An Extensible Synthesis Framework

Theodor Lettmann Benno Stein
lettmann@upb.de stein@upb.de

Paderborn University
Department of Computer Science
D-33095 Paderborn, Germany

Abstract This paper describes the background and ideas of an on-going development at our institute: The creation of a framework that comprises state-of-the-art configuration and synthesis technology.

Key words: configuration framework, design problem solving, synthesis tasks

1 Introduction

Starting point of a design problem is a space $\mathcal{S}$ of possible design solutions along with a set D of demands. Solving a design problem means to determine a system $S^* \in \mathcal{S}$ that fulfills D. Typically, S^* is not found by experimenting in the real world but by operationalizing a search process after having mapped the system space, $\mathcal{S}$, onto a model space $\mathcal{M}$. $\mathcal{M}$ comprises all models M that could be visited during the design process.[1] Figure 1 illustrates these connections.

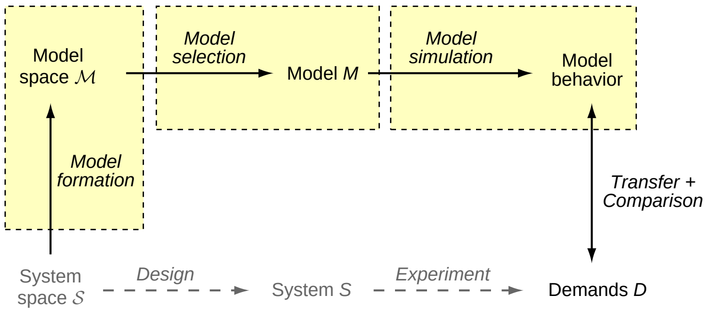

Figure 1. A generic scheme of design problem solving according to [16]: Given is a space $\mathcal{S}$ of possible design solutions and a set of demands D. On a computer, $\mathcal{S}$ is represented as a model space, $\mathcal{M}$, wherein a model M^* is searched whose behavior fulfills D.

It is the job of a design algorithm to efficiently find a model $M^* \in \mathcal{M}$ whose simulation produces a behavior that complies with D and which optimizes a possible goal criterion. A synthesis framework can support this job at several places:

[1] Following Minsky we call M a model of a system S, if M can be used to answer questions about S [9]. In particular in connection with design problems M may establish a structural, a functional, an associative, or a behavioral model.

1. *Model Formation.* Given a system space $\mathcal{S}$ there exist several paradigms to define a suited model space $\mathcal{M}$ for synthesis purposes. They include differential equations [6], taxonomies (is-a relations), compositional hierarchies [3], design graph grammars [1, 12], resource descriptions [5], case-based reasoning [7, 11], or propositional logic [13, 14].
2. *Model Selection.* Usually model selection does not happen by chance but is a guided process that defines in which way the search is organized. In fact, most of the model formation paradigms mentioned before advise a particular search strategy.
3. *Model Simulation.* Within this step the "behavior" of the selected model is analyzed—a process which can vary largely in its complexity: E. g., when a behavior-based design problem is to be solved, complex differential-algebraic equations need to be processed, while in a simple compositional design problem merely the existence of components is checked [4].

There is a strong research background for solving synthesis problems at our institute, and, in the past, we have developed several specialized configuration solutions for real world tasks [6, 12, 15]. Based on this experience we launched in 2003 the development of an extensible synthesis framework that shall comprise a wide range of technologies related to defining model spaces on the one hand, and, on the other hand, to select, search, and simulate models from a given model space.

Note that we are not aiming at a *generic* synthesis engine since we know that many synthesis problems require specialized and tailored solutions. Instead, we take existing configuration and synthesis technology and put much emphasis on software engineering aspects: extendibility, transparent coupling of technologies, state-of-the-art interfaces, or Web-based access.

Related Work

There has been considerable research effort related to configuration and design in the last 20 years. The starting point for developing tailored software techniques to tackle synthesis problems was the R1 system, which emerged from a joint project between CMU and DEC in the early eighties [8]. R1 was no generic platform but specifically designed to realize the configuration of Vax computers.

The PLAKON system developed from the TEX-K project (1986 - 1990) whose objective was the creation of a knowledge-based kernel for planning and configuration tasks, independent from a concrete domain. Configuration technology in PLAKON was centered around the skeletal configuration paradigm. Aside from powerful compositional and taxonomic descriptions PLAKON enabled the modeling of basic functional and associative constraints as well as the specification of control knowledge to guide the search.

