of the modeling process: this can be used when the domain needs to be augmented, or a similar

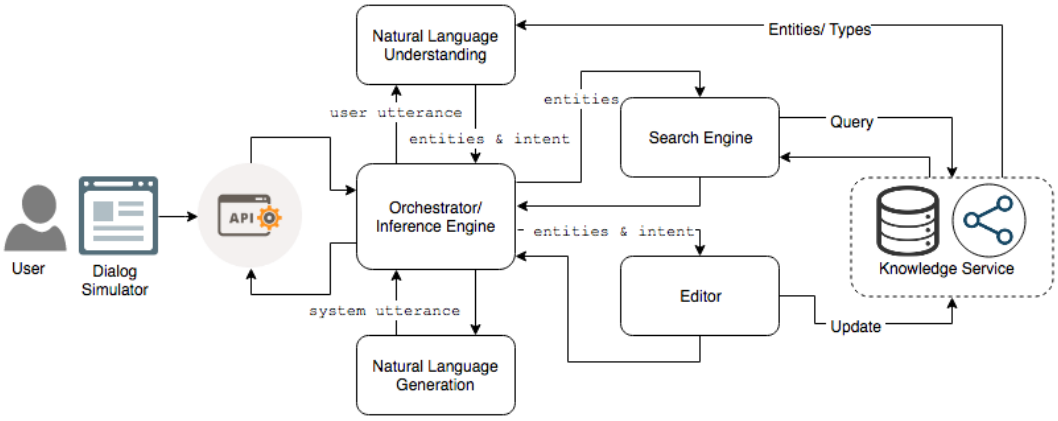

Figure 1: The architecture of the NL2PDDL system.

domain modeled sometime in the future. The conversation history is also a valuable resource to train language models for better prompt generation and interaction.

In this paper, we detail the first steps towards realizing a conversational assistant for domain modeling tasks via the NL2PDDL system. We first look at related work in the field of knowledge engineering for planning and scheduling (KEPS), and point out differences and synergies with our contribution. We then describe the architecture of the NL2PDDL system and its constituent components, as well as a specific web-based implementation of a minimum viable product. This is followed by a demonstration of the system's current capabilities, and a discussion about existing limitations and future work. This paper describes the first iteration of an ambitious system, and we ask that it be seen as a report on work in progress.

2 Related Work

Over the past decade or so, the planning community has embraced the difficulty as well as the importance of the knowledge engineering task; and the bottleneck posed by it for the wider adoption of planning technology. Motivated by McCluskey's early work (McCluskey 2000) and the KEPS (Knowledge Engineering in Planning and Scheduling) series of workshops, this area has received steady attention.

The prior work we highlight in this section can be divided into three distinct categories: theoretical, applications, and tools. Theoretical work focuses on the verification and modeling challenges inherent in the knowledge engineering for planning space, without looking at specific application domains. Application work, on the other hand, combines planning techniques with real world problem domains. Finally, tools offer an implementation of knowledge engineering methods in an overall system, in order to elicit planning models. Our work intersects with all three of these categories, and can be extended in conjunction with many of the prior efforts mentioned here.

Theoretical The backing engine for a tool like NL2PDDL needs to necessarily accommodate functionalities that determine whether a planning model is being built correctly, and in a fashion that will work efficiently with planners. Although the current beta version of our tool does not handle this – and instead places the burden of ensuring model correctness on the user – prior work on testing and verifying that a model satisfies a given set of requirements (Raimondi, Pecheur, and Brat 2009) can be very useful for this extension. Another important issue that needs to be handled is that of domain model configuration (Vallati et al. 2015) – this will eventually determine the computational efficiency of planners that use such a model. Finally, more recent work by McCluskey et al. (McCluskey, Vaquero, and Vallati 2016; 2017) defines measures for planning model elicitation that go beyond mere correctness, and look instead at a host of other metrics including accuracy, consistency, completeness, adequacy, and operationality. All of these ideas can be operationalized in a framework such as VAL (Howey, Long, and Fox 2004), which can then be integrated with NL2PDDL.

