Design Problem Solving by Functional Abstraction

Benno Stein

Paderborn University
Dept. of Computer Science
D-33095 Paderborn, stein@upb.de

Abstract The paper introduces the Functional Abstraction paradigm as a concept to solve complex design problems.

Design problem solving can be considered as selecting an appropriate model from a space of possible models $\mathcal{M}$. By Functional Abstraction a simplified view $\widehat{\mathcal{M}}$, $\widehat{\mathcal{M}} \ll \mathcal{M}$, onto the design space $\mathcal{M}$ is constructed. I. e., Functional Abstraction follows a model simplification strategy in order to make a synthesis problem tractable.

Aside from a description of the paradigm the paper outlines two design problems where the paradigm of Functional Abstraction has been applied.

1 Introduction

In order to make this paper self-contained to a certain degree, the introductory section presents the definitions that are used later on.

1.1 Systems and Models

A system, S, can be considered as a clipping of the world and has, as its salient property, a boundary. On account of this boundary, it can be stated for each object of the world whether or not it is part of S.[1] Models are the essential element within the human ability to reason about systems. Numerous definitions, explanations, and introductions have been formulated about the term model—my favorite definition stems from Minsky:

> *"To an observer B, an object A^* is a model of an object A to the extent that B can use A^* to answer questions that interest him about A."*
>
> Minsky, 1965, pg. 45

Here,

- the interesting objects, A, are technical systems,
- the observer, B, is a domain expert who works on a problem solving task from the field synthesis, such as a configuration or design task,
- the questions are embedded in a ternary way; they relate to the technical system, to the problem solving task, and to the domain expert,
- the models, A^*, are exactly that what Minsky has pointed out above.

[1] This characterization of the term system is derived from Gaines's definition [5], which, like other definitions, can be found in the well written introduction of Cellier's book on continuous system modeling [2].

1.2 Synthesis Tasks and Model Spaces

Our starting point when solving a synthesis task is characterized as follows. We are given a set of systems, $\mathcal{S}$, called the system space, along with an open question D. The question D may ask whether a system with a desired functionality (= demands) exists, say, is member of $\mathcal{S}$, or how much a system with a desired functionality at least costs. Answers to such questions can be found by constructing the desired system and analyzing its functionality.

Clearly, constructing a system solely for answering an open question cannot be accepted in the very most cases. A way out is the creation of a bijective function, φ, that maps each system $S \in \mathcal{S}$ onto a model M in a *model space* $\mathcal{M}$ (see Figure 1). Usually the model space is described intensionally, by means of a finite number of combinable objects along with operations that prescribe how objects can be connected to each other.

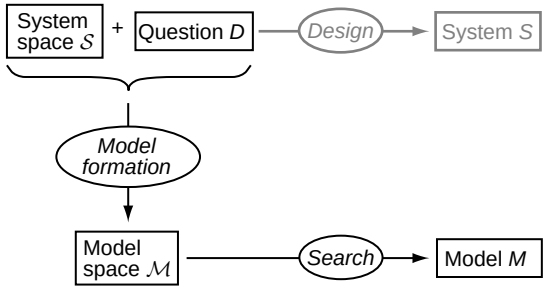

Figure 1: Synthesis problems are solved by means of a systematic search in the model space—provided that a bijective mapping from the system space onto the model space can be stated.

A computer program for automated design or configuration realizes a function $\Phi : \mathcal{D} \longrightarrow \mathcal{M}$ that maps a set of demands D onto a model M, $\Phi(D) \mapsto M$. Put in a nutshell, synthesis problems are solved by developing a systematic search strategy that turns a model space into a search space [12].

1.3 Structure Models and Behavior Models

Our view to models and model construction is bivalent: it is oriented at structure and behavior (see Figure 2).

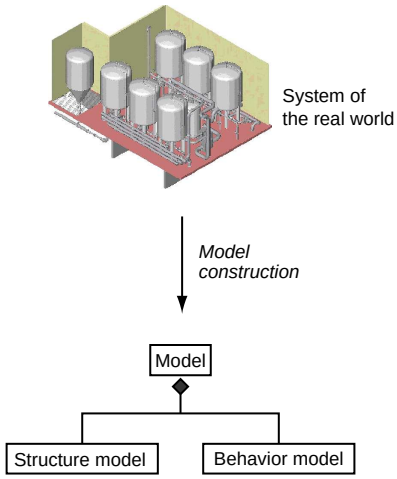

Figure 2: Basically, model construction divides into structural and behavioral considerations.

A structure model renders the structural or topological setup of a system. A behavior model reproduces, in extracts, a system's behavior. In this connection, model formation relates to both structural composition and behavior specification.