Based on the experiences gathered in TEX-K the successor of PLAKON—the system KONWERK was created. Among others, KONWERK extended the modeling capabilities of PLAKON and came along with a clear modular architecture. Both PLAKON and KONWERK were implemented in Lisp.

The system ENCON can be considered as a partial relaunch of the KONWERK platform, having a strong emphasis on commercial requirements [2]. ENCON is implemented in Java, and, since its configuration kernel is encapsulated by an abstraction layer, the tool is qualified to bring configuration technology to the Web.

2 The ESF Synthesis Framework

The configuration and design projects that we conducted in the last fifteen years came from very different domains, such as logistic systems, telecommunication systems, fluidic engineering, computer networks, testing equipments, or software systems. Our endeavors to find adequate solutions taught us several lessons, among others the following:

- Finding the "right" level of abstraction when defining the model space $\mathcal{M}$ is both the most crucial and difficult job.
- Given a real-world configuration task, then no modeling paradigm will do it all, say, modeling concepts have do be adapted, modified, or even created.
- No configuration strategy can provide that degree of flexibility that it leads to an acceptable search performance for all projects mentioned above.

Nevertheless, every configuration task aims at the construction of a system (to be precise: model of a system) whose function fulfills a set of given demands D. Function, in turn, can be regarded as a consequence of the interplay between structure and behavior of a system [17]. Put another way, a configuration tool must provide concepts for modeling and processing both structure and behavior.[2]

These considerations form the ground for our implementation efforts that flow into the extensible synthesis framework ESF (cf. Figure 2). It provides modeling paradigms and processing methods related to the following three knowledge sources:

(a) *Strategy Knowledge.* Design or configuration strategies can be easily defined by means of a scripting language. Strategies are objects that can be created, stored, combined, triggered, or modified.

(c) *Component Knowledge.* This knowledge source relates to the definition of component classes and instances. Component behavior can be described at different levels of granularity, which includes resource descriptions or simple functional constraints, and will allow complex behavior descriptions based on the Modelica™ language in the near future [10].

(c) *Structural Knowledge.* Instead allowing only compositional and taxonomic hierarchies we use the rather new concept of design graph grammars [12, 15]. Design graph grammars have been developed as a powerful means to manipulate arbitrary structure models of technical systems.

ESF provides several algorithms for manipulating the aforementioned object types. In particular, objects can be shared among different algorithms using a blackboard; the blackboard is also used to store a certain state of the search or to remember alternative choice points among which a strategy may chooses the most promising one. New

[2] PLAKON, for instance, addressed the question of structure with concepts for modeling skeletal plans; likewise, it addressed the question of behavior by a constraint processing mechanism.

