# Automated Design of Complex Systems with Constraint Programming Techniques

Stefano Demarchi[1], Marco Menapace[2], and Armando Tacchella[2]

[1] Dipartimento di Chimica e Farmacia, Università degli Studi di Sassari
Via Vienna, 2 - 07100 Sassari, Italy
sdemarchi@uniss.it
[2] DIBRIS, Università degli Studi di Genova
Via Opera Pia, 13 - 16145 Genova, Italy
armando.tacchella@unige.it marco.menapace@edu.unige.it

**Abstract.** Automated design and configuration of complex systems is well established as a research topic, specifically for constraint programming. In order to encode "quality of design" rules, i.e., preferences about the final outcomes, the encoding must be completed with optimization capabilities. Based on experience in the configuration of elevator systems we present a research agenda to develop methods and tools which should be able to leverage constraint solvers in order to compute designs for complex systems that can compete with the ones produced by humans. Besides elevator systems, we propose to evaluate the methodology and the tools on other case studies, e.g., hardware configuration or heating-ventilation and air conditioning systems.

**Keywords:** Product configuration, Product design, Constraint programming, Computer-automated Design



## 1 Introduction

In this paper we present and discuss the intention to build a tool able to handle both configuration and design, e.g, in the terms of "quality" of the configuration satisfaction [1] relying on the principles of constraint programming. This tool would automate both configuration — where given a system and associated constraints it is produced, if possible, a feasible result — and design — the actual step forward in the state-of-the-art. This "high-level configurator" could help non-programmers to experiment with a model by dynamically adding, removing or editing configuration constraints. In order to achieve this interactivity, a framework allowing models representation and custom configurations is crucial. In the end, the software ideally would not require expensive patches due to changes in requirements but could be reconfigured just by changing the encoding. The formulation we propose in order to achieve this goal is based on *Satisfiability Modulo Theories* (SMT for short) [2]. To the extent of our knowledge, SMT provides enough expressiveness to encode the configuration task as the feasibility of a set of constraints. Using SMT instead of, e.g., mathematical

programming, or constraints solving, gives us more flexibility in the choice of the encodings, which can include a variety of (decidable) theories in first order logic, combined through a Boolean structure. SMT can be extended with theories of cost yielding *Optimization Modulo Theories* (OMT for short) [3]. Using an OMT solver we can deal with both feasibility and optimality target at the same time, i.e., we can handle both configuration and design tasks with a single engine. The key idea is to encode the design problem into (*i*) a set of constraints on some decision variables, and (*ii*) a cost function that expresses "quality-of-design" rules.

In Section 2, after a short context definition, we focus on elevators domain, which is our success story covered in Section 3. Encouraged by our results, we propose in Section 4 a detailed roadmap for the development of an automated configuration tool. Finally, in Section 5 we draw some conclusions on both the work carried out until now and the methodologies for future contribution.

## 2 Motivation

### 2.1 Context

Product configuration has been a fruitful topic of research in Artificial Intelligence (AI) since 1970, and also became a commercially successful application of AI techniques. In the recent past, review and survey papers have been reported. With focus on specific topics on product configuration, these papers provide rich information and knowledge in the corresponding areas, such as configuration benefits and challenges, configuration modeling and solving, configuration knowledge handling and industrial cases, managerial issues related to configuration in practice and theory testing of product configuration with respect to time and quality performance. [4]

Configuration problems have been treated also as a class of *constraint satisfaction problems* (CSPs) — see, e.g., in [5] or [6] — appearing extremely suitable for treating a design task. In this setup, both the model and its feasibility can be mapped with a CSP model and relative constraints. In a static configuration problem, the decision variables, the domain and the constraints are known in advance and fixed, therefore the problem of configuration solving can be encoded as a (standard) CSP — see, e.g., [7]. In a dynamic configuration problem, depending on the assignment to the corresponding decision variable, a component may have a varying number of properties, whose existence is triggered by *activity constraints*. Handling dynamic configuration tasks requires *generative CSP* (GCSP) formulations[3] introduced in [9] specifically to handle dynamic configuration solving.

