Inspect, Edit and Debug PDDL Documents: Simply and Efficiently with PDDL Studio

Tomas Plch[1], Miroslav Chomut[2], Cyril Brom[3], Roman Barták[4]
Faculty of Mathematics and Physics, Charles University in Prague
[1]tomas.plch@gmail.com [2]chmirko@gmail. com [3]brom@ksvi.mff.cuni.cz [4]bartak@ktiml.mff.cuni.cz

Abstract

The Planning Domain Definition Language (PDDL) represents a standard for definitions of planning domains and problems. Researchers and designers often make semantic and syntax errors due to the language's complexity. At the same time, it is hard to read and work with larger documents in PDDL. We have developed a tool called *PDDL Studio*, which is aimed at aiding in creation and inspection of PDDL documents. The editor's main features are: 1) PDDL parser capable of localizing syntax and semantic errors, 2) PDDL syntax highlighting, 3) context sensitive code completion and hints - similar to Microsoft's *IntelliSense* for declarative languages, 4) code collapsing, 5) PDDL document management, and 6) planner integration. Our PDDL Editor also features a PDDL Parser tool, which can be used as a standalone parser for other projects.

Introduction

The Planning Domain Definition Language (PDDL) (McDermott et al. 1998) represents a standard for creating definitions of planning domains and problems and is utilized as input for various planners e.g. JSHOP, JSHOP2 (Nau et al. 1999), BlackBox (Kautz and Selman 1998), Metric-FF (Hoffman 2003).

The current practice is to create PDDL documents either by hand via simple editing tools e.g. Notepad++, or via tools and languages for domain knowledge and characteristics specification (i.e. knowledge engineering). To name a few of such tools: itSimple (Vaquero et al. 2009) utilizes the combination of graphical UML specification and XML, GIPO IV (Simpson 2007) uses custom diagram notation, and ViTAPlan (Vrakas and Vlahavas 2003) for domain and problem visualization. An extensive study of the various tools and approaches can be found in (Vaquero et al. 2011). It is noteworthy that there is a multitude of tools aimed at verification of PDDL like the VAL tool (Howey et al. 2004) or PDver (Raimondi et al. 2009).

At some point during the process of knowledge engineering for planning and scheduling, the need for directly inspecting and editing of PDDL documents often occurs. Regardless of whether these documents are created automatically or manually, they often are large and complex, thus being hard to inspect.

The imperative programming language community (i.e. utilizing languages like C++ and Java) has a wide range of Integrated Development Environments (IDEs) e.g. Visual Studio 2010 (VS10), NetBeans, and Eclipse. Various functionalities (e.g. syntax highlighting, on the fly syntax checking, code completion and contextual hints), help programmers to develop in a faster and more convenient manner. The planning community lacks such an integrated tool (i.e. editor) for PDDL. With this motivation in mind, we created our *PDDL Studio* application, which is aimed at bringing the imperative programming culture of editing source code to the planning community. In this paper we overview main features of PDDL Studio.

PDDL Studio

Our project, PDDL Studio (Figure 1), is focused on providing a simple editing IDE. The application itself is written in C++ and is designed with portability in mind, utilizing portable technologies like the Flex Lexical Parser (Paxson 2008), Bison parser generator (GNU 2011) and the Qt framework (Qt Project 1992) for visual representation.

We identified a broad range of necessary capabilities, which are present in most of applications like VS10, NetBeans, Eclipse and can be applied to the PDDL language:

- *Project management* – creation and management of documents
- *Syntax error detection* – interactive localization and identification of PDDL syntax errors
- *Syntax highlighting* – coloration of language elements

- *Semantic error detection* – detection of simple semantic errors
- *Context sensitive code completion* – providing hints to the user what to write based on the current context in the document's input
- *Code collapsing* – portions of the code can be *collapsed* to provide better readability
- *XML import/export* – creating an XML variant of the PDDL document
- *Planner integration* – the ability to execute a planner with the PDDL documents as input
- *Common Editor Features* – line numbering and bracket matching

