

NOTICE: This is the author's version of a work accepted for publication by Springer. The final publication is available at Springer via

[http://dx.doi.org/10.1007/978-3-319-11206-0\\_27](http://dx.doi.org/10.1007/978-3-319-11206-0_27)

# Planning in the Wild: Modeling Tools for PDDL

Volker Strobel, Alexandra Kirsch

Department of Informatics  
Eberhard Karls Universität Tübingen

**Abstract.** Even though there are sophisticated AI planning algorithms, many integrated, large-scale projects do not use planning. One reason seems to be the missing support by engineering tools such as syntax highlighting and visualization. We propose MYPDDL—a modular toolbox for efficiently creating PDDL domains and problems. To evaluate MYPDDL, we compare it to existing knowledge engineering tools for PDDL and experimentally assess its usefulness for novice PDDL users.

## 1 Introduction

A large community of researchers dedicate their efforts to AI planning. However, the progress made in this community is often ignored when it comes to developing complete AI systems. Planning is a fundamental cognitive function that is useful for most systems claiming to be intelligent, such as autonomous robots or decision support systems.

The basics of AI planning are taught in any AI course and there are many planners readily available, also due to the International Planning Competition<sup>1</sup>. So why is planning not used in more systems? We believe that one reason is the gap between modeling textbook toy problems and modeling complex, real-world problems. The standard AI planning language PDDL differentiates between domain files with definitions of types, predicates and actions, and problem files with definitions of objects and goals. Realistic domains contain hundreds of objects, different agents with different capabilities, able to perform a large variety of actions. Modeling such domains soon gets confusing: object and action definitions depend on the type hierarchy, goals have to be compatible with predicate and action definitions, etc.

This problem is not specific to planning, but poses a challenge to software engineering in general. As projects grow in size, developers have to be supported with appropriate tools in order to keep track of the overall structure. We think that it is time to move AI planning from a purely scientific discipline into the direction of *Planning Task Engineering* in the sense that planning becomes a standard component of AI systems, readily usable for anyone wishing to build intelligent systems.

In this paper we propose MYPDDL, a set of tools for modeling large domains and associated problems. After a review of existing tools for PDDL, we introduce

---

<sup>1</sup> <http://ipc.icaps-conference.org/>

the MYPDDL modules. We then compare MYPDDL to the other existing tools and present a user test for two MYPDDL modules. We conclude with an outlook on further steps necessary to improve the availability of planning for intelligent system development.

## 2 Existing Tools

There have been some attempts to provide modeling tools for PDDL. We introduce the three most sophisticated tools we found and use those as a benchmark for MYPDDL in Section 4.1.

PDDL STUDIO<sup>2</sup> [13] is an application for creating and managing PDDL projects, i.e. a collection of PDDL files. The PDDL STUDIO integrated development environment (IDE) was inspired by Microsoft Visual Studio and imperative programming paradigms. Its main features are syntax highlighting, error detection, context sensitive code completion, code folding, project management, and planner integration. PDDL STUDIO's error detection can recognize both syntactic (missing keywords, parentheses, etc.) and semantic (wrong type of predicate parameters, misspelled predicates, etc.) errors.

A major drawback of PDDL STUDIO is that it is not updated regularly and only supports PDDL 1.2. Later PDDL versions contain several additional features such as durative actions, numeric fluents, and plan-metrics [6].

ITSIMPLE [20] follows a graphical approach using Unified Modeling Language (UML) [3] diagrams. In the process leading up to ITSIMPLE, UML.P (UML in a Planning Approach) was proposed, a UML variant specifically designed for modeling planning domains and problems [19].