Satisfiability Modulo Theories is the problem of deciding the satisfiability of a first-order formula with respect to some decidable theory $\mathcal{T}$. In particular, SMT generalizes the boolean satisfiability problem (SAT) by adding background theories such as the theory of real numbers, the theory of integers, and the

[3] GCSP is also defined as *Conditional CSP* in [8] and *Dynamic CSP* in [9].

theories of data structures (*e.g.*, lists, arrays and bit vectors) — see, e.g., [2] for details. To decide the satisfiability of an input formula $\varphi$ in conjunctive normal form, SMT solvers typically first build a *Boolean abstraction* $abs(\varphi)$ of $\varphi$ by replacing each constraint by a fresh Boolean variable (proposition), e.g.,

$$\begin{array}{rl} \varphi & : \underbrace{x \geq y} \wedge (\ \underbrace{y > 0} \vee \underbrace{x > 0}\ ) \wedge \underbrace{y \leq 0} \\ abs(\varphi) & : \quad A \quad \wedge (\quad B \quad \vee \quad C \quad ) \wedge \quad \neg B \end{array}$$

where $x$ and $y$ are real-valued variables, and $A$, $B$ and $C$ are propositions. A propositional logic solver searches for a satisfying assignment $S$ for $abs(\varphi)$, e.g., $S(A) = 1$, $S(B) = 0$, $S(C) = 1$ for the above example. If no such assignment exists then the input formula $\varphi$ is unsatisfiable. Otherwise, the consistency of the assignment in the underlying theory is checked by a *theory solver*. In our example, we check whether the set $\{x \geq y,\ y \leq 0,\ x > 0\}$ of linear inequalities is feasible, which is the case. If the constraints are consistent then a satisfying solution (*model*) is found for $\varphi$. Otherwise, the theory solver returns a theory lemma $\varphi_E$ giving an *explanation* for the conflict, e.g., the negated conjunction some inconsistent input constraints. The explanation is used to refine the Boolean abstraction $abs(\varphi)$ to $abs(\varphi) \wedge abs(\varphi_E)$. These steps are iteratively executed until either a theory-consistent Boolean assignment is found, or no more Boolean satisfying assignments exist.

Adding theories of cost to SMT yields Optimization Modulo Theories (OMT), an extension that finds models to optimize given objectives through a combination of SMT and optimization procedures [3]. For example,

$$\begin{cases} \varphi : x \geq y \wedge (y > 0 \vee x > 0) \wedge y \leq 0 \\ \min_{x,y}(x + y) \end{cases}$$

requires all the constraints in $\varphi$ to be satisfied and the additional cost $x + y$ to be minimized. Notice that OMT extends classical formulations in mathematical programming, e.g., linear programming or mixed integer linear programming, since it allows Boolean structure to be taken into account together with the optimization target. OMT solvers have been developed for several first-order theories like, e.g., those of linear arithmetic over the rationals ($LRA$) or the integers ($LIA$) or their combination ($LIRA$).

### 2.2 A primer on elevator systems

Elevator systems are characterized by different lifting mechanisms, e.g., ropes and pistons, and different setups, e.g., one or two pistons, presence or absence of counterweights, and dedicated machine room versus in-plant machine room. In our work we focused on hydraulic elevators (HEs), which operate on hydraulics, i.e., they rely on one or more pistons to move the car. Energy is usually provided to the hydraulic fluid by an electrically driven pump, and typically no counterweight is needed to compensate for the weight of the car. Their low initial costs, compact footprint and ease of installation makes them a viable choice

for retrofitting old residential buildings, and a cost-effective solution for new ones alike. The choice of HEs as a case study is thus motivated by their popularity in low-rise applications, and by the fact that, in spite of their relative low part count, their structure presents already most of the challenges that are to be found in this domain.

