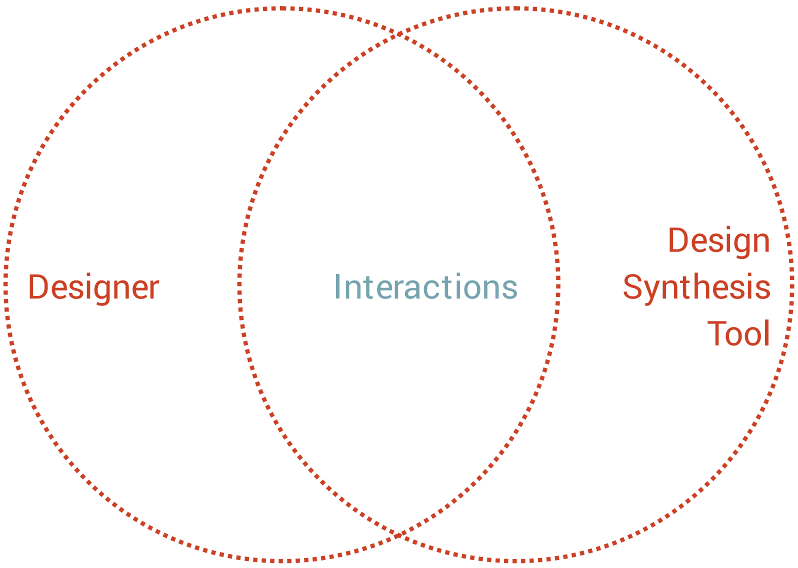
Implications of designer behavior for design synthesis tool interaction design



Maurice van Leeuwen

09-11-2015

Bachelor Industrieel Ontwerpen, Universiteit Twente

Implications of designer behavior for design synthesis tool interaction design

Maurice van Leeuwen
s0180858

Deze scriptie is bedoeld ter afsluiting van het afstudeer project van de bachelor opleiding Industrieel Ontwerpen te Universiteit Twente.

De beoordelingscommissie bestaat uit:
dr. J.M. Jauregui Becker (begeleider en examinator)
dr. ir. G.M. Bonnema (tweede examinator)

Beoordeling vindt plaats in November 2015.

Summary

Design synthesis tools offer considerable oportunities for improving current design practice. Not only can it automatise design tasks, improving pace and efficiency of the design cycle; it is also a means by which design knowledge can be formalized, making it easier to exchange, discuss, understand, and improve the design knowledge. The current state of research and use of design synthesis tools is limited. Industry does not pay much attention to the developments of these tools, with lack of traction as a result for research in this field. This research asserts that usability concerns of design synthesis tools are an important factor for not adopting them: designers, simply enough, are not able to use the tools. The way designers act and understand in design situations clashes with the way design synthesis tools work. We argue design synthesis tools are very capable of providing explanations and insight into design problems.

The research uses a scenario-based development approach to this human-computer interaction problem. It specifically researches current smart synthesis tools developed at the University of Twente. Usability problems found include: designers finding it hard to interpret synthesis results, designers making unwanted simplifications or adaptations of the design task, designers finding it hard to use a suitable synthesis method for the task at hand, designers resorting to alternative workflows. Identified causes for bad usability include: insufficient functionality in solution viewing, unsufficient expressiveness and organizability of design synthesis methods, inability to formulate the design problem and relate design synthesis methods to it.

A proposal is made for the functionality, architecture and use of a design synthesis tool. This proposal shows how the found usability may be improved on. Composing *instruments* aims to improve how a designer constructs and deploys design synthesis methods, *orchestration* is a means to relate design synthesis methods to actual design problems, a *query interface* should allow designers to ask interesting questions about synthesis results.

Functionality of the proposal is related to the identified usability problems, and compared to the smart synthesis tools. This suggests that the proposal indeed allows designers to better use design synthesis tools. Further research into the query language, interface to be used by design synthesis methods, graphical user-interface, can lead to an actual implementation of the proposed tool.