ITSIMPLE's modeling workflow is unidirectional as changes in the PDDL domain do not affect the UML model and UML models have to be modeled manually, meaning that they cannot be generated from PDDL. However, [18] present a translation process from a PDDL domain specification to an object-oriented UML.P model as a possible integration for ITSIMPLE. This translation process makes extensive semantic assumptions for PDDL descriptions. For example, the first parameter in the `:parameters` section of an action is automatically declared as a subclass of the default class *Agent*, and the method is limited to predicates with a maximum arity of two. The currently version of ITSIMPLE does not include the translation process from PDDL to UML.

Starting in version 4.0 [21] ITSIMPLE expanded its features to allow the creation of PDDL projects from scratch (i.e. without the UML to PDDL translation process). Thus far, the PDDL editing features are basic. A minimal syntax highlighting feature recognizes PDDL keywords, variables, and comments. ITSIMPLE also provides templates for PDDL constructs, such as requirement specifications, predicates, actions, initial state, and goal definitions.

Both PDDL STUDIO and ITSIMPLE do not build on existing editors and therefore cannot fall back on refined implementations of features that have been modified and improved many times throughout their existence.

---

<sup>2</sup> <http://amis.mff.cuni.cz/PDDLStudio/>

The PDDL-mode for the widely used Emacs editor [17] builds on the sophisticated features of Emacs and uses its extensibility and customizability. It provides syntax highlighting by way of basic pattern matching of keywords, variables, and comments. Additional features are automatic indentation and code completion as well as bracket matching. Code snippets for the creation of domains, problems, and actions are also available. Finally, the PDDL-mode keeps track of action and problem declarations by adding them to a menu and thus intending to allow for easy and fast code navigation.

PDDL-mode for Emacs supports PDDL versions up to 2.2, which includes derived predicates and timed initial predicates [5], but does not recognize later features like object-fluents.

In sum, there is currently no tool available supporting all features of PDDL 3.1, nor all the steps in the modeling process.

### 3 MyPDDL

MyPDDL is designed as a modular framework. We first introduce the implemented modules and then explain their details with respect to design guidelines for knowledge engineering tools.

#### 3.1 Modules

MyPDDL-IDE is an integrated development environment for the use of MyPDDL in Sublime Text<sup>3</sup>. Since MyPDDL-SNIPPET and -SYNTAX are devised explicitly for Sublime Text, their integration is implicit. The other tools can be used independently of Sublime Text with the command-line interface and any PDDL file, but were also integrated into the editor.

MyPDDL-SYNTAX is a context-aware syntax highlighting feature for Sublime Text. It distinguishes all PDDL constructs up to version 3.1. Using regular expressions that can recognize both the start and the end of code blocks by means of a sophisticated pattern matching heuristic, MyPDDL-SYNTAX identifies PDDL code blocks and constructs and divides them into so called *scopes*, i.e. named regions. Sublime Text colorizes the code elements via the assigned scope names and in accordance with the current color scheme. These scopes allow for a fragmentation of the PDDL files, so that constructs are only highlighted if they appear in the correct context. Thus missing brackets, misplaced expressions and misspelled keywords are visually distinct and can be identified (see Figure 1).

MyPDDL-NEW helps to organize PDDL projects by generating the following folder structure:

---

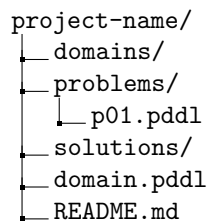
<sup>3</sup> <http://www.sublimetext.com/>

```

coffee_errors.pddl
1 (define COFFEE
2
3 (requirements
4  :typing)
5
6 (:types room - location
7      robot human _ agent
8      furniture door - (at ?l - location)
9      kettle ?coffee cup water - movable
10     location agent movable - object)
11
12 (:predicates (at ?l - location ??o - object)
13             (have ?m - movable ?a - agent)
14             (hot ?m - movable) = true
15             (on ?f - furniture ?m - movable))
16
17 (:action boil
18     :parameters (?m - movable $k - kettle ?a - agent)
19     :preconditions (have ?m ?a)
20     :effect (hot ?m))
21
Line 20, Column 22 Spaces: 2 PDDL