## 3 Elevator Design

### 3.1 Model

Figure 1 shows the top view of a technical drawing produced by our tool LiftCreate[4]. The enclosure space in which all components are placed is the *shaft*, and the *car* is sustained on the left by the *car frame*, a mechanic gear that runs thanks to a vertical piston and a set of ropes on a pulley. At the bottom of the drawing it is visible a pair of doors with three sliding panels each, namely the *car door* at the top and the *landing door* at the bottom. The car door is one for the whole elevator, and runs with the car, while there is one landing door for each floor. The car frame and the car-landing door pair are the components upon which the whole elevator design hinges. The reference system origins from the top left corner of the shaft and the $y$ axis is inverted with respect to canonical Cartesian systems. In the drawing we can see that the car frame structure is comprised of the *brackets* — visible in the picture as wall-mounted T-shaped supports — which support the *car rails* on which the car frame *core gear* runs. The car frame *base point*, i.e., the insertion point of the car frame structure in the drawing, lies on the outer corner of the topmost bracket and it is marked with a circle. The coordinates $(x_{cf}, y_{cf})$ of the car frame base point determine a specific placement of the structure. The *overhang* of the car with respect to the car frame is the distance from the car walls to the car frame core gear; *dcr* is the distance between the car rails, i.e., the size of the core gear. The base point of the car door is denoted as $(x_{cd}, y_{cd})$ and the base point of the landing door is denoted as $(x_{ld}, y_{ld})$: they are always at the top left corner of the corresponding structure. The value of these coordinates represent a specific placement of the car-landing door pair. The *opening* of the car is the available space when entering the elevator and it is surrounded on the landing door by the *frame*, i.e., the structure that surrounds the entrance to the car. The door structure is divided in two parts starting from the midpoint of the opening, in which we distinguish a *left axis* from a *right axis*.

In Tables 1 and 2 we summarize all the quantities involved in the elevator configuration problem, highlighting the actual decision variables and the parameters that are related to the components database.

[4] Part of the AiLift suite available upon request at http://www.ailift.it. AiLift is a web-based application for the automated design of elevator systems.

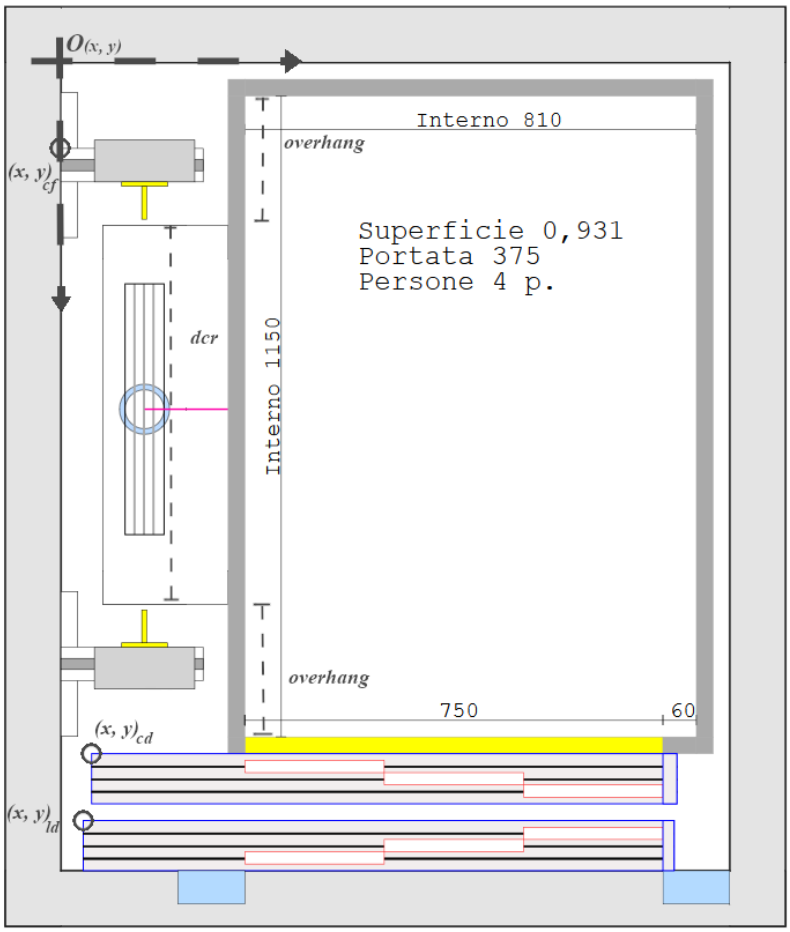


**Fig. 1.** Top view of a configured HE. The shaft is outlined by a gray enclosure in which the components are placed. This elevator has the car frame on the left side and a single door which opens on the bottom of the drawing. The opening area is outlined in the bottom car wall.

### 3.2 Encoding

Those components must satisfy a set of constraints that guarantee the design compliance with respect to directive 2014/33/EU and related EN 81-20/81-50 norms. The problem of configuring this kind of system is that for both car frame and doors are selected from a database of existing components from different vendors and with different parameters.

