Knowledge Engineering Tools for Planning in PDDL - Syntax Highlighting, Task Generation and Plan Visualization and Execution - an extensible framework

Volker Strobel

February 27, 2014

Contents

Abstract

Automated planning and scheduling blazes a trail for artificial intelligent behavior. PDDL, a planning language widely used for AI task specifications requires extensive knowledge engineering effort. After giving an overview of PDDL syntax and planning basics, a plug-in (including syntax highlighting) for the code editor Sublime Text and related editors will be given. A type diagram generator will be presented that supports modeling of the environment. The usability of these tools will be evaluated by subjects without prior knowledge in PDDL. By the use of these tool, the average modeling time of a shorter task specification could be improved by ... percent and the error rate could be reduced from ... to ... . Furthermore, this thesis will present approaches for the automatic generation of task specifications and provide a basic interface between the functional programming language Clojure and PDDL. The proposed tools can assist knowledge engineers in the design process.

`https://www.ece.cmu.edu/~koopman/essays/abstract.html`

Chapter 1

Introduction

TODO:

- Scope of the article (What did I miss out, what is included?)
- Name Tools and provide found possibilities and limitations
- Motivation: Why is this article interesting?
- Structure of the article
- Review of relevant literature? Note: I can review the relevant literature and related work in the related chapter, if this is convenient
- Planning Architecture - 3 Tier Architecture
- ICAPS
- Handicapped people -> Planning Interaction
- Why is something interesting? -> reasons! (and not because it does not exist yet)

Knowledge engineering (KE), that means representing world information in a computer system, is the crucial step for utilizing AI planning system to solve problems. By definition, it requires an expert that knows the underlying syntax and models the information manually. This process is naturally error-prone and time-consuming. While planning system are improving steadily, the main work for useful systems lies on the representing language. The Planning Domain Definition Language (PDDL) (McDermott et al. 1998) allows for a standardized way of specifying planning tasks. While on the one hand, recent PDDL extensions (Fox and Long 2003; Kovacs 2011) extended the expressiveness of PDDL and tackle a route for real-world applications, they also demand a higher level of knowledge and attention of the knowledge engineer. Several approaches to shift the modeling process from a text based 'programming paradigm' to a user-friendly graphical design tool exist, however, they also arouse drawbacks: limited functionality, expenditure of time, editing difficulty, to name a few. As in other computer languages, so far, there is no real way around diving into the 'code paradigm'. This thesis will, after introducing the basic terms and definitions of PDDL, focus on the software support for knowledge engineers. A package for the text and source code editor Sublime Text (ST) will be introduced, that provides syntax highlighting and PDDL templates. And the functionality is transferred to a on-line editor with instant access.

Planning is one the the classic Artificial Intelligence (AI) tasks. The main focus of this thesis is about real world applications and the development of handy tools that support (and partially automatize) the planning process.

Chapter 2

Related Work

2.1 PDDL Studio

2.2 VISPlan

2.3 Emacs PDDL Mode

2.4 itSimple

Chapter 3

Planning Basics and PDDL

- Brief summary at start
- Start with a paragraph that describes the context
- Very interesing for basics of PDDL:
- `http://www.ida.liu.se/~TDDC17/info/labs/planning/writing.html`
- Konstruktionsanleitung
- Propositionale Logic -> Articifial Intelligence a Modern Approach
- To insert somewhere:
  - It should be mentioned, that almost no planner supports every part of PDDL. And, additionally, the quality of error messages is very diversified. While some simple state: error occured, other list the problem and the line.

Introduction to planing: `http://books.google.de/books?id=eCj3cKC_3ikC&printsec=frontcover&dq=automated+planning&hl=en&sa=X&ei=3wgNU5fQIcHx4gSTsoDABA&rec`

esc=y#v=onepage&q=automated%20planning&f=false

AI planning describes ...

A planner and use the generated solution file (*plan*).

PDDL was first described in PDDL-the planning domain definition language (1998) and has been in constant development since then. This thesis makes use of *PDDL 3.1* (n.d.) if not otherwise stated.

PDDL planning task specifications are composed of two separate text files:

- Domain file: description of general types, predicates, functions and actions -> uninstanciated problem independent
- Problem file: description of a concrete problem environment -> instance specic

This separation allows for an intuitive process of task modeling: While general instances are described in the domain file, specific instances of problems are created in the problem files.