```

**Fig. 1.** Syntax highlighting in the MYPDDL-IDE in the Sublime Text editor. White text contains errors.

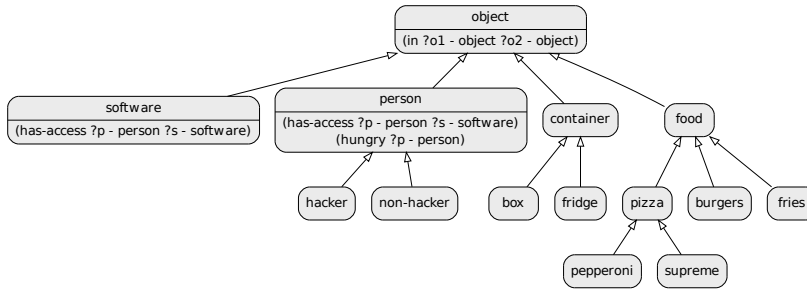


The domain file `domain.pddl` and the problem file `p01.pddl` initially contain corresponding PDDL skeletons which can also be customized. All problem files that are associated with one domain file are collected in the folder `problems/`. `README.md` is a Markdown file, which is intended for (but not limited to) information about the author(s) of the project, contact information, informal domain and problem specifications, and licensing information. Markdown files can be converted to HTML by various hosting services (like GitHub or Bitbucket).

MYPDDL-SNIPPET provides code skeletons, i.e. templates for often used PDDL constructs such as domains, problems, type and function declarations, and actions. They can be inserted by typing a triggering keyword.

MYPDDL-CLOJURE provides a preprocessor for PDDL files to bypass PDDL's limited mathematical capabilities, thus reducing modeling time without overcharging planning algorithms. We decided to use Clojure [7], a modern Lisp dialect that runs on the Java Virtual Machine (JVM) [9], facilitating input and output of the Lisp-style PDDL constructs. MYPDDL-CLOJURE is the basis for MYPDDL-DISTANCE and MYPDDL-DIAGRAM.

MYPDDL-DISTANCE provides special preprocessing functions for distance calculations. For domains with spatial components, the distance of objects is often important and should not be omitted in the domain model. However, calculating distances from coordinates requires the square root function, which is not supported by PDDL (it only supports the four basic arithmetic oper-



**Fig. 2.** Type diagram generated by MYPDDL-DIAGRAM.

ators). More sophisticated calculations can be achieved with the supported operators, but the solutions are rather inefficient and inelegant [12]. By calculating the distances offline and including them as additional predicates in the problem file using MYPDDL-DISTANCE, the distances between objects are given to the planner as part of the problem description.

MYPDDL-DIAGRAM generates a png image from a PDDL domain file as shown in Figure 2. The diagrammatic representation of textual information helps to quickly understand the connection of hierarchically structured items and should thus be able to simplify the communication and collaboration between developers. In the process of generating the diagrams, a copy of the PDDL file is created, so that a simple version control is also included.

### 3.2 Design Principles

As guidelines for design decisions, we used the seven criteria for knowledge engineering tools proposed by Shah et al. [16] as well as general usability principles.

*Operationality* asks whether the generated models can improve the planning performance. This is not a design principle for MYPDDL, because we assume that MYPDDL does not improve the quality (with respect to planning performance) of the resulting PDDL specifications. Therefore, we replaced this criterion with *functional suitability* from the ISO/IEC 25010 standard, which is defined as “the degree to which the software product provides an appropriate set of functions for specified tasks and user objectives” (ISO 25010 6.1.1). MYPDDL supports the current version 3.1 of PDDL. It encompasses and exceeds most of the functionality of the existing tools. It specifically supports basic editor features with a high customizability as well as visualization support.