Feeding a constraint solver with the domain outlined before yields a feasible configuration, if any. We may as well be interested in enumerating feasible con-

**Table 1.** Explanation of the decision variables

| Symbol | Description |
|---|---|
| $x_{cf}$, $y_{cf}$ | Car frame base point coordinates |
| $x_{cd}$, $y_{cd}$ | Car door base point coordinates |
| $x_{ld}$, $y_{ld}$ | Landing door base point coordinates |
| $x_{car}$, $y_{car}$ | Car base point coordinates |
| $w_{car}$, $d_{car}$ | Car width and depth |

**Table 2.** Explanation of the design parameters

| Symbol | Description |
|---|---|
| $x_{shaft}$, $y_{shaft}$ | Shaft base point coordinates |
| $w_{shaft}$, $d_{shaft}$ | Shaft width and depth |
| $red_{[N,E,S,W]}$ | Distance between shaft and car walls (North, East, South, West) |
| $cwt_{[N,E,S,W]}$ | Car wall thickness (North, East, South, West) |
| $w_{cf}$ | Distance from $x_{cf}$ to the left car wall |
| $d_{cf}$ | External distance between car frame rails |
| $y_{gear}$ | Core gear placement with respect to the car frame base point |
| $d_{br}$ | External depth of the car frame bracket from the base point |
| $dcr$ | Distance between car rails |
| $max_{oh}$ | Maximum car overhang that the car frame is able to sustain |
| $opening$ | Doors opening |
| $la_{cd}$, $ra_{cd}$ | Left and right axis size (car door) |
| $la_{ld}$, $ra_{ld}$ | Left and right axis size (landing door) |
| $step_{cd}$ | Car door step |
| $step_{ld}$ | Landing door step |
| $d_{step}$ | Distance between doors |
| $w_{frame}$ | Landing door external frame width |

figurations, or in pinpointing "optimal" configurations, i.e., designs which have desirable properties according to some best principles and practices in elevator engineering. In Table 3 we show a quick summary of the combinations among feasibility (**SAT**) and optimality (**OPT**) targets with search for a single assignment (**Search**) and enumeration of configurations (**Enum**). The two questions that we found most relevant for practical purposes are feasibility and optimality in a search context. Checking whether a feasible configuration exists is useful in all the cases in which the size of the shaft and/or a restricted availability of components makes the design hardly feasible. Finding an optimal configuration, assuming that several alternative configurations exists, is about stepping from plain configuration problems to full-fledged automated design. However, in order to achieve optimality, we need to instruct the solver not just about what *cannot* be done, but also about what *should* be done to obtain a "good" configuration.

**Table 3.** Alternative configuration problems.

| | **Search** | **Enum** |
|---|---|---|
| **SAT** | Is there a feasible configuration? | How many feasible configurations are there for each choice of car frame and door? |
| **OPT** | Is there an optimal configuration? | What is the optimal configuration for each choice of car frame and door? |

We encoded this configuration problem with the SMT formalism, not only providing the feasibility constraints but also connecting the system with the components database; in this way, the solver has the duty to select those components (car frame and doors) whose parameters allow a feasible (possibly optimal) design. This encoding is embedded in a Java application which orchestrates at an higher level the components relationships — the whole problem is still bigger than this module — and provides database transactions and drawing services.

## 4 Research roadmap

In order to extend our work towards system-independent design, we need to build a framework in which the configurator is accompanied by another system providing the correct interpretation and translation of a generic model in a set of variables, constraints and cost function(s).

Ontology-based modeling is widely used in order to build consistent models from a structured description of a system: *Ontology Web Language* (OWL) [12] grants the modeling capabilities to express the structure of any system [13] [14] [15] and should help interfacing the core configurator with any model described in this way. With this model interface we will build a first prototype of the core configurator using our elevator configurator as a baseline.