Samenvatting

Design synthesis tools kunnen de huidige ontwerpwijze te verbeteren. Niet alleen is het mogelijk ontwerp taken te automatiseren, met verbetering van de snelheid en efficicientie van het ontwerptraject tot gevolg; het is ook een middel om ontwerpkennis the formaliseren, waardoor het makkelijker is om ontwerpkennis uit te wisselen, bespreken, begrijpen en verbeteren. De huidige staat van onderzoek en gebruik van design synthesis tools is beperkt. Bedrijven geven weinig aandacht aan ontwikkelingen van deze tools, waardoor onderzoek op dit gebied weinig voeten aan aarde krijgt. Dit onderzoek gaat er van uit dat moeizame usability een belangrijke factor is voor het niet in gebruik nemen van design synthese tools: ontwerpers kunnen de tools simpelweg amper gebruiken. De manier waarop ontwerpers denken en handelen tijdens ontwerpen botst met de manier waarop design synthesis tools werken. We beweren dat design synthesis tools echter wél in staat zijn uitleg en inzicht te geven bij de ontwerp problemen waar een ontwerper zich op richt.

Dit onderzoek volgt een aanpak gebaseerd op scenario-based development of human-computer interaction. Er is specifiek onderzoek gedaan naar huidige smart synthesis tools ontwikkeld door Universiteit Twente. Gevonden usability problemen zijn: moeite voor de ontwerper om resultaten te begrijpen, het maken van ongewensde aanpassingen aan de ontwerp taak, het lastig vinden om geschikte synthesis methoden te gebruiken voor de ontwerp taak, het uitwijken naar alternative werkwijzen. Gevonden oorzaken voor deze slechte usability zijn: onvoldoende mogelijkheden in het bekijken van resultaten, matige uitingskracht en organiseerbaarheid in het bouwen van design synthesis methoden, onmogelijkheid om het ontwerp vraagstuk te formuleren en relateren aan design synthesis methoden.

Een voorstel omschrijft de functionaliteit, architectuur en het gebruik van een design synthesis tool. Het laat zien hoe de beschreven usability te verbeteren is. Het samenstellen van *instrumenten* verbetert hoe het maken en inzetten van design synthesis methoden, *orkestratie* is een manier om design synthesis methoden toe te wijzen aan ontwerp problemen, een *query interface* stelt ontwerpers instaat interessante vragen te stellen over synthesis resultaten.

Functionaliteit van het voorstel is gerelateerd aan de gevonden usability problemen en vergelijken met de smart synthesis tools. Dit laat zien dat het voorstel inderdaad ontwerpers de mogelijkheid geeft design synthesis tools beter toe te passen. Verder onderzoek naar de query taal, interfaces, GUI, kan leiden tot een daadwerkelijke implementatie van de voorgestelde tool.

Introduction

Designers encounter increasingly complex design situations. Issues of sustainability, usability, smart devices, make product design a difficult problem. In order to deliver better products in a shorter development cycle, designers reach out for technologies and methods that lead to more effective problem solving capabilities. Design synthesis tools can be such a technology, however it turns out that adoption of such tools by the industry is limited.

It is remarkable that design synthesis tools have not seen the same adoption as other CAD technologies. It is not hard to imagine the benefits designers can have from using a tool that automates synthesis tasks. Not only may automation speed up the design task; but also deliver better results more consistently—effects often seen in other fields. And there are other arguments in favor of design synthesis tools as well: the design synthesis methods they use make design knowledge explicit; knowledge that is valuable to be reused or that can be improved upon. From that perspective design synthesis tools not only provide automation, but also explanation. It is exactly that part, *explanation*, that we argue current design synthesis tools fail to provide designers enough insight in.

1.2 Research goals and approach

This research takes the position that poor understanding of factors in human-computer interaction that play a role in design synthesis tools is a cause for the lack of industry adaption. In order to improve usability, this research aims to:

1. Provide an analysis based on a literature research of how designers think and behave in design situations.
2. Identify the usability issues this causes current smart synthesis tools.
3. Propose solutions for a design synthesis tool that adress these usability concerns.