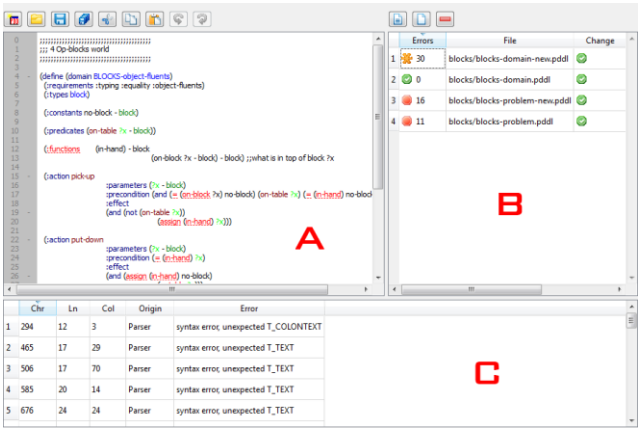

Figure 1: PDDL Editor window with Editing Window (A) having highlighted syntax, Interactive Error Report (C) and Project Manager (B) with additional information about file status (number of errors, save status etc.)

The remainder of this section is focused on describing the realization of the outlined capabilities in PDDL Studio. The main aim of the realization is to provide intuitive and easy to use features that would facilitate the task of creating PDDL encodings.

Project management

Presently, the *Project Management* in PDDL Studio is rather simplistic, providing only the capability to create projects, add and remove files from the project. The Project Manager also provides additional information on project files (e.g. present error counts, change status etc.).

Syntax error detection

Syntax error detection is one of the most important features of the PDDL Studio. We created our own dedicated PDDL parser build upon Flex and Bison (presently supporting PDDL 1.7), making the project more portable. The parser is designed to be used independently from the PDDL Studio's code as a library or C++ code. The parsing process creates a *tree-like representation* of the PDDL document's elements. Our testing domains of 1000 lines can be parsed below 100 ms on a standard Dual Core notebook processor at 1.8GHz utilizing one core.

The error detection mechanism is executed on the fly during editing of the PDDL document. If the user only views the file, the detection mechanism is suspended. If the user inserts or removes text, the detection mechanism is prepared for execution after a predefined period of inactivity – i.e. the user stopped typing. The parsing mechanism can be further suspended if the user resumes typing. This allows providing smoother error detection, avoid over-consuming resources, and avoid bothering the user with false positives on syntax errors.

The error detection module provides a list of errors presented as an interactive error table and underlines the found errors in the document. It can be seen in Figure 1 (C). When accessed (e.g. double click on an error's row), the editor points directly to the detected error. In respect to the PDDL specification, we can localize misspelled or missing keywords. When a mandatory keyword is missing, we also identify which keyword is missing, thus providing a hint to the user.

Syntax Highlighting

Based on our experience with various programming IDEs, we perceive the syntax highlighting as one of the most important aids when inspecting any code. It is a key feature included in every usable IDE tool since colored fonts were available. It vastly improves the ability for the developer to read code and distinguish language elements – i.e. variables, types, functions, predicates.

We use the tree-like structure provided by our parser to identify language elements and assign colors to the resulting editable text. The result can be seen in Figure 1 (A). The user can set his preferred colors for the edited text and the following language constructs:

- problem and domain names
- variable names
- detected errors – these are underlined for better display and this color overrides the color of any other element
- PDDL keywords – e.g. *requirements, predicates* etc.
- predicate names
- type names
- highlighted brackets – pairs of brackets are highlighted when the user points the editing cursor at one of them

Semantic error detection

The PDDL Studio is also capable of detecting semantic errors in respect to the language specification and the provided requirements (e.g. *Disjunctive-Preconditions*). The parsed tree-like structure is used to determine where these errors are located in the PDDL documents and what

their nature is. This information is provided within the error table (Figure 1(C)). We can detect the following semantic errors:

- Use of non-existent types – when the user misspells a type which is not present in the *type* declaration. This is checked based on the *typed* requirement.
- Use of non-existent predicates – when a predicate is misspelled or not defined correctly.
- Inconsistent use of predicates parameters – when a predicate is used with wrong parameter types.
- Inconsistent use in respect to domain requirements.

