Challenges of Developing an API for Interactive Configuration using ASP

Richard Comploi-Taupe[1], Susana Hahn[2], Torsten Schaub[2], and Gottfried Schenner[1]

[1] Siemens AG Österreich, Vienna, Austria
{richard.taupe,gottfried.schenner}@siemens.com
[2] University of Potsdam, Germany, and Potassco Solutions
hahnmartinlu@uni-potsdam.de, torsten@cs.uni-potsdam.de

1 Introduction

Product configuration has been one of the first successful applications of Answer Set Programming (ASP [8,10]) [11]. Nonetheless, more than 20 years later, using ASP in a product configurator is still challenging. One open challenge is to allow interactivity during the configuration process.

Industrial product configuration needs to deal with large problems. For example, large infrastructure projects may contain thousands of components and hundreds of component types. These configurations are typically solved in a step-wise manner by combining interactive actions with automatic solving of sub-problems [5]. Such a process requires additional modularity to cope with different types of users. While domain experts provide the configuration model, other users, such as engineers and sales people, expect a system that allows them to guide the configuration process by interacting with the solver.

When using a grounding-based formalism like ASP in this context, the so-called grounding bottleneck [4] arises due to the large number of required components for satisfying all requirements. Furthermore this required domain size is not known beforehand and can vary significantly. Therefore, we require a way to dynamically introduce new components during the configuration process.

We developed an API to satisfy basic requirements for interactive configuration [5]. Our implementation is based on OOASP [6], a framework for representing object-oriented configurations in ASP. Additionally, we exploited multiple features of the ASP system *clingo*[3] [7] to provide interactive functionalities.

2 Approach

Our API was implemented using python, relaying heavily on multiple features provided by *clingo*'s API, as well as the systems *clorm*[4] and *clingraph* [9]. *Clorm* is a python library providing an Object Relational Mapping (ORM) interface to

[3] https://potassco.org/clingo/
[4] https://github.com/potassco/clorm

clingo, which we use to map the OOASP predicates defining the knowledge base an the configuration into python classes. These elements were then visualized as graphs (resembling UML diagrams) using *clingraph*. Additionally, we employed our API to create a prototype UI using *ipywidgets*.

The basic idea behind our approach is to modularize the encodings so that the program can be built incrementally as the domain size increases. We use the multi-shot capabilities of *clingo* for solving continuously changing logic programs. This approach avoids re-grounding and benefits from learned constraints by grounding and solving on demand. More specifically, we defined subprograms that depend on the identifier of the newly introduced object. Therefore, whenever the domain size is extended by a new object, all the rules referring to this object are grounded. In this sense, this approach differs from the previous work [2], in which subprograms were subject to domain-specific actions.

During the interactive process a user will edit a partial configuration $\mathcal{P}$ to build a complete configuration $\mathcal{C}$. The user is able to edit $\mathcal{P}$ by adding and removing: the type of an existing object, associations between two objects, and values for attributes. Such editions are done using external atoms in *clingo*, so that no re-grounding is required. Thus, grounding is only necessary when the user extends the configuration with a new object. Then, only the sub-program for the new object identifier is grounded and added to the ground program.

We identified three reasoning tasks in which solving was necessary. (1) Using the current objects to generate $\mathcal{C}$ from $\mathcal{P}$ using choice rules; (2) obtaining the list of available editions for the user via brave reasoning; (3) checking if $\mathcal{P}$ is complete or if it violates any constraints. These tasks are distinguished within the encoding via externals. In this case one external atom states that the guessing of the objects' types, associations and values is active, while two additional externals activate the integrity constraints for the constraint violations of partial and total constraints, respectively.

Due to the interactive requirements we divided constraints into two types; partial and total constraints. Partial constraints are those that can be corrected by adding more information in a latter stage of the process, for instance, a lower bound of an association that has not been reached, or a value that is missing. On the other hand, total constraints refer to violations that can not longer be fixed, such as upper bounds of an association or having a value of a wrong type. Unlike the edition externals, constraint externals are set internally by the system to differentiate the task at hand within the same program.