Applications Natural language interfaces – of the kind used by the NL2PDDL system – have not been used extensively in conjunction with automated planning work. To be sure, there has been a lot of work in the opposite direction, in applying planning techniques to the problem of dialog management: led by the seminal work of Williams & Young (2007). However, to the best of our knowledge, the work by Yates et al. (Yates, Etzioni, and Weld 2003) is the only work that uses natural language as a medium to automatically generate planning knowledge – in their case, the goals needed for household appliances to carry out their tasks. We believe there is much more promise in this direction. On the model creation and maintenance side, a rather interesting application of planning was the work of Puissant et al. (Puissant, Mens, and Van Der Straeten 2010) on resolving model inconsistencies in software engineering models with the help of automated planning.

Tools Knowledge engineering have vastly improved in the past decade, under the aegis of the twin KEPS workshops and the International Competition on KEPS (ICKEPS). Most modeling tools build up from some notion of ontology all the way to the planning model (Vaquero et al. 2013; Wickler, Chrpa, and McCluskey 2014); while PDDL

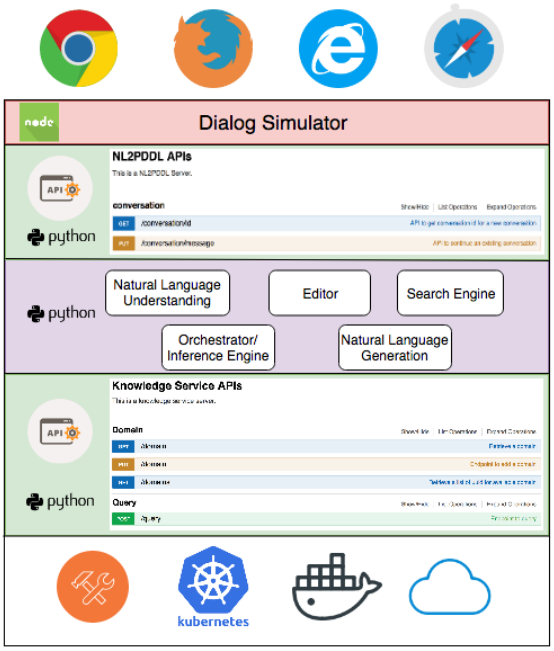

Figure 2: The NL2PDDL system's infrastructure.

Studio (Plch et al. 2012) provides a number of useful features for users who are already familiar with PDDL. A good overview of such tools is provided by Jilani et al. (Jilani et al. 2014). PRIDE (Bonasso and Boddy 2010) supports a complex query structure, but relies heavily on the availability of a domain ontology: a requirement that is self-identified by the authors as non-trivial. PRONTOE (Bell et al. 2013) addresses ontology design by providing a graphical editing tool that allows users to define various ontology kernels that can be used to separate complex systems into their constituent components – however, it is accompanied by a steep learning curve when it comes to usage. itSIMPLE$_{2.0}$ (Vaquero et al. 2013) similarly uses a UML-driven interface for domain modeling, which comes with its own learning curve and limitations. Our work is complementary to many of these tools, in that it can be extended by combining with any (or indeed all) of them. However, the availability of a simple natural language interface will significantly decrease the load on users trying to model novel domains.

The work that is closest in spirit to the NL2PDDL system is that of Sethia et al. (Sethia, Talamadupula, and Kambhampati 2014), which uses the Google API in order to construct a mapping between higher level actions at the PDDL abstraction, and lower level actions that are present in a humanoid robot's internal libraries. However, even this interface is very form-like, and insists upon the user providing information in the same order as that desired by the PDDL syntax. It also does not allow the user to query an existing model.

3 System Architecture

In this section, we 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 Watson Natural Language Classifier (NLC) by IBM. 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 component 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 subfields.
- **Knowledge Service**: This service is the internal memory of the system. It reads in an existing planning domain (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 employ a rule-based system with a few predefined response classes; however, our aim is to replace this with

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

a generative model in order to deliver 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.