Context sensitive code completion

Code completion is an important feature currently included in every major IDE. The basic idea is to provide the user with hints based on the current scope (e.g. available functions, keywords etc.) while editing the document. This feature takes the burden of the user to avoid errors i.e. typing errors and syntactic and semantic errors. It also provides a speed-up for development.

Our code completion is context sensitive – based on the current edited portion of the PDDL document as can be seen in Figure 2.

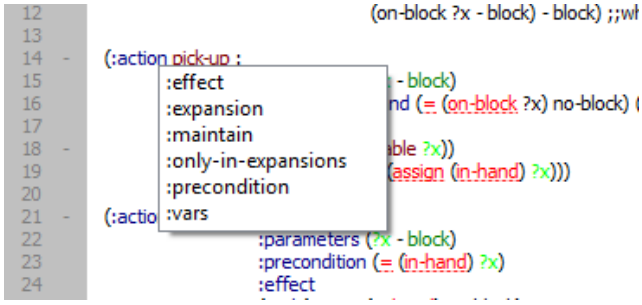

Figure 2: Context sensitive auto completion – a hint is given for a subsection of the Action element

We can provide completion hints in the following areas:

- language keywords – basic PDDL language elements
- domain specification for problems on known domains in the project
- predicates for problem initialization on known predicates in the project
- content for the requirements specification
- defined variables and parameters
- defined predicates

Code collapsing

The code collapsing feature is important for better code readability. Parts of the code – code blocks – can be hidden, because the reader does not need to read them at the moment.

Our project provides possibility to collapse automatically detected portions of code (e.g. actions, predicates etc.). The PDDL language is based on Lisp notation, therefore is full of code blocks. We presently support only high level code collapsing (Figure 3) – i.e. at the level of whole actions, predicates etc. We are working on a method to specify how deep the code collapsing should be allowed to maintain readability and limit the amount of collapse points. This context sensitive code collapsing feature is currently under development.

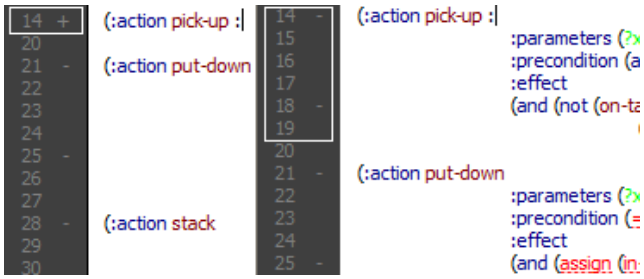

Figure 3: Code collapsing of an *action* pick-up block. The '-' sign is used to indicate collapse points, the '+' sign is used to indicate expansion points.

XML export and import

On one hand the PDDL language is hard to read and on the other hand it is hard to parse. We created our XML equivalent of the PDDL Language notation. The PDDL Studio can export and import this format for use by other applications or for better readability by other developers.

Planner integration

PDDL Studio provides the capability to integrate any planner which can be executed from a system command line – i.e. console prompt. We provide the user with our execution console which allows for project specific simple scripts (Figure 4).

These scripts are parsed and the result is executed via the system console. Various parameters can be used, e.g. current project directory, file names of PDDL files etc.

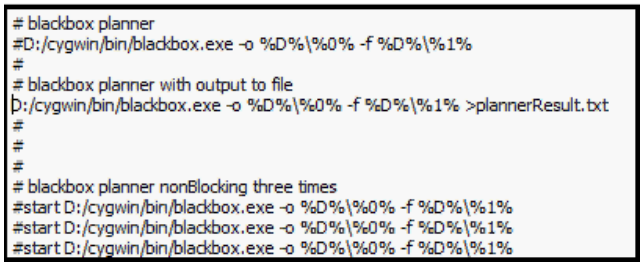

Figure 4: Planner integration using simple scripts executed in trough the system command line

Common Editor Features

The PDDL Editor also incorporates common editor features like *line numbering* and *bracket matching*. Line numbering is represented in the left portion of the screen. Bracket matching is used to identify bracket pairs by coloring them. It allows the user to detect missing brackets.