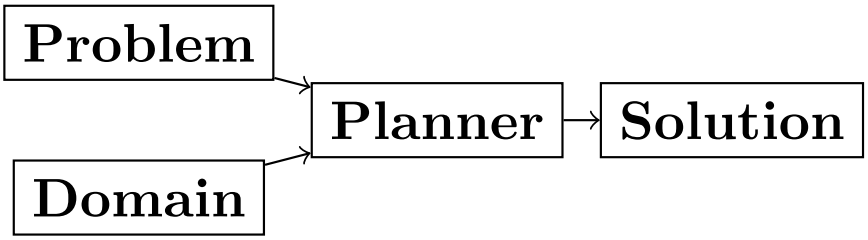

Figure 3.1: PDDL Planning workflow

These two files shell be investigated further in the following sections.

3.1 Format of the Domain File

Domain files have a strict format: All keyword arguments must appear in the order specified in the manual (an argument may be omitted) and just one PDDL definition (of a domain, problem, etc.) may appear per file. Fox and Long 2003, p. 6.

Include simple domain -> LaTeX Include simple problem -> LaTeX Include simple plan -> not yet in LaTeX

3.1.1 Define

Every domain file start with (define (domain <domainName>) ...) where, <domainName> can be any string

3.1.2 Requirements

The requirements part is not a mandatory part of a PDDL domain file. However, as most planners only support a subset of PDDL they are useful for determining if a planner is able to act on a given problem. They are declared by the (:requirements ...) part. Some often used requirements include ...

3.1.3 Types

If order to be able to use types in a domain file, the requirement :typing should be declared.

In order to assign to assign categories of objects, PDDL allows for type definitions. Like that, parameters in actions can be typed, as well as arguments in predicates, functions [extra source!]. Later, in the problem file, objects will be assigned to types, like objects to classes in Object Orientated Programming (OOP). Adding to the (:requirement ...) part of the file guarantees, that typing can be correctly used. Strips (no types) vs ADL (types).

3.1.4 Functions

Functions are not supported by many planners (source!) and, before PDDL 3.1 they could only be modeled as

It is notable that before PDDL 3.0 the keyword functors was used instead

3.1.5 Actions

PDDL 3.1 supports two types of actions: durative-action and the 'regular' action.

3.2 Format of the Problem File

3.3 Format of the Solution File (Plan)

3.4 Planning Process

Chapter 4

Software Engineering Tools for AI Planning

- PDDL type hierarchy and object instantiation to UML / TikZ, store predicates (and action?) in same box as type
- Research Knowledge Engineering in Planning
- Human Computer Interaction
  - `http://hci.waznelle.com/checklist.php`
- Write Tiago (itSimple) regarding PDDL -> UML (and knowledge engineering in general
- ICKEPS (International Competition on Knowledge Engineering for Planning and Scheduling)
- Orient on "How to Design Classes"

4.1 Statement of Problem

Writing and maintaining PDDL files can be time-consuming and cumbersome Li et al. (2012). So, the following development tools shell support and facilitate the PDDL task design process and reduce potential errors.

Below, methods are presented for

Syntax Highlighting and Code Snippets Environment for Editing PDDL files

Class Diagram Generator The automation of the PDDL task design process. File input and output and dynamic generation (design level)

Human Planner Interaction An interactive PDDL environment: speech synthesis and recognition.

Domain Generator Mathematical limitations (design level)

4.2 Syntax Highlighting and Code Snippets

Writing extensive domain and problem files is a cumbersome task: longer files can get quickly confusing. Therefore, it is convenient to have a tool that supports editing these files. Syntax highlighting describes the feature of text editors of displaying code in different colors and fonts according to the category of terms (source: Wiki). A syntax highlighting plug-in for the text and source code editors Skinner (2013a) and Skinner (2013b) is proposed and transferred to the on-line text editor Ace are used to implement this feature, as ST Syntax Highlighting files can easily be converted to Ace Files.

For Mac user, TextMate (TM) is very similar to ST and the syntax highlighting file can be used there, too. Besides, the general principles (e.g. regular expressions) outlined here, apply to most of other editors as well.