As soon as the system implementation will be adequately ready, an experimental analysis campaign will test the prototype development. The availability of a research prototype will enable such analysis, targeted to assess the quality of the proposed solution. We expect interactions between experimental part and implementation part in a sense that implemented techniques may be subject to modifications and refinements based on the experimental findings. From this prototype release on, another test campaign will accurately cover the quality and correctness of the application. The evaluation will be carried out considering computational times as well as evaluation of the computer generated configuration designs by human experts of the model tested, in order to also assess the quality and correctness of the generated solutions. In order to provide reliable benchmarks we will test the prototype on our well-known application of elevators, but we also identified more uncorrelated domains like hardware con-

figuration and heating/ventilation systems. Hardware configuration refers to the problem of choosing, connecting and verifying a set of components considering multiple constraints related to physical occupancy, power consumption, equivalent circuits and memory availability just to mention some, and the domain also allows the re-configuration of an already configured system where some constraints change and some components can change their connections. Heating/ventilation systems, on the other hand, are likely to present few constraints related to the number of machines and their parameters but pose a more important weight on the optimization of, e.g., number of radiators, power consumption, overall cost.

At the end of experimental analysis we expect that the prototype will be able to produce feasible configurations of a given system and will be comparable with other constraint expression solutions. The ideal conclusion of the proposed system is the implementation of an interactive interface, i.e., making possible to interactively add, remove or edit constraints to the model in analysis. With this feature the system would then let the user experience a complete and controlled design process in which not only hard, model-relative feasibility constraints are present but it is also possible to add custom refinements on the fly.

## 5 Conclusion

In the context of our LiftCreate design automation tool, we presented a constraint-based encoding to solve configuration and design for hydraulic elevators, which proved successful and pushes us to continue the integration of such methodology until the — possibly — complete design process with constraint-based formulations.

This success story leads to our agenda toward a model-independent automated configurator for both configuration and design targets, creating a framework in which a generic model expressed in an almost standard language can be interpreted, translated in the core formulation and processed in order to obtain a design for the requested cost function.

## References

1. David C. Brown. Defining configuring. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 12(4):301–305, 1998.
2. Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Solving sat and sat modulo theories: from an abstract davis–putnam–logemann–loveland procedure to dpll (t). *Journal of the ACM (JACM)*, 53(6):937–977, 2006.
3. Roberto Sebastiani and Patrick Trentin. On optimization modulo theories, maxSMT and sorting networks. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 231–248. Springer, 2017.
4. Linda L Zhang. Product configuration: a review of the state-of-the-art and future research. *International Journal of Production Research*, 52(21):6381–6398, 2014.

5. SM Fohn, JS Liau, AR Greef, RE Young, and PJ O'grady. Configuring computer systems through constraint-based modeling and interactive constraint satisfaction. *Computers in Industry*, 27(1):3–21, 1995.
6. Michel Aldanondo, Khaled Hadj-Hamou, Guillaume Moynard, and Jacques Lamothe. Mass customization and configuration: Requirement analysis and constraint based modeling propositions. *Integrated Computer-Aided Engineering*, 10(2):177–189, 2003.
7. Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.
8. Mihaela Sabin, Eugene C Freuder, and Richard J Wallace. Greater efficiency for conditional constraint satisfaction. In *International Conference on Principles and Practice of Constraint Programming*, pages 649–663. Springer, 2003.
9. Sanjay Mittal and Brian Falkenhainer. Dynamic constraint satisfaction. In *Proceedings eighth national conference on artificial intelligence*, pages 25–32, 1990.
10. Clark Barrett and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of Model Checking*, pages 305–343. Springer, 2018.
11. Arnaud Hubaux, Dietmar Jannach, Conrad Drescher, Leonardo Murta, Tomi Mannisto, Krzysztof Czarnecki, Patrick Heymans, Tien Nguyen, and Markus Zanker. Unifying software and product configuration: A research roadmap. *Proceedings of the 2012 International Conference on Configuration - Volume 958*, 2012.
12. Deborah L McGuinness, Frank Van Harmelen, et al. Owl web ontology language overview. *W3C recommendation*, 10(10):2004, 2004.
13. Liana Razmerita, Albert Angehrn, and Alexander Maedche. Ontology-based user modeling for knowledge management systems. In Peter Brusilovsky, Albert Corbett, and Fiorella de Rosis, editors, *User Modeling 2003*, pages 213–217, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
14. Ming Dong, Dong Yang, and Liyue Su. Ontology-based service product configuration system modeling and development. *Expert Systems with Applications*, 38(9):11770 – 11786, 2011.
15. X. H. Wang, D. Q. Zhang, T. Gu, and H. K. Pung. Ontology based context modeling and reasoning using owl. In *IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second*, pages 18–22, March 2004.