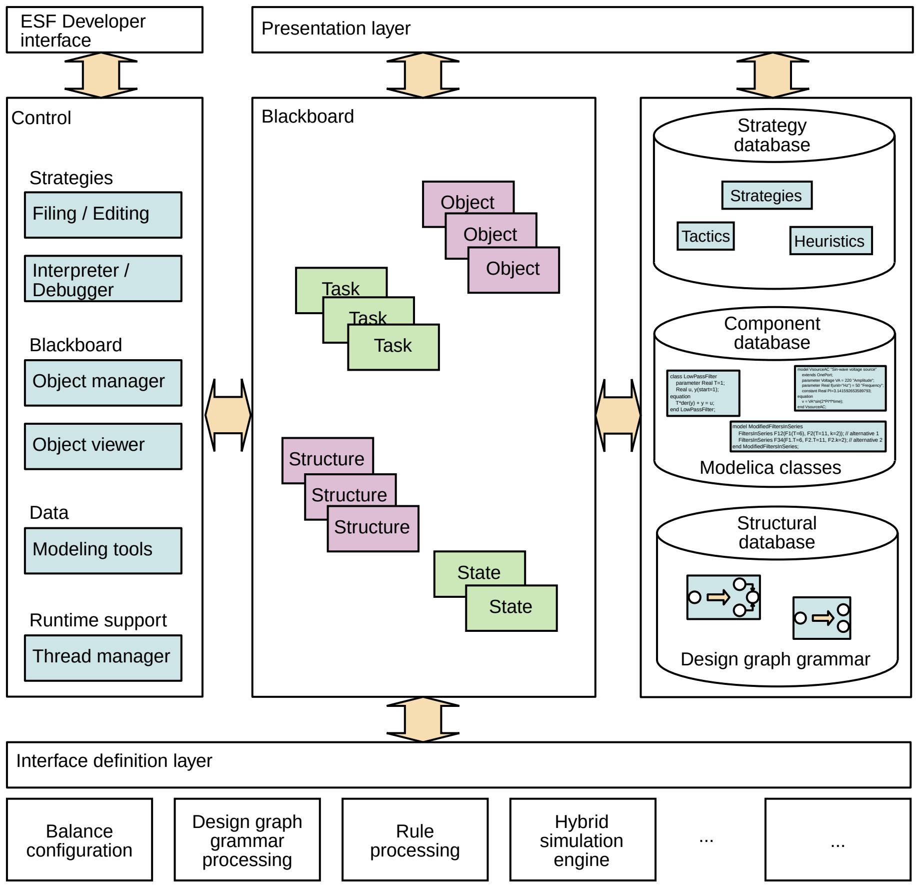

Figure 2. Conceptual view onto ESF. A blackboard serves as central communication turntable (middle); the three different knowledge sources are shown on the right-hand side, the respective algorithms are shown below.

modeling paradigms or component types can integrated straightforwardly: An interface definition layer prescribes a set of accessors and manipulation functions that each object must implement to be shareable via the blackboard.

ESF is being implemented in Java. Note that there are various interesting software-technical aspects that deserve an in-depth discussion; they will be described in a forth-coming white paper.

3 Application: Task-Oriented Plant Layout

This section outlines a particular job-shop manufacturing task that has been realized with ESF. The task combines several aspects of planning, configuration, and simulation: For the production of small metal parts such as screws or gear-wheels the necessary machines have to be selected, a suited processing sequence has to be found, and, a layout of the manufacturing plant has to be generated (see Figure 3).

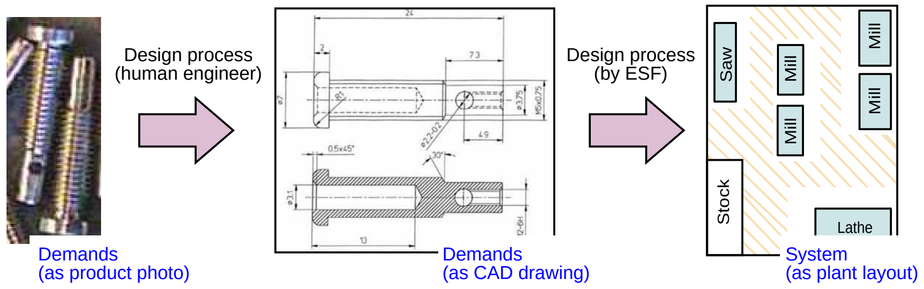

Figure 3. The figure shows the desired product (left), a technical drawing from which the manufacturing constraints are derived (middle), and the plant layout (right), which is part of the solution of the design process.

The solution of this design task happens in a cycle of the following steps:

1. Creation of a process flow graph by means of a design graph grammar.
2. Selection and parameterization of the machines used in the process flow graph.
3. Simulation of the work-flow. In case of a positive evaluation, the solution is compared to alternative realizations already derived.

The machines that are used to realize the production process provide certain functionalities: sawing, drilling, milling, nibbling, or turning. Their capacities and performances, as well as their constraints are modeled by means of the resource-based paradigm. For the simulation of the process flow along with the configured machines tailored algorithms were implemented and plugged into ESF.