Structure models are a powerful means to define a model's composition space—without committing the level of abstraction at which a technical system is described. Note that a structure model says nothing about the models type or purpose, whether it establishes a qualitative model, a dynamic behavior model, or some other model. Typically, an infinite number of behavior models can be associated with the same structure model.[2]

In the following, let M_S define the structure model of some model M in question.

2 Functional Abstraction

The term model simplification speaks for itself: By reducing a model's complexity a problem solving task in question shall become tractable. Functional Abstraction is a model simplification paradigm that aims at a substantial reduction of the number of models that have to be synthesized and analyzed in order to solve a configuration or design problem.

The two case studies sketched out in Section 3 present approaches to solve complex design tasks. Both tasks comprise creative aspects, and for neither a design recipe is at hand: The acquisition effort for the design knowledge exceeds by far the expected pay back [10], and, moreover, the synthesis search spaces are extremely large and scarcely to control—despite the use of knowledge-based techniques.

[2] In this place a precise and formal definition of structure models and behavior models should be given [16]. However, I continue without doing so since (1) most readers may have a more or less clear understanding of these terms, and (2) a formal definition would go beyond the scope of this paper.

Two possibilities to counter this situations are "competence partitioning" and "expert critiquing". The idea of competence partitioning is to separate the creative parts of a design process from the routine jobs, and to provide a high level of automation regarding the latter (see [15, pg. 93]). Expert critiquing, on the other hand, employs expert system technology to assist the human expert rather than to automate a design problem in its entirety [6, 4].

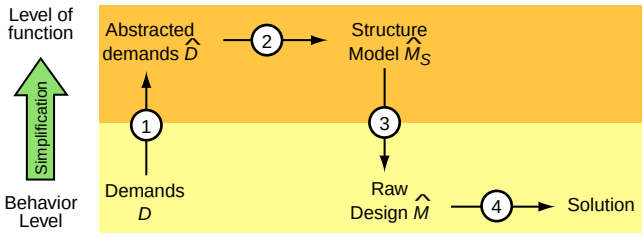

Figure 3: The paradigm of Functional Abstraction in design problem solving.

The paradigm of "Design by Functional Abstraction", illustrated in Figure 3, can be regarded as a special expert critiquing representative. We have chosen this name for the problem solving method to reveal its similarity with the problem solving method HEURISTIC DIAGNOSIS, which became popular as the diagnosis approach underlying MYCIN [3].

> The key idea of Design by Functional Abstraction is a systematic construction of candidate solutions within a very simplified design space $\widehat{\mathcal{M}}$, $\widehat{\mathcal{M}} \ll \mathcal{M}$, which typically is some structure model space: The structure model of a solution candidate, $\widehat{M}_S \in \widehat{\mathcal{M}}$, is transformed into a preliminary raw design, $\widehat{M}$, by *locally* attaching behavior model parts to $\widehat{M}_S$. The hope is that $\widehat{M}$ can be repaired with reasonable effort, yielding an acceptable solution for D.

Design by Functional Abstraction makes heuristic simplifications at least at two places: The original demand specification, D, is simplified towards a functional specification $\widehat{D}$ (Step 1 in Figure 4), and, $\widehat{M}_S$ is transformed locally into $\widehat{M}$ (Step 3 in Figure 4). Both, the synthesis step and the adaptation step may be operationalized with complete algorithms (Step 2 and 4 in the figure).

Putting it overstated the paradigm says: At first, we construct a poor solution of a design problem, which then must be repaired.

The solutions in the following case studies were developed after this paradigm.