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. As seen in Figure 2, the Knowledge Service provides the '/domain' and '/query' endpoints. The remaining components of the system (detailed above) are written in Python. `NL2PDDL` also comes with a built-in web-based chat application written in NodeJS; this is introduced in the next section.

4 Instruction Set

As we have stated earlier in the paper, this is a first-of-a-kind solution that attempts to provide a natural language interface via the conversation modality for planning models. We use this section 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. 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:[2]
  - `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`

[2]Please note that words like `named`, `addeffects` and `argument` are necessary for conditional statements in order to uniquely identify appropriate nested concepts. Proper nouns like action, type, or predicate names must also be capitalized.

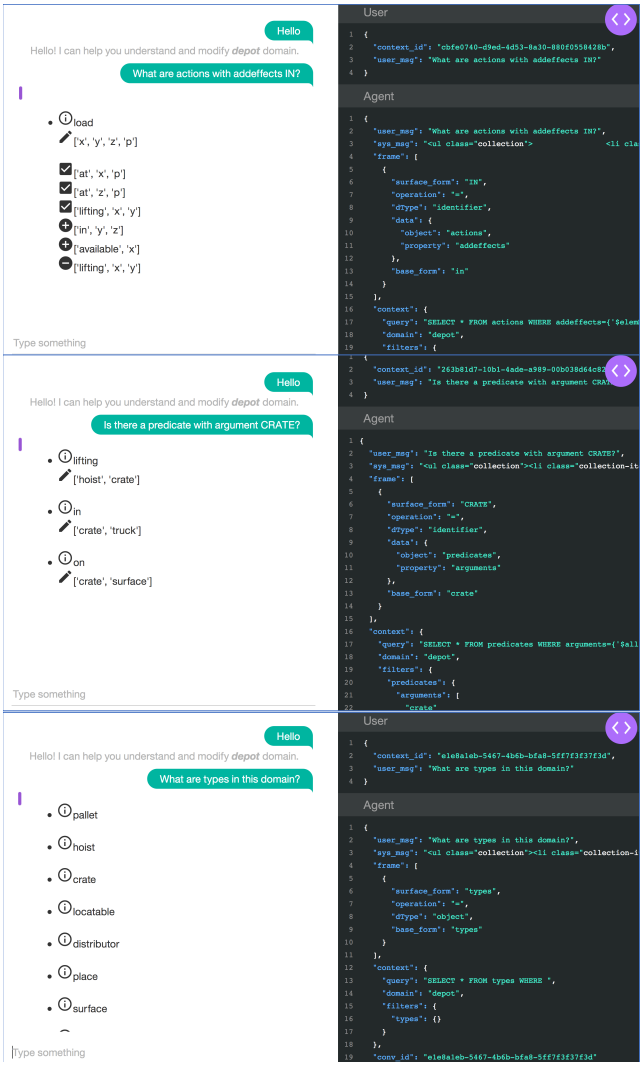

Figure 3: Some examples of queries that are possible on `NL2PDDL`: (a) Action (b) Predicate (c) Types.

5 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 for a description of the interactions that are currently supported.

In this section, we explain the various features of this interface via a few examples. In Figure 3(a), we ask the system to return actions that satisfy a given query pertaining to add effects. The system returns an action ('load') that satisfies this query. The right-hand side pane on the interface allows a user to check on how the system interpreted their utterance. In this particular example, the system identified that an action with an add effect on the 'in' predicate was being requested. Figure 3(b) and Figure 3(c) demonstrate that we currently support different degrees of granularity when inquiring about the domain physics. Figure 3(c) highlights the system's ability to look for major artifacts in the domain definition, like types, predicates, or actions. Figure 3(b), on the other hand, shows an example of narrowing the scope of the query further, in order to identify predicates that take certain argument types.

The system allows for spelling mistakes by using a built-in spelling correction mechanism with a settable parameter for its minimum match threshold. Currently, we set it to require a minimum threshold of 85% spelling match with a suggested correction. Each intent also features eight to ten phrasal variations, granting coverage over a wider range of conversation. Finally, the system currently only supports a limited subset of possible ways to declare an entity. Specifically, we insist that users capitalize the name of an action, predicate, or type fully: for example, 'IN' in Figure 3(a), or 'CRATE' in Figure 3(b).