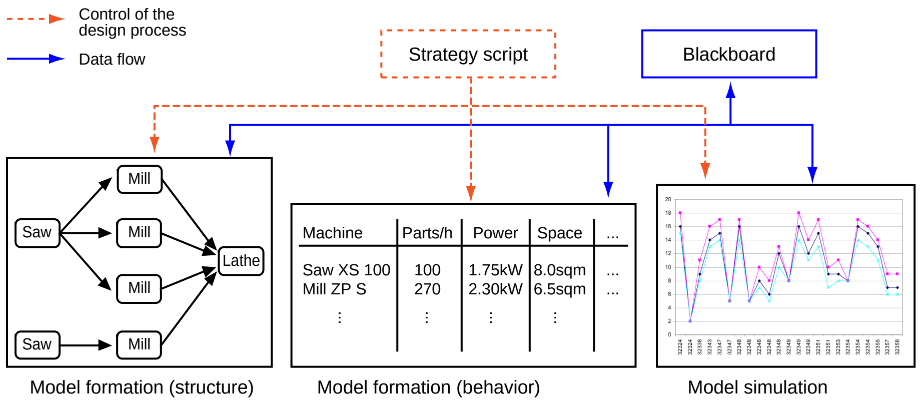

Figure 4. The employed modeling paradigms: Design graph grammars for plan generation (left) and resource-based configuration for material properties, production constraints, and space requirements (middle). The simulation (right) is necessary to determine an optimum solution.

A detailed description of the system, the design graph grammar rules, and the resource model can be found in [18].

References

1. Erik K. Antonsson and Jonathan Cagan. *Formal Engineering Design Synthesis*. Cambridge University Press, 2001. ISBN 0-521-79247-9.
2. V. Arlt, A. Günter, O. Hollmann, and T. Wagner. Engineering & Configuration—A Knowledge-Based Software Tool for Complex Configuration Tasks. In *Proceedings of the AAAI 99 Workshop on Configuration*, 1999.
3. R. Cunis, A. Günter, I. Syska, H. Peters, and H. Bode. PLAKON—An Approach to Domain-independent Construction. Technical Report 21, BMFT Verbundsprojekt, Universität Hamburg, FB Informatik, March 1989.
4. John S. Gero. Design Prototypes: A Knowledge Representation Scheme for Design. *AI Magazine*, 11:26–36, 1990.
5. M. Heinrich and E. W. Jüngst. A Resource-based Paradigm for the Configuring of Technical Systems for Modular Components. In *Proc. CAIA '91*, pages 257–264, 1991.
6. Marcus Hoffmann. *Zur Automatisierung des Designprozesses fluidischer Systeme*. Dissertation, University of Paderborn, Department of Mathematics and Computer Science, 1999.
7. Mary Lou Maher and Pearl Pu, editors. *Issues and Applications of Case-Based Reasoning in Design*. Lawrence Erlbaum Associates, Mahwah, New Jersey, 1997.
8. John McDermott. R1: A Rule-based Configurer of Computer Systems. *Artificial Intelligence*, 19:39–88, 1982.
9. Marvin Minsky. Models, Minds, Machines. In *Proceedings of the IFIP Congress*, pages 45–49, 1965.
10. Modelica Association. *Modelica™—A Unified Object-Oriented Language for Physical Systems Modeling: Tutorial*. Modelica Association, Linköping, Sweden, 2000.
11. Michael M. Richter. Introduction to CBR. In Mario Lenz, Brigitte Bartsch-Spörl, Hans-Dieter Burkhard, and Stefan Weß, editors, *Case-Based Reasoning Technology. From Foundations to Applications*, Lecture Notes in Artificial Intelligence 1400, pages 1–15. Berlin: Springer-Verlag, 1998.
12. André Schulz. *On the Automatic Design of Technical Systems*. Dissertation, University of Paderborn, Department of Mathematics and Computer Science, 2001.
13. David B. Searls and Lewis M. Norton. Logic-based Configuration with a Semantic Network. *Journal of Logic Programming*, 8:53–73, 1990.
14. Benno Stein. *Functional Models in Configuration Systems*. Dissertation, University of Paderborn, Institute of Computer Science, 1995.
15. Benno Stein. *Model Construction in Analysis and Synthesis Tasks*. Habilitation, University of Paderborn, Institute of Computer Science, June 2001.
16. Benno Stein. Engineers Don't Search. In Wolfgang Lenski, editor, *Proceedings of a Symposium on Logic versus Approximation, Schloss Dagstuhl, Germany*, volume LNAI of *Lecture Notes in Artificial Intelligence*, Berlin Heidelberg New York, October 2003. Springer.
17. Nam Oyo Suh. *Axiomatic Design*. Oxford University Press, 2001. ISBN 0-19-513466-4.
18. Achim Wullenkort. Eine Entwicklungsumgebung für Design-Aufgaben. Diploma thesis, University of Paderborn, Institute of Computer Science, 2004.