*Collaboration:* With the growing importance of team work and team members not necessarily working in the same building, or in the same country, there is an increasing need for tools supporting the collaboration effort. In developing MYPDDL, this need was sought to be met by MYPDDL-DIAGRAM. Complex type hierarchies can be hard to overlook, especially if they were constructed by someone else. Therefore, a good way of tackling this problem seemed to be by providing a means to visualize such hierarchies in the form of type diagrams.

*Experience:* MYPDDL was designed specifically for users with a background in AI, but not necessarily in PDDL. The tools are similar to standard software engineering tools and should thus be easily learnable. The user evaluation (Section 4.2) confirms that MYPDDL helps novices in PDDL to master planning task modeling. In addition, it is also possible to customize MYPDDL so as to adapt its look and feel to other programs one is already familiar with, or simply to make it more enjoyable to use. The project site<sup>4</sup> provides MYPDDL video introductions and a manual to get started quickly.

*Efficiency:* All MYPDDL tools are intended to increase the efficiency with which PDDL files are created. MYPDDL-SNIPPET enables the fast creation of large and correct code skeletons that only need to be complemented. MYPDDL-SYNTAX can reduce the time spent on searching errors. Code folding allows users to hide currently irrelevant parts of the code and automatic indentation increases its readability. To easily keep track of all the parts of a project, folders are automatically created and named with MYPDDL-NEW. MYPDDL-CLOJURE and -DISTANCE allow for a straightforward inclusion of numerical values in the problem definition.

*Debugging:* MYPDDL-SYNTAX highlights all syntactically correct constructs and leaves all syntactical errors non-highlighted. In contrast, PDDL-mode for Emacs and ITSIMPLE only provide basic syntax highlighting for emphasizing the structure. PDDL STUDIO explicitly detects errors, but the user is immediately prompted when an error is detected. Often, such error messages are premature, for example, just because the closing parenthesis was not typed yet, does not mean it was forgotten. MYPDDL indicates errors in a more subtle way: syntactic errors are simply not highlighted, while all correct PDDL code is. The colors are customizable, so that users can choose how prominently the highlighting sticks out.

*Maintenance:* The possibility to maintain PDDL files is a key aspect of MYPDDL. The automatically generated type diagram (MYPDDL-DIAGRAM) gives an overview of the domain structure and thereby serves as a continuous means of documentation. Helping to understand foreign code, though, it follows logically that MYPDDL-DIAGRAM also helps in coming back and changing one's own models if some time has elapsed since they were last edited. The basic revision control feature of MYPDDL-DIAGRAM keeps track of changes, making it easy to revert to a previous domain version. Furthermore, MYPDDL-NEW encourages adhering to an organized project structure and stores corresponding files at the same location. The automatically created readme file can induce the user to provide further information and documentation about the PDDL project.

*Support:* MYPDDL-IDE can be installed using Sublime Text's Package Control [2]. This allows for an easy installation and staying up-to-date with future versions. In order to provide global access and with it the possibility for developing an active community, the project source code is hosted on GitHub<sup>5</sup>. Additionally, the project site provides room for discussing features and reporting bugs.

---

<sup>4</sup> <http://pold87.github.io/myPDDL/>

<sup>5</sup> <https://github.com/Pold87/myPDDL>

## 4 Validation and Evaluation

To assess the utility of MYPDDL, we used the criteria listed in Section 3.2. We show the functional suitability in a benchmark validation, comparing MYPDDL’s functionality with the tools described in Section 2. The criteria collaboration, experience, efficiency and debugging were evaluated in a user test. The MYPDDL components supporting maintenance are the same that are used in the user test, but their long-term usage is difficult to evaluate. The support criterion depends primarily on the infrastructure, which has been established as explained in Section 3.2.

### 4.1 Benchmark Validation

Functional suitability encompasses the set of functions to meet the user objectives. The tools of Section 2 basically all follow the same objectives as MYPDDL: creating PDDL domains and problems. They intend to support this process in general and the various stages of the design cycle to different degrees. The features offered by each tool are summarized in Table 1.