This research follows an approach to usability engineering as put forward in (Rosson, 2002). This approach is titled *"scenario-based development of human-computer interaction"*, and is well received by usability experts. By applying the approach, software can be made more usable. The primary method used in the approach is to write scenarios, typical situations of a user interacting with the software, and to make claims based on these scenarios. A multiple of different analyses are used to construct the scenarios and describe the behavior of users.

The analyses used in this research to describe the behavior of designers include: literature research; analyses of roles, tasks and artefacts. For every described aspect of the designer's behavior, implications for design synthesis tools are made: what should the tool do or not do to support this kind of behavior. The analyses are used to construct problem scenarios, which lead to claims about problematic features of current smart synthesis tools.

A proposal is made for a design support tool. This proposal is described in terms of its functionality, architecture, and illustrative use cases. These use cases lead to claims about how features of the proposal solve usability problems.

1.3 Methodology

The research was structured into three phases. These phases more or less coincide with a "standard" design cycle, excluding the "implementation" stage: Discovery > Analysis > Design > Evaluation. We will explain the steps in each phase:

Phase 1: Understand the topic

The first phase of the research consisted of getting acquainted with the topic. Literature on computational design synthesis was researched in order to understand the functioning and use of synthesis tools. This lead to discoveries about current issues in the implementation and use of these tools. Literature on design cognition was researched in order to understand how designers think and behave, what situations they encounter, and how they interact with tools. This was used to understand and write relevant and realistic use cases for smart synthesis tools. Current smart synthesis tools were explored in order to gain an understanding of the functions and working of these tools, this was useful for determining what interactions occur between the user and tool in use cases. Conversations with expert Jauregui-Becker uncovered problems in the adoption of smart synthesis tools by industry, which were later related to the found usability problems.

Phase 2: Find usability problems

The second phase of the research consisted of uncovering the usability problems present in smart synthesis tools. To do so textbooks on user-centered design, usability engineering and scenario-based design were consulted in order to devise an appropriate methodology. This led to considering which users for smart synthesis tools exist, what their goals are, and what tasks they perform with the tool. This information formed a necessary bases for the use cases. These use cases were constructed subsequently by investigating how the design tasks can be performed with a smart synthesis tool. The use cases, a description of the interactions between user and tool, led to the discovery of usability problems. The causes and effects of these problems were investigated in order to find areas of improvement.

Phase 3: Improving the interaction design

The third phase of the research aimed to produce improvements to the interaction design of smart synthesis tools. During the course of the research ideas were explored for improving the interaction design; usually in reaction to the problems discovered at that point. This led to a number of user-interface sketches. These sketches were reflected on, to later be discarded. This step contributed to the development of a vision for improving the interaction design. Later a use case was created that describes desirable tool interactions. This contributed to understanding how the interaction design can be improved on, without being specific about an implementation. Consequently a system architecture was described and detailed that can fulfill the interactions as described in the use case. The research was concluded by evaluating what the important usability problems in smart synthesis tools are, and how the proposed interaction design can improve this.

1.4 Problem definition

Current functionality of smart synthesis tools leads to usability issues. These usability issues have effect on:

1. The sentiment / attitude the user has towards smart synthesis tools. It creates a lack of trust in smart synthesis tools.
2. The workflow of the user. The user is disrupted in performing his tasks with the smart synthesis tools and is forced to use alternative tools.
3. The workload of the user. The user needs to perform excise tasks that are disrupting his goal-oriented tasks.

1.5 Smart synthesis tools

Throughout the years, research efforts at the University of Twente brought forth a number of smart synthesis tools. These tools automatize certain design tasks, and are developed for a specific domain and a specific application. New synthesis tools originate from academic efforts, in order to explore and demonstrate the possibilities of new ideas. But the development of smart synthesis tools also happens with the collaboration of industry partners. Promising smart synthesis tools have been developed for applications such as design synthesis in injection molding (Jauregui-Becker 2010), optical systems (Schotborgh et al. 2012), linkage design (Draijer et al. 2002). As a result of the ongoing efforts in design synthesis research there is a growing body of knowledge on this topic at the University of Twente.