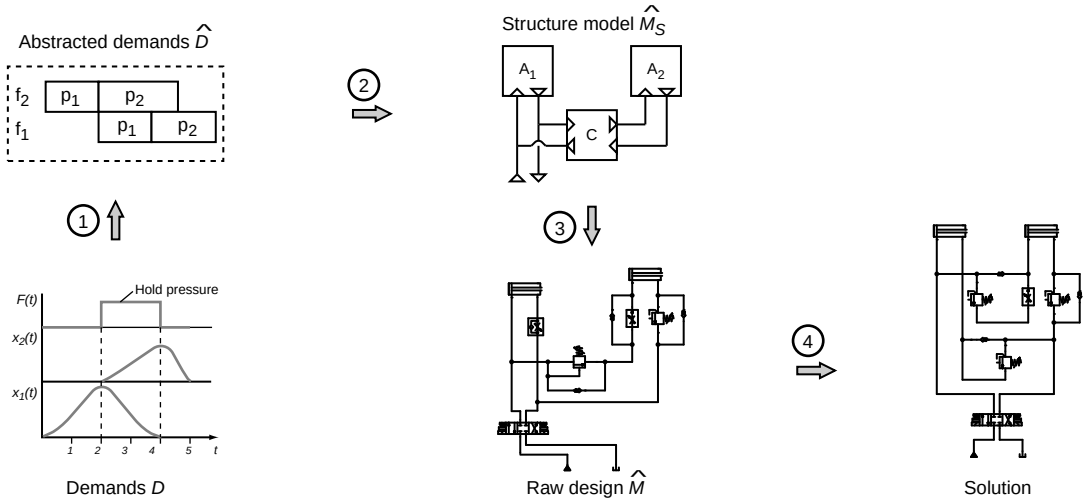

Figure 4: The Functional Abstraction paradigm applied to fluidic circuit design.

3 Case Studies

This section introduces two design problems where the paradigm of Functional Abstraction has been applied. The background of the respective domains, the related constraints, and an in-depth description of the solutions are given in [16]. Here we content ourselves with a short introduction to the ideas.

3.1 Case-Based Design in Fluidics

Fluidic drives are used to realize a variety of production and manipulation tasks. Even for an experienced engineer, the design of a fluidic system is a complex and time-consuming task, that, at the moment, cannot be automated completely. Designing a system means to transform demands, D, towards an explicit system description, which is a behavior model of the desired system in most cases.

Taken the view of configuration, the designer of a fluidic system selects, parameterizes, and connects components like pumps, valves, and cylinders such that D is fulfilled by the emerging circuit.[3] Solving a fluidic design problem at the component level is pretty hopeless, and model simplification must be applied to reach tractability. Key idea is to perform a configuration process at the level of functions (instead of components), which in turn requires that fluidic functions possess constructional equivalents that can be treated in a building-block-manner. This requirement is fairly good fulfilled in the fluidic domain—the respective building blocks are called "fluidic axes".

The overall design approach outlined here follows the paradigm depicted in Figure 3 and is illustrated in Figure 4:

(1) The original demand specification, D, is abstracted towards a functional specification $\widehat{D}$.

(2) At this functional level a structure model $\widehat{M}_S$ according to the coupling of the fluidic functions in $\widehat{D}$ is generated.

(3) $\widehat{M}_S$ is completed towards a tentative behavior model $\widehat{M}$ by plugging together locally optimized fluidic axes; in [7] this step is realized by a tailored case-based reasoning approach.

(4) The tentative behavior model $\widehat{M}$ is repaired, adapted, and optimized globally. In this connection scaling rules and heuristic repair rules come into operation.

Remarks. A human designer is capable of working at the component level, *implicitly* creating and combining fluidic axes towards an entire system. His ability to automatically derive function from structure—and vice versa: structure *for* function—allows him to construct a fluidic system without the idea of high-level building blocks in the form of fluidic axes.

Operationalization The concepts have been embedded within a design assistant[4] and linked to FLUIDSIM, a drawing and simulation environment for fluidic systems [17]. The design assistant enables a user to formulate his design requirements D by specifying both a set of fluidic functions and a coupling hierarchy. For each fluidic functions a sequence of phases can be defined, where for each phase a set of characteristic parameters, such as duration, precision, or maximum values can be stated.

Clearly, a direct evaluation of generated design solutions must be limited within several respects since

(1) an absolute measure that captures the quality of a design does not exist, and

(2) the number of properties that characterizes a design is large and their quantification often requires a high effort.

[3] The ideas presented in section have been verified in the hydraulic domain in first place; however, they can be applied in the pneumatic domain in a similar way, suggesting us to use preferably the more generic word "fluidic".

[4] The design assistant has been realized and evaluated as a part of the doctoral thesis of Hoffmann [7].

# Axes	Reuse, repair	Correct (at $\sigma > 0.9$)	Quality		
1	0.10s	80%	60% (+)	35% (o)	5% (–)
2	0.63s	75%	50% (+)	45% (o)	5% (–)
3	0.91s	70%	40% (+)	50% (o)	10% (–)
4	1.43s	60%	20% (+)	65% (o)	15% (–)
5	2.00s	20%	5% (+)	80% (o)	15% (–)

Table 1: Runtime results and modification effort for automatically generated designs. Test environment was a Pentium II system at 450 MHz with 128 MB main memory.