Besides supporting the latest PDDL version, a strength of MYPDDL is its high customizability, which comes with the Sublime Text editor. Being the only one of the four tools capable of visualizing parts of the PDDL code, it must be understood as complementary to ITSIMPLE, which takes the opposite approach of transforming UML diagrams into PDDL files. The fact that MYPDDL does not check for semantic errors is not actually a drawback as planners will usually detect semantic errors. All in all, MYPDDL combines the most useful tools of PDDL STUDIO, ITSIMPLE, and PDDL-mode for Emacs and strives to support the planning task engineer during all phases of the modeling process. Additionally, it features some unique tools, such as domain visualization. It can therefore be concluded that MYPDDL provides an appropriate set of functions for developing PDDL files and is thus functionally suitable.

### 4.2 User Evaluation

The two most central modules of MYPDDL are MYPDDL-SYNTAX and MYPDDL-DIAGRAM, since they support collaboration, efficiency and debugging independently of the user’s experience with PDDL.

*Procedure.* We invited eight participants<sup>6</sup> to a user test (three female, average age 22.9, standard deviation 0.6), who had some basic experience with at least one Lisp dialect (in order not to be confused with the many parentheses), but no experience with PDDL or AI planning in general.

---

<sup>6</sup> In Usability Engineering, a typical number of participants for user tests is five to ten. Studies have shown that even such small sample sizes identify about 80% of the usability problems [10, 8]. Our study design required at least eight participants.



**Table 1.** Comparison of knowledge engineering tools and their features.

| Feature                             | PDDL<br>STUDIO | ITSIMPLE | PDDL-<br>mode | MYPDDL |
|-------------------------------------|----------------|----------|---------------|--------|
| latest supported PDDL version       | 1.2            | 3.1      | 2.2           | 3.1    |
| syntax highlighting                 | yes            | basic    | basic         | yes    |
| semantic error detection            | yes            | no       | no            | no     |
| automatic indentation               | no             | no       | yes           | yes    |
| code completion                     | yes            | no       | yes           | yes    |
| code snippets                       | no             | yes      | yes           | yes    |
| code folding                        | yes            | no       | yes           | yes    |
| domain visualization                | no             | planned  | no            | yes    |
| project management                  | yes            | yes      | no            | yes    |
| UML to PDDL code translation        | no             | yes      | no            | no     |
| planner integration                 | basic          | yes      | no            | basic  |
| plan visualization                  | no             | yes      | no            | no     |
| dynamic analysis                    | no             | yes      | no            | no     |
| declaration menu                    | no             | no       | yes           | no     |
| interface with programming language | no             | no       | no            | yes    |
| customization features              | basic          | no       | yes           | yes    |

No earlier than 24 hours before the experiment was to take place, participants received the web link to a 30-minute interactive video tutorial on AI planning and PDDL<sup>7</sup>. This method was chosen in order not to pressure the participant with the presence of an experimenter when trying to understand the material.

We defined four tasks: two debugging tasks and two type hierarchy tasks asking for details of a given domain (e.g. “Can a Spleus be married to a Schlok?”). As a within-subjects design was considered most suited (to control for individual differences within such a small sample), it was necessary to construct two tasks (matched in difficulty) for each of these two types to compare the effects of having the tools available. The two tasks to test syntax highlighting presented the user with domains that were 54 lines in length, consisted of 1605 characters and contained 17 errors each. Errors were distributed evenly throughout the domains and were categorized into different types. The occurrence frequencies of these types were matched across domains as well, to ensure equal difficulty for both domains. To test the type diagram generator, two fictional domains with equally complex type hierarchies consisting of non-words were designed (five and six layers in depth, 20 and 21 types). The domains were also matched in length and overall complexity (five and six predicates with approximately the same distribution of arities, one action with four predicates in the precondition and two and three predicates in the effect).