Computational design synthesis field

Research at the University of Twente into smart synthesis tools contributes to the field of computational design synthesis. This field consists of research on the methods and tools that can automate design design tasks, and often seeks to formalize this. Recurring issues in this field are the topics of representations, modelling, solver algorithms, software architecture, and the principles of design synthesis. Applications are numerous in various domains, such as the design of electronics, mechanics, product design, architecture, supply-chain, and in manufacturing. (Antonsson, Cagan 2001)

Approach used in Smart Synthesis Tools

Current smart synthesis tools aim to propose solutions for routine design problems. These are design problems for which there is complete knowledge on the functions, behaviors and structures involved. It aims to find suitable design parameters for the structures. Innovative design concerns itself with finding new structures that satisfy certain behaviors, creative design concerns itself with finding which behavior satisfies certain functions. The current view is that innovative and creative design involves routine design. Research in smart synthesis tools for routine design, will eventually lead to solutions for innovative and creative design. (Jauregui 2010) Smart synthesis tools find solutions by using the knowledge available in a routine design problem. FBS representations and DPU structures of the design problem contain enough information for the smart synthesis tool to operate.

Function-behavior-structure representations

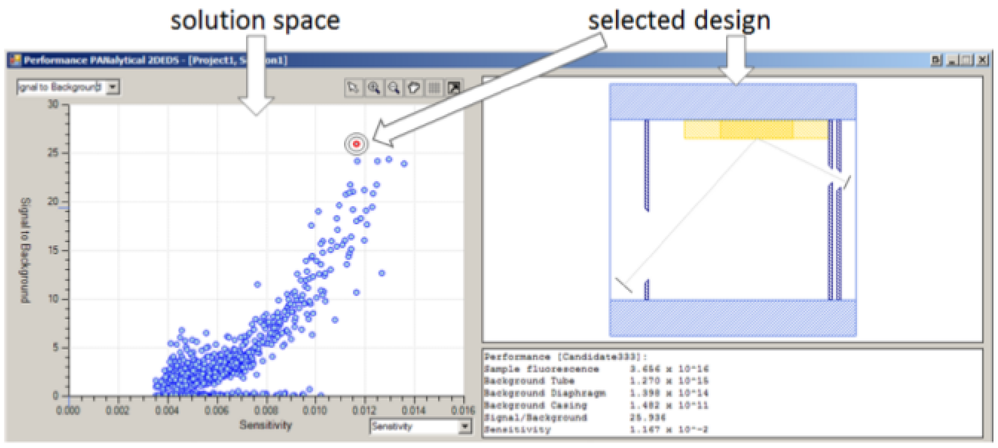
When design engineers reason about a system they design, they may do so in terms of its function, behavior and structure. This way of representing a design problem is described by a number of authors, such as in (Gero 2004). The FBS model plays a role in design synthesis because, by Gero's definition, synthesis is the translation of expected behavior into a structure that should exhibit these behaviors. Analysis derives the actual behavior of the designed structure. While describing a system under design using these terms can be useful, it is not necessarily without problems. Discussions about what definition should be used for "function" or "behavior" are recent and unsettled (Dorst, Vermaas 2005), (Chandrasekaran 2005), (Deng 2002), (Hirtz 2002). This does not mean that FBS is not useful in smart synthesis tools; many of the existing views on FBS can be of practical use. In current smart synthesis tools, before solutions are created, the user describes requirements and constraints for variables in involved design. Performance and scenario variables describe how the system reacts to external stimuli: behavior. Design variables describe the structure of the system. The synthesis tool uses this knowledge to create proposals on a structural level; while analysis performed by the tool is used to report information to the user on a behavioral level. Therefore the user is concerned with both the structural and behavioral level of the system.