Conclusion and Future work

The PDDL Studio is a simple but powerful tool for creation and management of PDDL projects. We created it to mimic the behavior of the commonly and widely utilized IDEs like Visual Studio 2010 or Eclipse (in respect to code editing). The main features of this tool are the capability to locate and identify syntax and semantic errors in the PDDL document and provide semantic hints on code completion. The tool also provides features like syntax highlighting and code collapsing, which allow the user to read and inspect the code easily.

In the next version, we intend to integrate a more complex and capable project management system similar to the one present in VS10. We intend to extend our PDDL parser to be used with the newest version of PDDL. We also work on extending our semantic error detection and work on integrating our tool with a plan visualization and inspection tool and our own planner.

We also intend to provide the user with the capability to create custom templates for various purposes – e.g. action skeletons filled based on given or guessed parameters. We also want to provide automated on the fly indentation of documents. The user might request a view-only custom indentation of the displayed documents to suit his reading/writing style. We also want to incorporate a simple mechanism to invoke various actions (e.g. commenting of selected text portions, inserting custom text templates etc.) via user defined key combinations (e.g. Ctrl + Shift + F1).

Acknowledgement: This work was partially supported by a student grant GA UK No. 0449/2010/A-INF/MFF, by project P103/10/1287 (GA ČR) and GA UK No. 655012/2012/A-INF/MFF.

The application can be downloaded at: http://amis.mff.cuni.cz/PDDLStudio.

References

GNU Project. 2001. GNU Bison. http://www.gnu.org/software/bison (ver. 2.5 2011).

Hoffmann, J. 2003. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. Journal of Artificial Intelligence Research, Volume 20, pages 291 - 341.

Howey, R., Long, D., Fox, M. 2004. VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning using PDDL. Tools with Artificial Intelligence, ICTAI 2004

Kautz, H., Selman, B. 1998. BLACKBOX: A New Approach to the Application of Theorem Proving to Problem Solving. Working notes of the Workshop on Planning as Combinatorial Search, held in conjunction with AIPS-98, Pittsburgh, PA.

McDermott D., Ghallab M., Howe A., Knoblock C., Ram A., Veloso M.; Weld D., Wilkins D. 1998. PDDL – The Planning Domain Definition Language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, New Haven, CT.

Nau, D., Cao, Y., Lotem, A., Muñoz-Avila, H. 1999. *SHOP: Simple Hierarchical Ordered Planner*. In IJCAI-99, pp. 968-973.

Paxson, V. 1987. Flex Lexical Analyzer. http://flex.sourceforge.net (ver. 2.5.35 2008).

Qt Project. 1992. Qt. http://www.qt-project.org (ver. 4.8.0 2011).

Raimondi, F., Pecheur, C., Brat , G. 2009. *PDVer, a tool to verify PDDL planning domains*. In Proceedings of ICAPS'09 Verification and Validation of Planning and Scheduling Systems.

Simpson, R. M. 2007. Structural Domain Definition using GIPO IV. In Proceedings of the Second International Competition on Knowledge Engineering for Planning and Scheduling

Vaquero, T., S., Silva, J., R., Beck, J., C. 2011. A Brief Review of Tools and Methods for Knowledge Engineering for Planning & Scheduling. In: Proceedings of the Knowledge Engineering for Planning and Scheduling (KEPS) workshop. The 21th International Conference on Automated Planning & Scheduling (ICAPS 2011). Freiburg. Germany.

Vaquero, T. S., Silva, J. R., Ferreira, M., Tonidandel, F., Beck, J. C. 2009. *From Requirements and Analysis to PDDL in itSIMPLE3.0*. In Proceedings of the Third International Competition on Knowledge Engineering for Planning and Scheduling, ICAPS 2009, 54–61.

Vrakas, D. and Vlahavas, I. 2003. ViTAPlan: A Visual Tool for Adaptive Planning, In Proceedings of the 9th Panhellenic Conference on Informatics, Thessaloniki, Greece, pp. 167-177.