Each participant started either with a debugging or type hierarchy task and was given the MYPDDL tools either in the first two tasks or the second two tasks, so that each participant completed each task type once with and once

<sup>7</sup> <http://www.youtube.com/playlist?list=PL3CZzLUZuiMWEfJxy-G6OxYVzUrvjwuV>

without MYPDDL. This results in 2 (first task is debugging or hierarchy)  $\times$  2 (task variations for debugging and hierarchy)  $\times$  2 (starting with or without MYPDDL) = 8 individual task orders, one per participant.

For the debugging tasks, participants were given six minutes<sup>8</sup> to detect as many of the errors as possible. They were asked to record each error in a table (pen and paper) with the line number and a short comment and to immediately correct the errors in the code if they knew how to, but not to dwell on the correction otherwise. For the type hierarchy task, participants were asked to answer five questions concerning the domains, all of which could be facilitated with the type diagram generator, but one of which also required looking into the code. Participants were told that they should not feel pressured to answer quickly, but to not waste time either. Also they were asked to say their answer out loud as soon as it became evident to them. They were not told that the time it took them to come up with an answer was recorded, since this could have made them feel pressured and thus led to more false answers. At the end of the usability test they were asked to evaluate the perceived usability of MYPDDL using the system usability scale [4].

### *Results.*

#### 1. Debugging Tasks

As shown in Figure 3(a), on average participants found 7.6 errors without syntax highlighting and 10.3 errors with syntax highlighting (i.e. approximately 36% more errors were found with syntax highlighting).

Two participants remarked that the syntax highlighting colors confused them and that they found them more distracting than helpful. One of them mentioned that the contrast of the colors used was so low that they were hard for her to distinguish. She found the same number of errors with and without syntax highlighting. The other of the two was the only participant who found fewer errors with syntax highlighting than without it. With MYPDDL-SYNTAX, two participants found all errors in the domain, while none achieved this without syntax highlighting.

#### 2. Type Hierarchy Tasks

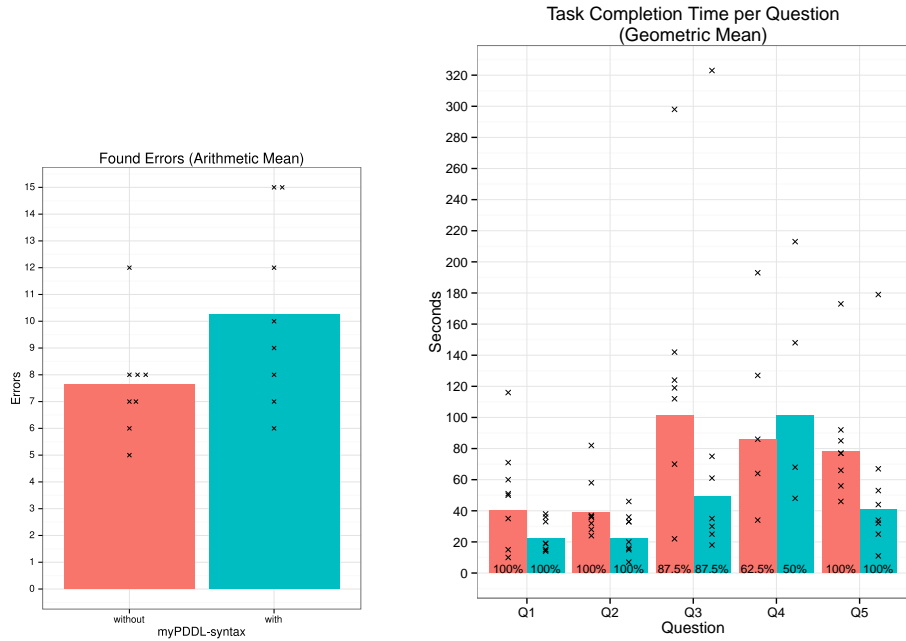
Figure 3(b) shows the geometric mean<sup>9</sup> of the completion time of successful tasks for each question with and without the type diagram generator.