DPU model

Another approach that is relevant in the current smart synthesis tools is DPU modelling (Jauregui-Becker, Wits 2012). This is an approach for describing the behavior of a (sub)system, so that it can be used for simulation or analysis. This is done by collecting and structuring information used in designing a system, resulting in a knowledge structure referred to as a design process unit. The behavior of complex parts or systems can be described by combining various DPU building blocks. The knowledge in a DPU building block includes declarative knowledge on analysis (what behavior the system has, depending on variables), and variables of the design (embodiment), scenario (context the system operates in), and performance (measured effects). When a DPU is created for a system under design, its knowledge can easily be related to the synthesis of the system, and it is a useful structure in determining design strategy.

Example

The following pictures present examples of the typical user interfaces of smart synthesis tools:



```
Performance [Candidate333]:
Sample fluorescence      3.656 x 10^-16
Background Tube          1.270 x 10^-15
Background Diaphragm     1.398 x 10^-14
Background Casing        1.482 x 10^-11
Signal/Background        25.936
Sensitivity              1.167 x 10^-2
```

Figure 1. An UI of the smart synthesis tool in (Schotborgh et al. 2012)

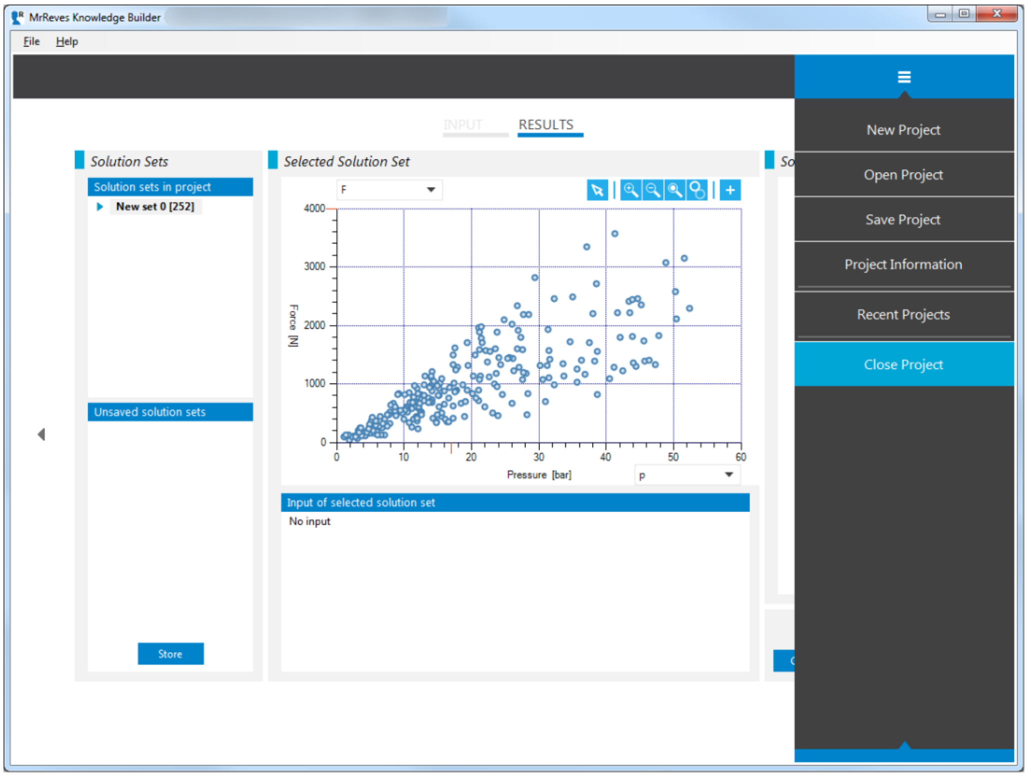


Figure 2. An UI of the MrReves tool developed by Reden.

Literature review

2.1 Design synthesis

This section discusses a few results found in design synthesis research that provide insights in the usability problems found in these tools.

