




Progress on an ASP-based Interactive Configurator

Lucia Balážová¹, Richard Comploi-Taupe¹ ,
Susana Hahn² , and Nicolas Rühling² 

¹ Siemens AG Österreich, Vienna, Austria
{balazova.lucia,richard.taupe}@siemens.com

² University of Potsdam, Germany, and Potassco Solutions
{hahnmartinlu,nruehling}@uni-potsdam.de

1 Introduction

Configuration [8] has been one of the first successful applications [7, 10, 12] of Answer Set Programming (ASP; [11, 13]). Nonetheless, there remain many open challenges for ASP-based product configurators, one of them being interactive configuration.

Industrial configuration problems in large infrastructure projects may contain thousands of components and hundreds of component types and are typically solved in a step-wise manner by combining interactive actions with automatic solving of sub-problems [5]. When using a grounding-based formalism like ASP, the so-called grounding bottleneck [4] arises due to the large number of components. Furthermore, the necessary domain size is not known beforehand and can vary significantly. Therefore, we require a way to dynamically introduce new components during the configuration process.

In our previous work we developed an API to satisfy basic requirements for interactive configuration [5] based on OOASP [6] and using various features of *clingo*³ [9]. In this extended abstract, we report on our recent advancements in developing an ASP-based domain-independent interactive configuration platform. The new achievements, compared to previously published reports [2, 3], consist of dramatically improved solving performance by using novel so-called *smart expansion* techniques and the creation of a UI prototype based on *clinguin* [1].

2 Novel Contributions

We have aimed to accomplish the interactive tasks outlined in [2], focusing on improving the performance of extending a partial configuration (\mathcal{P}) to a complete one (\mathcal{C}) (T8 [2]). We have achieved this by enhancing the classical incremental approach for multi-shot solving with four different smart expansion functions. The idea is that before checking for the existence of \mathcal{C} with the current objects, we calculate cautious and brave consequences (intersection and union of stable

³ <https://potassco.org/clingo/>

models, respectively) and subsequently apply the smart expansion functions which determine and add necessary specific objects or associations to \mathcal{P} until no more information can be extracted. This approach limits the number of costly unsatisfiability checks and reduces the search space by adding type-specific objects and associations.

All smart expansion functions exploit knowledge about UML class diagrams, a common way to represent object-oriented (configuration) problems [8]. Due to space constraints, we explain here only one and refer to the source code for further details.⁴ As in [2], we use the racks example depicted in Appendix A.

The first function, *assoc_needs_object*, identifies the need to add objects of a specific type and associate them to an existing object. Consider the partial configuration formed by a rack r and a frame f with no associations. A rack needs to be associated to at least four frames, which means that even if r gets associated to f , it needs at least three more frames in the configuration to be associated to. This information is present in the cautious consequences with three target atoms indicating that at least $X \in \{1, 2, 3\}$ objects of type frame are needed. The smart function evaluates this information by taking the maximum of X and subsequently grounds three more frames which are associated to object r via association `rack_frames`. We use multiple atoms under the notion of “at least” and leverage the intersection of models to discard target atoms from models where possible associations with existing objects were not considered. For instance, the model where r and f are not associated yields target atoms for $X \in \{1, 2, 3, 4\}$, but when intersected with the target atoms of the model including the association, $X = 4$ is discarded.

Two of the other smart functions, namely, *global_ub_gap* and *global_lb_gap*, make use of cautious consequences as well but do not refer to specific objects. Instead they calculate gaps between upper (respectively, lower) bounds for each object type and the actual number of existing objects, thus, adding global objects without any specific associations.

Finally, the function *association_possible* reduces the search space by establishing viable associations for completing the configuration using brave consequences. This can have effects on the completeness of solutions by limiting options, however it speeds up the process of finding/estimating the minimal domain size.

In addition, by using *clinguin* for the UI, many interactive tasks (T1-3, T5, and T7 [2]) were directly integrated into the ASP UI encoding. The final prototype enhances usability with features such as saving/loading configurations and a clear button, while utilizing *clinguin*’s integration with *clingraph* to visualize and interact with the configuration graph. A snapshot of the UI is shown in Figure 3 in Appendix C.

Our experimental results depicted in Figure 2 in Appendix B show that the smart functions significantly increase performance overall. A drawback of smart functions is that they do not perform well with additional domain-specific constraints that cannot be expressed solely by associations. How to improve this situation will be investigated in future work.

⁴ <https://github.com/siemens/00ASP>

Acknowledgements

This research has been partially supported by the Austrian Research Promotion Agency (FFG) as part of the “AI for Green” programme.