With the type diagram generator participants answered all questions (except Question 4) on average nearly twice as fast. The fact that the availability of tools did not have a positive effect on task completion times for Question 4 can probably be attributed to the complexity of this question. In contrast to the other four questions, to answer Question 4 correctly, the participants were required to look at the actions in the domain file in addition to the type diagram. Most participants were confused by this, because they had assumed that once having the type diagram available, it alone would suffice

---

<sup>8</sup> A reasonable time frame tested on two pilot tests.

<sup>9</sup> The geometric mean is a more accurate measure of the mean for small sample sizes as task times have a strong tendency to be positively skewed [15].



(a) Comparison of detected errors with and without the syntax highlighting feature. The bars display the arithmetic mean.

(b) Task completion time for the type hierarchy tasks. The bars display the geometric mean. The percent values at the bottom of the bars show the percentage of users that completed the task successfully.

**Fig. 3.** Results of the user test. The crosses (×) represent single participants. red: with MYPDDL, blue: without MYPDDL

to answer all questions. This initial confusion cost some time, thus negatively influencing the time on the task.

### 3. System Usability Scale

MYPDDL reached a score of 89.6 on the system usability scale <sup>10</sup>, with a standard deviation of 3.9. Since the overall mean score of the system usability scale has an approximate value of 68 with a standard deviation of 12.5 [14], the score of MYPDDL is well above average with a small standard deviation. A score of 89.6 is usually attributed to superior products [1]. Furthermore, 89.6 corresponds approximately to a percentile rank of 99.8%, meaning that it has a better perceived ease-of-use than 99.8% of the products in the database used by Sauro [14].

*Discussion.* The user test shows that overall MYPDDL provides very useful tools for novices in AI planning and PDDL. It also shows that customizability is im-

<sup>10</sup> The range of possible values for the system usability scale is 0 to 100.

portant, as not all users prefer the same colors or syntax highlighting at all and their personal preferences seem to correlate with the effectiveness of the tools.

Visualization tools such as MYPDDL-DIAGRAM can improve the understanding of unknown PDDL code and thus support collaboration. But users may be unaware of the limitations of such tools. A possible solution is to extend MYPDDL-DIAGRAM to display actions, but this can overload the diagram and especially for large domains render it unreadable. Different views for different aspects of the domain or dynamically displayed content could integrate more data, but this also hides functionality, which is generally undesired for usability [11].

## 5 Conclusion

MYPDDL was designed with the goal to support plan engineers in modeling domains and planning problems as well as in understanding, modifying, extending, and using existing planning domains. This was realized with a set of tools comprising code editing features, namely syntax highlighting and code snippets, a type diagram generator, and a distance calculator. To also have all tools accessible from one place, they were made available in the Sublime Text editor. The different needs and requirements of knowledge engineers are met by the modular, extensible, and customizable architecture of the toolkit and Sublime Text. The evaluation of MYPDDL has shown some initial evidence that it allows a faster understanding of the domain structure, which could be beneficial for the maintenance and application of existing task specifications and for the communication between engineers. Users perceive it as easy and enjoyable to use, and the increase in their performance when using MYPDDL underpins their subjective impressions.

Despite MYPDDL already providing a rich modeling environment, there are still numerous features that could be added in the future. Especially MYPDDL-CLOJURE offers multiple interesting further research directions: It provides a basis for dynamic planning scenarios. Applications could be the modeling of learning and forgetting (by adding facts to or retracting facts from a PDDL file) or the modeling of an ever changing real world via dynamic predicate lists. Another way of putting the interface to use would be by making the planning process more interactive, allowing for the online interception of planning software in order to account for the needs and wishes of the end user.