4.2.1 Implementation

ST syntax definitions are written in property lists in the XML format.

The syntax definition is implemented by the use of the ST plug-in *AAAPackageDev* (2014). So, the definitions can be written in YAML in converted to Plist XML later on. AAAPackageDEV provides the following features:

> AAAPackageDev is a Sublime Text 2 and 3 plug-in that helps to create and edit syntax definitions, snippets, completions files, build systems and other Sublime Text extensions.

By means of Oniguruma regular expressions (Kosako 2007), scopes are defined, that determine the meaning of the PDDL code block. The scope naming conventions mentioned in the *TextMate 2 Manual* are applied here. By the means of the name, the colors are assigned. Different ST themes display different colors (not all themes support all naming conventions).

The syntax highlighting is intended for PDDL 3.1, but is downward compatible, as previous versions are subsets of later versions.

Are later versions really subsets?
nil

Like that, the PDDL file is parsed

Is it really parsed, or are just parts highlighted?
nil

into different parts.

4.2.2 Usage and Customization

By using ST as editor, language independent ST features are supported, like auto completion, code folding and column selection, described in the Sublime Text 2 Documentation.

To enable syntax highlighting and code snippets, the files of the repository have to be placed in the ST packages folder. The first part of the PDDL.YAML-tmlanguage describes the parts of the PDDL task that should be highlighted. By removing (or commenting) include statements, the syntax highlighter is adjustable the user's need.

By default, all scopes are included.

1. Related Work
    (a) PDDL Studio PDDL Studio (Plch et al. 2012) is an Integrated Development Environment (IDE) for creating PDDL tasks.
    (b) PDDL Mode Announced 2005 in a mailing list entry, PDDL mode supports PDDL 2.2.
    (c) itSIMPLE
    (d) Pygments
    (e) ModPlan
        - Very interesting: `http://www.tzi.de/~edelkamp/modplan/`

4.2.3 Evaluation

4.3 Clojure Interface

As PDDL's syntax is inspired by LISP (Fox and Long 2003, p. 64), using a LISP dialect for the interface seems reasonable. This thesis uses Clojure (Hickey 2008), a relatively modern LISP dialect that runs on the Java Virtual Machine.

In this section, a kitchen domain will be presented, whereby PDDL structures are presented that will be also useful in other domains. I will start with a rather simple domain, present possible limitations and then extend the file by more sophisticated constructs.

4.3.1 Functions

As functions have a return value, the modeling possibilities dramatically increase.

4.3.2 Numerical Expressiveness

One might assume that the distance could be modeled as follows:

```
(durative action ...
...
  :duration (= ?duration (sqrt (coord-x )))
...
```

However, PDDL does only support basic arithmetic operations (+, -, /, *).

An Euclidean distance function that uses the square root would be convenient for distance modeling and measurement. However, PDDL 3.1 supports only four arithmetic operators (+, -, /, *). These operators can be used in preconditions, effects (normal/continuous/conditional) and durations. Parkinson and Longstaff (2012) describe a workaround for this drawback. By declaring an action 'calculate-sqrt', they bypass the lack of this function and rather write their own action that makes use of the Babylonian root method.

1. Alternative #1: Only sqrt exists Assuming that a function sqrt would actually exist, the duration could be modeled as follows:

```
:duration (= ?duration
             (sqrt
              (+
               (*
                (- (pos-x (current-pos))
                   (pos-x ?goal))
                (- (pos-x (current-pos))
                   (pos-x ?goal)))
               (*
                (- (pos-y (current-pos))
                   (pos-y ?goal))
                (- (pos-y (current-pos))
                   (pos-y ?goal))))))
```

2. Alternative #2: sqrt and expt exist Assuming that a function sqrt would actually exist, the duration could be modeled as follows:

```
:duration (= ?duration
             (sqrt
              (+
               (expt
                (-
                 (pos-x (current-pos))
                 (pos-x ?goal)))
               (expt
                (-
                 (pos-y (current-pos))
                 (pos-y ?goal))))))
```