Given all these functionalities, finding the smallest $\mathcal{C}$ that extends $\mathcal{P}$ becomes trivial. This is done following an incremental approach: $\mathcal{P}$ is extended with a new object and then the solver attempts to find $\mathcal{C}$, the process is repeated until $\mathcal{C}$ is found. By following this process one must prove unsatisfiability many times before finding the solution, leading to performance issues.

Addressing these performance issues to make the approach scale better is subject of future work. Furthermore, we intend to explore different means to extend $\mathcal{P}$, such as scheduling techniques [3] or pre-computing the minimal number of required objects [1].

References

1. Aschinger, M., Drescher, C., Gottlob, G., Vollmer, H.: Loco – A logic for configuration problems. ACM Trans. Comput. Log. **15**(3), 20:1–20:25 (2014). https://doi.org/10.1145/2629454
2. Comploi-Taupe, R., Francescutto, G., Schenner, G.: Applying incremental answer set solving to product configuration. In: Proceedings of the 26th ACM International Systems and Software Product Line Conference – Volume B. pp. 150–155. Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3503229.3547069, https://doi.org/10.1145/3503229.3547069
3. Dimopoulos, Y., Gebser, M., Lühne, P., Romero, J., Schaub, T.: plasp 3: Towards effective ASP planning. In: Balduccini, M., Janhunen, T. (eds.) Logic Programming and Nonmonotonic Reasoning – 14th International Conference, LPNMR 2017, Espoo, Finland, July 3-6, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10377, pp. 286–300. Springer (2017). https://doi.org/10.1007/978-3-319-61660-5_26
4. Eiter, T., Faber, W., Fink, M., Woltran, S.: Complexity results for answer set programming with bounded predicate arities and implications. Ann. Math. Artif. Intell. **51**(2-4), 123–165 (2007). https://doi.org/10.1007/s10472-008-9086-5
5. Falkner, A.A., Haselböck, A., Krames, G., Schenner, G., Schreiner, H., Taupe, R.: Solver requirements for interactive configuration. J. Univers. Comput. Sci. **26**(3), 343–373 (2020), http://www.jucs.org/jucs_26_3/solver_requirements_for_interactive
6. Falkner, A.A., Ryabokon, A., Schenner, G., Shchekotykhin, K.M.: OOASP: connecting object-oriented and logic programming. In: Calimeri, F., Ianni, G., Truszczynski, M. (eds.) Logic Programming and Nonmonotonic Reasoning – 13th International Conference, LPNMR 2015, Lexington, KY, USA, September 27-30, 2015. Proceedings. Lecture Notes in Computer Science, vol. 9345, pp. 332–345. Springer (2015). https://doi.org/10.1007/978-3-319-23264-5_28
7. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Multi-shot ASP solving with clingo. Theory Pract. Log. Program. **19**(1), 27–82 (2019). https://doi.org/10.1017/S1471068418000054
8. Gelfond, M., Kahl, Y.: Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach. Cambridge University Press, New York, NY, USA (2014)
9. Hahn, S., Sabuncu, O., Schaub, T., Stolzmann, T.: Clingraph: ASP-based visualization. In: Gottlob, G., Inclezan, D., Maratea, M. (eds.) Logic Programming and Nonmonotonic Reasoning – 16th International Conference, LPNMR 2022, Genova, Italy, September 5-9, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13416, pp. 401–414. Springer (2022). https://doi.org/10.1007/978-3-031-15707-3_31
10. Lifschitz, V.: Answer Set Programming. Springer (2019). https://doi.org/10.1007/978-3-030-24658-7
11. Soininen, T., Niemelä, I., Tiihonen, J., Sulonen, R.: Representing configuration knowledge with weight constraint rules (2001), http://www.cs.nmsu.edu/%7Etson/ASP2001/20.ps