References

1. Beiser, A., Hahn, S., Schaub, T.: ASP-driven user-interaction with clinguin. In: Cabalar, P., Swift, T. (eds.) Technical Communications of the Fortieth International Conference on Logic Programming (ICLP’24). EPTCS (2024)
2. Comptoi-Taupe, R., Falkner, A., Hahn, S., Schaub, T., Schenner, G.: Interactive configuration with ASP multi-shot solving. In: Horcas, J., Galindo, J., Comptoi-Taupe, R., Fuentes, L. (eds.) Proceedings of the Twenty-fifth International Configuration Workshop (CONF’23). vol. 3509, pp. 95–103. CEUR Workshop Proceedings (2023), <https://ceur-ws.org/Vol-3509/paper13.pdf>
3. Comptoi-Taupe, R., Hahn, S., Schenner, G., Schaub, T.: Challenges of developing an API for interactive configuration using ASP. In: Tarzariol, A., Laferrière, F., Saribatur, Z. (eds.) Proceedings of the Fifth Workshop on Trends and Applications of Answer Set Programming (TAASP’22) (2022), http://www.kr.tuwien.ac.at/events/taasp22/papers/TAASP_2022_paper_5.pdf
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., Haselböck, A., Krames, G., Schenner, G., Schreiner, H., Taupe, R.: Solver requirements for interactive configuration. *Journal of Universal Computer Science* **26**(3), 343–373 (2020). <https://doi.org/10.3897/jucs.2020.019>
6. Falkner, A., Ryabokon, A., Schenner, G., Shchekotykhin, K.: OOASP: connecting object-oriented and logic programming. In: Calimeri, F., Ianni, G., Truszczyński, M. (eds.) Proceedings of the Thirteenth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR’15). Lecture Notes in Artificial Intelligence, vol. 9345, pp. 332–345. Springer-Verlag (2015). https://doi.org/10.1007/978-3-319-23264-5_28
7. Felfernig, A., Falkner, A., Atas, M., Erdeniz, S., Uran, C., Azzoni, P.: ASP-based knowledge representations for IoT configuration scenarios. In: Zhang, L., Haag, A. (eds.) Proceedings of the Nineteenth International Configuration Workshop (CONF’17). pp. 62–67 (2017)
8. Felfernig, A., Hotz, L., Bagley, C., Tiihonen, J. (eds.): Knowledge-Based Configuration: From Research to Business Cases. Elsevier/Morgan Kaufmann (2014). <https://doi.org/10.1016/C2011-0-69705-4>
9. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Multi-shot ASP solving with clingo. *Theory and Practice of Logic Programming* **19**(1), 27–82 (2019). <https://doi.org/10.1017/S1471068418000054>
10. Gebser, M., Kaminski, R., Schaub, T.: aspcud: A Linux package configuration tool based on answer set programming. In: Drescher, C., Lynce, I., Treinen, R. (eds.) Proceedings of the Second International Workshop on Logics for Component Configuration (LoCoCo’11). Electronic Proceedings in Theoretical Computer Science (EPTCS), vol. 65, pp. 12–25 (2011). <https://doi.org/10.4204/eptcs.65.2>
11. Gelfond, M., Kahl, Y.: Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach. Cambridge University Press (2014)

12. Gençay, E., Schüller, P., Erdem, E.: Applications of non-monotonic reasoning to automotive product configuration using answer set programming. *Journal of Intelligent Manufacturing* **30**, 1407–1422 (2019). <https://doi.org/10.1007/s10845-017-1333-3>
13. Lifschitz, V.: *Answer Set Programming*. Springer-Verlag (2019). <https://doi.org/10.1007/978-3-030-24658-7>

A The Racks Example

Figure 1 shows a UML class diagram for the racks knowledge base generated by *clingraph*. The “hardware racks” domain is a typical example of an industrial configuration problem [6]. The problem consists of instantiating various hardware components and associating them to each other while satisfying several constraints: Elements are controlled by modules of different types, which are placed in frames, which are located in racks.

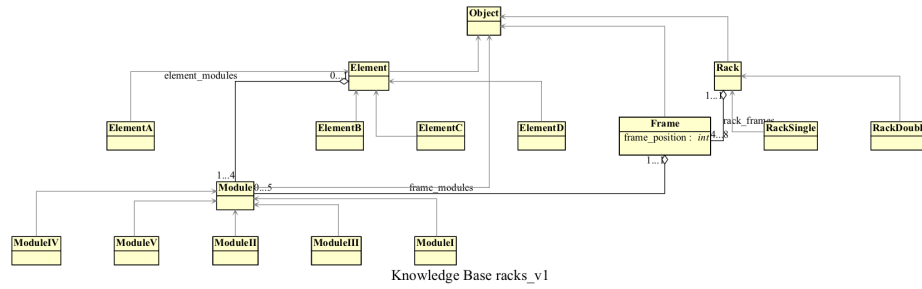


Fig. 1. UML class diagram for the racks knowledge base generated by *clingraph*.