However, the quality of a generated design can also be rated *indirectly*, by quantifying its "distance" to a design solution created by a human expert. In this connection, the term "distance" stands for the real modification effort that is necessary to transform the computer solution into the human solution. The experimental results presented in Table 1 describe such a competition; a more detailed discussion of evaluation concepts as well as related problems can be found in [7]. Description of the table columns:

- *Axes number.* Number of axes of each test set; a test set contains 20 queries.
- *Reuse, Repair.* Average time of the reuse and repair effort in seconds.
- *Correct.* Number of generated designs with a similarity ≥ 0.9.
- *Quality.* Evaluation of a human expert. In this connection a (+), an (o), and a (–) designate a small, an acceptable, and a large modification effort necessary to transform the machine solution into a solution accepted by the human expert.

3.2 Design in Chemical Engineering[5]

A chemical plant can be viewed as a graph where the nodes represent the devices, or unit-operations, while the edges correspond to the pipes responsible for the material flow. Typical unit-operations are mixing (homogenization, emulsification, suspension, aeration, etc.), heat transfer, and flow transport. The task of designing a chemical plant is defined by the given demands D in the form of properties of various input substances, along with the desired output substance. The goal is to mix or to transform the input substances in such a way that the resulting output substance meets the imposed requirements.

The design happens by passing through (and possibly repeating) the following five steps: Preliminary examination of the demands, choice of unit-operations, structure definition, component configuration, and optimization. An automation of the steps at a behavioral level would be very expensive—if possible at all. Present systems limit design support to isolated subjobs; such systems relieve the human designer from special simulation or configuration tasks, and the effort involved there is high enough [1, 8].

However, the Functional Abstraction paradigm can be applied to develop an overall design approach (cf. Figure 5):

(1) The properties of the input and output substances, D, are abstracted towards linguistic variables, $\widehat{D}$.

(2) At the functional level a structure model $\widehat{M_S}$ is synthesized that fulfills $\widehat{D}$ and that is used as a solution candidate for D; this step is realized by so-called design graph grammars [14].

(3) $\widehat{M_S}$ is completed towards a behavior model $\widehat{M}$.

(4) $\widehat{M}$ is repaired, adapted, or improved.

Both Step 2 and Step 4 rely on a knowledge base with domain-specifc and task-specific design rules. These rules have been formulated as graph grammar rules by a domain expert.

Operationalization The concepts described above are implemented within a prototypical tool, called DIMOD (domain independent modeler). Figure 6 shows a screen snapshot. The core of the system consists of a generic graph grammar engine, used for modeling and application of knowledge, and a domain-specific module used to guide the search process.

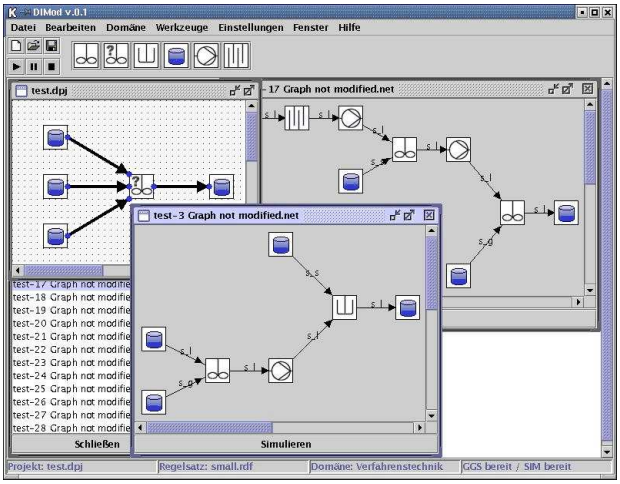

Figure 6: The DIMOD system. Upper left window represents the abstracted demands, the windows to the right and center show two generated structure models.

Within DIMOD the simulation is realized as follows. A generated structure model is completed towards a tentative behavior model by attaching behavior model descriptions to the components of the structure model (Step 3 in Figure 5). The behavior model is then validated by a simulation. For this purpose, the ASCEND IV simulator is used [13]; the attached model descriptions stem from the ASCEND IV model library and from custom models.

[5]This work developed from a cooperative project with the Chemical Engineering Group at Paderborn University that focused on the design of plants for the food processing industry [9].