6 Future Work

As we have stressed throughout this paper, the `NL2PDDL` system is currently the first, work-in-progress version of a much larger vision that involves more natural authoring and maintenance of planning models. There are many avenues of extension, which we list here in no specific order. First, we have partially implemented an 'edit' functionality that complements the 'query' functionality that we demonstrated in this paper. This allows users to add or change things in the domain of interest; we will be deploying this shortly. We are also extending the system so that it can deal with more than just a single proof-of-concept domain (currently Depots). A more challenging future direction involves the integration of `NL2PDDL` with existing domain authoring, maintenance, and verification tools like VAL and itSIMPLE: we would like for the system to provide (automated) intelligent feedback to domain authors. On the natural language and conversation side, we are exploring ways to liberate the system from its current rule-based NLU and NLG implementations, and move to a more natural, automatically learned model of conversation. Finally, we hope to release a more polished interface for demonstration at ICAPS 2018.

References

Bell, S.; Bonasso, P.; Boddy, M.; Kortenkamp, D.; and Schreckenghost, D. 2013. Prontoe a case study for developing ontologies for operations.

Bonasso, P., and Boddy, M. 2010. Eliciting planning information from subject matter experts. In *Proceedings of ICAPS 2010 Workshop on Scheduling and Knowledge Engineering for Planning and Scheduling (KEPS)*, 5–12.

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

Howey, R.; Long, D.; and Fox, M. 2004. Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In *Tools with Artificial Intelligence, 2004. ICTAI 2004. 16th IEEE International Conference on*, 294–301. IEEE.

Jilani, R.; Crampton, A.; Kitchin, D. E.; and Vallati, M. 2014. Automated knowledge engineering tools in planning: state-of-the-art and future challenges.

McCluskey, T. L.; Vaquero, T.; and Vallati, M. 2016. Issues in planning domain model engineering.

McCluskey, T. L.; Vaquero, T. S.; and Vallati, M. 2017. Engineering knowledge for automated planning: Towards a notion of quality. In *Proceedings of the Knowledge Capture Conference*, 14. ACM.

McCluskey, T. 2000. Knowledge engineering for planning roadmap.

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,.

Plch, T.; Chomut, M.; Brom, C.; and Barták, R. 2012. Inspect, edit and debug pddl documents: Simply and efficiently with pddl studio. *System Demonstrations and Exhibits at ICAPS* 15–18.

Puissant, J. P.; Mens, T.; and Van Der Straeten, R. 2010. Resolving model inconsistencies with automated planning. In *Proceedings of the 3rd Workshop on Living with Inconsistencies in Software Development*, 8–14.

Raimondi, F.; Pecheur, C.; and Brat, G. 2009. Pdver, a tool to verify pddl planning domains.

Sethia, S.; Talamadupula, K.; and Kambhampati, S. 2014. Teach Me How To Work: Natural Language Model Updates and Action Sequencing. In *ICAPS 2014 Systems Demonstrations*.

Vallati, M.; Hutter, F.; Chrpa, L.; and McCluskey, T. L. 2015. On the effective configuration of planning domain models. In *IJCAI*, 1704–1711.

Vaquero, T. S.; Silva, J. R.; Tonidandel, F.; and Beck, J. C. 2013. itsimple: towards an integrated design system for real planning applications. *The Knowledge Engineering Review* 28(2):215–230.

Wickler, G.; Chrpa, L.; and McCluskey, T. L. 2014. Kewi: A knowledge engineering tool for modelling ai planning tasks.

Williams, J. D., and Young, S. 2007. Partially observable markov decision processes for spoken dialog systems. *Computer Speech & Language* 21(2):393–422.

Yates, A.; Etzioni, O.; and Weld, D. 2003. A reliable natural language interface to household appliances. In *Proceedings of the 8th international conference on Intelligent user interfaces*, 189–196. ACM.