We hope that MYPDDL will be accepted in the planning community, but especially that it will be used by people who create integrated AI systems and are not necessarily planning experts. We are looking forward to feedback for continual improvements.

Project website: <http://pold87.github.io/myPDDL/>

## References

1. Bangor, A., Kortum, P.T., Miller, J.T.: An empirical evaluation of the system usability scale. *Intl. Journal of Human - Computer Interaction* 24(6), 574–594 (2008)

2. Bond, W.: (2013), <https://sublime.wbond.net/about>
3. Booch, G., Rumbaugh, J., Jacobson, I.: *The Unified Modeling Language User Guide*. Pearson Education India (1999)
4. Brooke, J.: SUS — a quick and dirty usability scale. *Usability evaluation in industry* 189, 194 (1996)
5. Edelkamp, S., Hoffmann, J.: PDDL2.2: The language for the classical part of the 4th international planning competition. *4th International Planning Competition (IPC'04)*, at ICAPS'04 (2004)
6. Fox, M., Long, D.: PDDL2.1: An extension to PDDL for expressing temporal planning domains. *J. Artif. Intell. Res.(JAIR)* 20, 61–124 (2003)
7. Hickey, R.: The Clojure programming language. In: *Proceedings of the 2008 Symposium on Dynamic languages*. ACM (2008)
8. Hwang, W., Salvendy, G.: Number of people required for usability evaluation: The 10±2 rule. *Communications of the ACM* 53(5), 130–133 (2010)
9. Lindholm, T., Yellin, F., Bracha, G.: *Virtual machine specification*. Java Card™ Platform, Version 2(2) (2011)
10. Nielsen, J.: Estimating the number of subjects needed for a thinking aloud test. *International Journal of Human-Computer Studies* 41(3), 385–397 (1994)
11. Norman, D.: *The Design of Everyday Things*. Basic Books, New York (1988)
12. Parkinson, S., Longstaff, A.P.: 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)
13. Plch, T., Chomut, M., Brom, C., Barták, R.: Inspect, edit and debug PDDL documents: Simply and efficiently with PDDL studio. In: *System Demonstrations and Exhibits at ICAPS 2012* (2012)
14. Sauro, J.: *A Practical Guide to the System Usability Scale: Background, Benchmarks & Best Practices*. Measuring Usability LCC (2011)
15. Sauro, J., Lewis, J.R.: *Quantifying the User Experience: Practical Statistics for User Research*. Elsevier (2012)
16. Shah, M., Chrupa, L., Jimoh, F., Kitchin, D., McCluskey, T., Parkinson, S., Vallati, M.: Knowledge engineering tools in planning: State-of-the-art and future challenges. *Knowledge Engineering for Planning and Scheduling* p. 53 (2013)
17. Singhi, S.: Emacs mode for PDDL (2005), <http://rakaposhi.eas.asu.edu/planning-list-mailarchive/msg00085.html>
18. Tonidandel, F., Vaquero, T.S., Silva, J.R.: Reading PDDL, writing an object-oriented model. In: *Advances in Artificial Intelligence-IBERAMIA-SBIA 2006*, pp. 532–541. Springer (2006)
19. Vaquero, T.S., Tonidandel, F., de Barros, L.N., Silva, J.R.: On the use of UML.p for modeling a real application as a planning problem. In: *International Conference on Automated Planning and Scheduling (ICAPS)*. pp. 434–437 (2006)
20. Vaquero, T.S., Tonidandel, F., Silva, J.R.: The itSIMPLE tool for modeling planning domains. *Proceedings of the First International Competition on Knowledge Engineering for AI Planning, Monterey, California, USA* (2005)
21. Vaquero, T., Tonaco, R., Costa, G., Tonidandel, F., Silva, J.R., Beck, J.C.: it-simple4.0: Enhancing the modeling experience of planning problems. In: *System Demonstration-Proceedings of the 22nd International Conference on Automated Planning & Scheduling (ICAPS-12)* (2012)