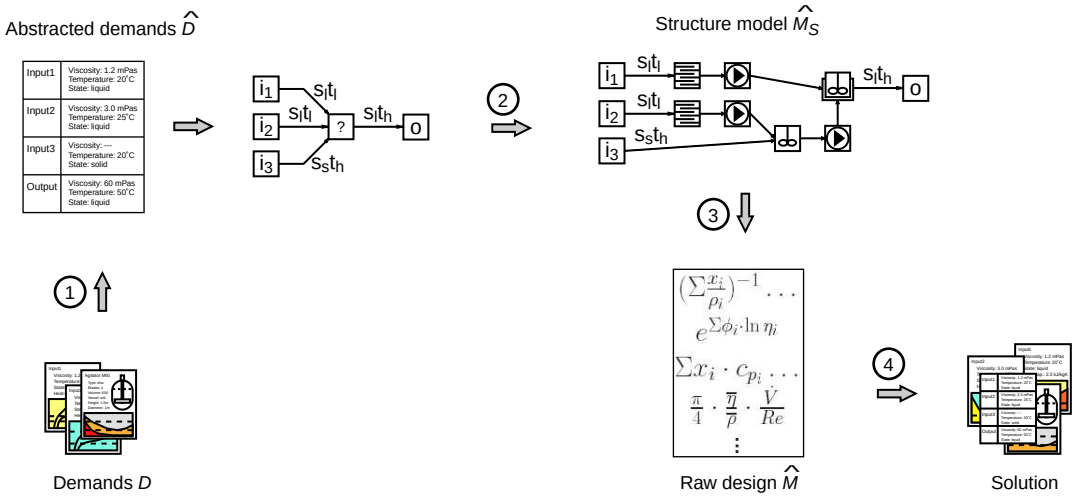

Figure 5: The Functional Abstraction paradigm applied to the design of food processing plants.

References

[1] Axel Brinkop, Norbert Laudwein, and Rüdiger Maassen. Routine Design for Mechanical Engineering. In *Proceedings of the Sixth Annual Conference on Innovative Applications of AI (IAAI 94)*, Seattle, August 1994.

[2] François E. Cellier. *Continuous System Simulation*. Springer, Berlin Heidelberg New York, 1991.

[3] William J. Clancey. Heuristic Classification. *Artificial Intelligence*, 27:289–350, 1985.

[4] Gerhard Fischer, Kumiyo Nakakoji, Jonathan Ostwald, and Gerry Stahl. Embedding Critics in Design Environments. *The Knowledge Engineering Review*, 8(4):285–307, December 1993.

[5] Brian Gaines. General Systems Research: Quo Vadis. In *General Systems Yearbook*, volume 24, pages 1–9, 1994.

[6] Sture Hägglund. Introducing Expert Critiquing Systems. *The Knowledge Engineering Review*, 8(4):281–284, December 1993.

[7] Marcus Hoffmann. *Zur Automatisierung des Designprozesses fluidischer Systeme*. Dissertation, Department of Mathematics and Computer Science, University of Paderborn, Germany, 1999.

[8] Achim Knoch and Michael Bottlinger. Expertensysteme in der Verfahrenstechnik – Konfiguration von Rührapparaten. *Chem.-Ing.-Tech.*, 65(7):802–809, 1993.

[9] Annett Kurzok, Manfred H. Pahl, and André Schulz. Software zur wissensbasierten Prozeßmodellierung – WIP. *Chemie Ingenieur Technik*, 73(9), 2001.

[10] Mary Lou Maher and Andres Gomez de Silva Garza. The Adaptation of Structural System Designs Using Genetic Algorithms. In *Proceedings of the International Conference on Information Technology in Civil and Structural Engineering Design: Taking Stock and Future Directions*, Glasgow, Scotland, August 1996.

[11] Marvin Minsky. Models, Minds, Machines. In *Proceedings of the IFIP Congress*, pages 45–49, 1965.

[12] Judea Pearl. *Heuristics*. Addison-Wesley, Massachusetts, 1984.

[13] P. C. Piela, T. G. Epperly, K. M. Westerberg, and A. W. Westerberg. ASCEND: An Object-Oriented Computer Environment for Modeling and Analysis: The Modeling Language. *Computers Chemical Engineering*, 15(1):53–72, 1991.

[14] André Schulz and Benno Stein. On Automated Design of Technical Systems. Notes in Computer Science tr-ri-00-218, Department of Mathematics and Computer Science, University of Paderborn, Germany, December 2000.

[15] Benno Stein. *Functional Models in Configuration Systems*. Dissertation, Department of Mathematics and Computer Science, University of Paderborn, Germany, June 1995.

[16] Benno Stein. *Model Construction in Analysis and Synthesis Tasks*. Habilitation thesis, University of Paderborn, Department of Mathematics and Computer Science, 2001.

[17] Benno Stein, Daniel Curatolo, and Marcus Hoffmann. Simulation in FLUIDSIM. In Helena Szczerbicka, editor, *Workshop on Simulation in Knowledge-Based Systems (SIWIS 98)*, number 61 in ASIM Notes, Bremen, Germany, April 1998. Technical committee 4.5 ASIM of the GI.