3. Alternative #3: Calculate distance and hard code it, e.g. (distance table kitchen) = 5.9

   - Distance Matrix
   - `http://stackoverflow.com/questions/20654918/python-how-to-speed-up-calculatio`
   - Scipy.spatial.distance (-> Clojure?)
   - Mention that the Taxicab geometry allows different ways that have an equal length

   Another alternative is to make use of an external helper and, instead of calculating every entry of the distance matrix. the distance only if

needed, incorporate every possible combination of two locations. This approach has certainly a major drawback: With an increasing amount of locations, the number of combinations increases exponentially. That means, if there are 100 locations, there will be

TODO: Calculate possibilities

nil

. . . . The native approach would be to iterate over the cities twice and calculate only the half of the matrix (as it is symmetric, that mean distance from A to B is the same as the distance from B to A).

4. Alternative #4: Use the Manhattan distance

   Allowing the agent to move only vertically and horizontally would be that one can use the so called Taxicab geometry (or Manhattan length) as distance measurement. In the Kitchen domain, this could be modeled as follows:

```
% => Metric: reduce duration

% dKitchenware.pddl
\begin{figure}[t]
\inputminted[mathescape, linenos, numbersep=5pt, frame=lines, framesep=2mm]
            {csharp}
            {Code/dKitchenware.pddl}
\caption{The basic kitchenware domain}
\end{figure}
\section
```

TODO:

TODO Human Planner Interaction

4.3.3 PDDL Parsing

4.4 Type Diagram Generator

Add actions to the Type Diagram?

Types play a major role in the PDDL design process: they are involved, besides their definition, in the constants, predicates and actions part. So, a fine grasp of their hierarchy, as well as their involved predicates becomes handy and assists the KE in the planning process.

Chapter 5

Analysis

5.1 Participants

Ten non-paid students (six female) took part in the experiment. All had knowledge about LISP syntax, but neither one had faced PDDL prior to this study.

5.2 Material

The usability of the Syntax Highlighter (see 4.2) and the Type Diagram Generator (see `Type Diagram Generator`) were tested.

5.3 Design

The participants had to

5.4 Procedure

Chapter 6

Discussion and Outlook

Chapter 7

Bibliography

Bibliography

[1] *AAAPackageDev*. 2014. URL: `https://github.com/SublimeText/AAAPackageDev/` (visited on 02/24/2014).

[2] Maria Fox and Derek Long. "PDDL2. 1: An Extension to PDDL for Expressing Temporal Planning Domains." In: *J. Artif. Intell. Res.(JAIR)* 20 (2003), pp. 61–124.

[3] Rich Hickey. "The clojure programming language". In: *Proceedings of the 2008 symposium on Dynamic languages*. ACM. 2008.

[4] K. Kosako. *Oniguruma Regular Expressions Version 5.9.1*. 2007. URL: `http://www.geocities.jp/kosako3/oniguruma/doc/RE.txt` (visited on 02/24/2014).

[5] DL Kovacs. "BNF definition of PDDL 3.1". In: *Unpublished manuscript from the IPC-2011 website* (2011).

[6] Yi Li et al. "Translating pddl into csp#-the pat approach". In: *Engineering of Complex Computer Systems (ICECCS), 2012 17th International Conference on*. IEEE. 2012, pp. 240–249.

[7] MacroMates Ltd. *TextMate 2 Manual*. 2013. URL: `http://manual.macromates.com/en/` (visited on 02/24/2014).

[8] Drew McDermott et al. "PDDL-the planning domain definition language". In: (1998).

[9] Simon Parkinson and Andrew P Longstaff. "Increasing the Numeric Expressiveness of the Planning Domain Definition Language". In: *Proceedings of The 30th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG2012)*. UK Planning and Scheduling Special Interest Group. 2012.

[10] *PDDL 3.1*. URL: `http://ipc.informatik.uni-freiburg.de/PddlExtension`.

[11] Tomas Plch et al. “Inspect, edit and debug pddl documents: Simply and efficiently with pddl studio”. In: *and Exhibits* (2012), p. 15.

[12] Jon Skinner. *Sublime Text 2*. `http://www.sublimetext.com/2`. Release 2.0.2. 2013.

[13] Jon Skinner. *Sublime Text 3*. `http://www.sublimetext.com/3`. Beta Build 3059. 2013.

Chapter 8

Appendix

α