B Experimental Results

As visualized in Figure 2, the smart expansion functions provide a significant improvement in terms of performance. This decrease in runtime makes the system capable of completing much more complex partial configurations than the previous version [2]. However, smart functions underperformed in cases where additional domain-specific constraints were imposed (by this we mean constraints that cannot be expressed solely by associations, such as that a Frame that contains a ModuleII additionally requires a ModuleV).

Combinations and order of the smart functions can be tailored to a particular problem (not shown in the figure). On average, combinations of all smart functions starting with *association_possible* provide the best results. In our benchmarking the combination *association_possible*, *assoc_needs_object*, *global_ub_gap*, *global_lb_gap* was used.

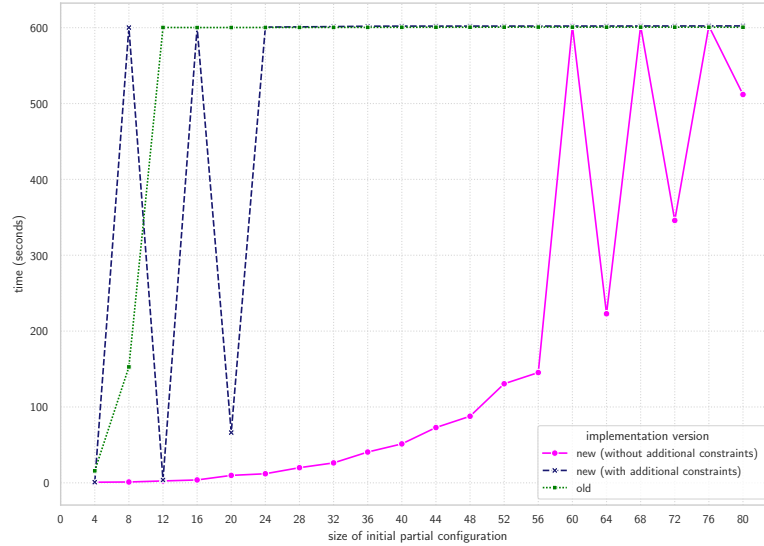


Fig. 2. Runtime comparison (time-out: 600s).

C UI Prototype

Figure 3 shows a screenshot of our *clinguin*-based UI prototype. This interface provides multiple improvements in terms of usability compared to the previous version [2], such as the ability to both save and load existing complete and partial configurations in the form of sets of facts. Furthermore, the user can resolve some constraint violations instantly by pressing the violation warnings in the bottom left panel. The objects are visualized dynamically upon addition and can be selected by directly clicking on them. The visualization also highlights the associated objects to help the user find connected objects in larger configurations and uses different colors of text to emphasize the type of information they convey.

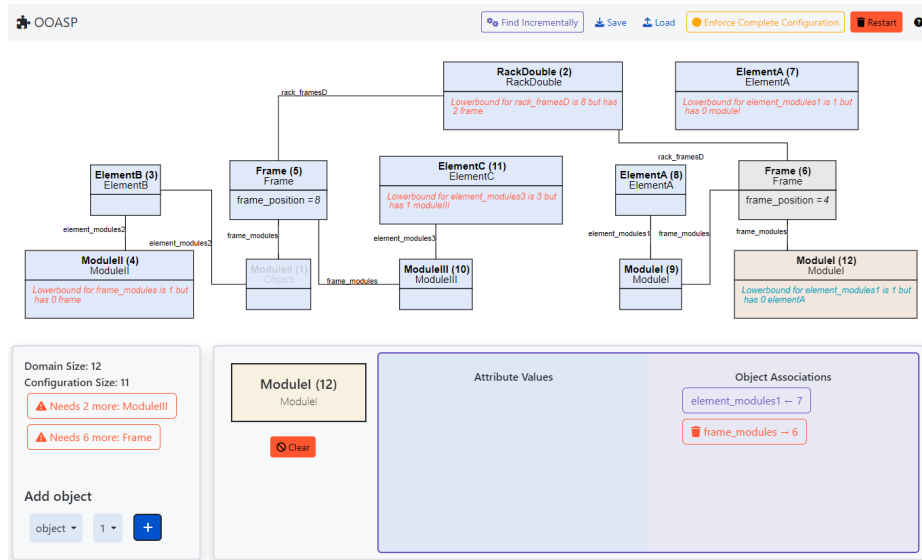


Fig. 3. Example of usage of the *clingin* GUI for the Interactive Configurator.