One approach for design synthesis is to use a compiler. A compiler is a program that transforms source code from one language into code in a target language. In computer software this is primarily used to translate code written in high-level languages such as Java, into executable machine code. For mechanical design at least one compiler has been described, namely by (Ward, 1989). This compiler works by providing it with a high-level description of the mechanical system the user desires. The compiler transforms this description into a design that is usable by a manufacturer. Ward later argues about what makes compilation hard, and what he thinks are necessary characteristics of a good compiler (Ward, 2001). Ward recognizes that, in many other design compilers, the synthesis method itself is flawed. One reason is that the method is only a part picker. It satisfies user-provided requirements directly, by providing suitable parts, but without using a higher-level language from which these requirements are infered. Another reason is that the synthesis method does not take interactions between components into account, because top-down synthesis is used.

Ward also considers the succes of the commercial product Pro-Engineer from PTC, which was developed in a similar timeframe as a commercial product by ICAD. Pro-engineer introduced parametric, associative feature-based, solid modeling, it was the first software on the market to do this [1]. Pro-Engineer had an advanced rule-based engine, which allows for reasonable design synthesis possibilities. ICAD provided geometric representation for LISP programming, and allowed users to program top-down definitions of mechanical objects, which made it the most advanced tool at that time for mechanical design synthesis. While ICAD had far better functionality for synthesis; the industry adopted Pro-Engineer instead (Ward, 2001). Ward discusses a number of issues with the ICAD software; which are infact usability concerns:

1. There is a big learning curve in using design synthesis software. ICAD uses top-down synthesis and a LISP-like programming language. [2] Training users was a problem.

---

1. www.ptc.com/about/history

2. Maintainance was an issue: reusing existing components in new systems proved to be problematic.
3. Other software (at that time using Pro-Engineer's rule based engine) made it easier to manage interaction effects; namely by direct intervention.

Ward does not investigate how user-interface or environmental factors (e.g. the software ecosystem, or the need for a Symbolics LISP-machine) may have caused these problems. Rather, he seeks the cause in how design synthesis works on a fundamental level. We consider Ward's argument for essential characteristics of a mechanical design compiler:

1. Ward recognizes the need for a high-level language, that will be automaticly transformed in a description that is useable to other users.
2. He argues for working with sets. Elements as described in the high-level language should relate to a set of (possible implementation of) components; analysis equations and procedures should be approriately abstract to be applied to sets of (rather than invidual) components. And constraints should be applied on the set of possible solutions.
3. Component, just as analysis methods, are organized in modules (possibly provided by companies that design components). These modules operate according to an agent-based architecture. This allows for collaboration between modules, instead of applying a top-down transformation. The idea of agents for computational design synthesis was researched by (Campbell et al. 1999), considered as promising for innovative design (Campbell et al. 2003), and is considered by (Chandrasegaran et al. 2012, pg. 217) as an emerging technology that will be important in collaborative design. Chandrasegaran remarks that the requirement for this approach will be the use of an appropriate knowledge representation that is relevant for the designer, problem domain and nature of information.
4. Ward emphasizes the need for a common interfaces that should be used by these modules. Otherise the modules can not interact.
5. Communication between modules is based on predicate calculus and describes sets of design possbilities. This results in having components, communicating as agents, describe amongst themselves the set of desing possibilities they represent.

2. Remarkably, when ICAD first appeared it was for use on Symbolics LISP-machines; just as the most prominent 3D graphics software of the time: S-Graphics. ICAD and S-Graphics share the same faith: to be ported to use on different machines after the bankrupcy of Symbolics, only to disappear after the year 2000.[3]

3. http://web.archive.org/web/20110716211447/http://lemonodor.com/archives/000256.html

In our proposal we take a position that is supportive of Ward's ideas: a high-level language, the use of sets, and an agent-based architecture. A high-level language, to model the design problem at hand, allows for better expression of the problem while reducing the need to describe implementation details. Sets allow users to think in terms of possible alternatives (rather than optimal solutions), which we deem an important use case for design automation software. Agents are a way of organizing design knowledge (logic, rules, procedures), and allocating and dividing design tasks, in a way that is conceptually similar to collaborative design. Organizations, teams, inviduals, and agents: they possess certain design knowledge; they carry out certain design tasks; and by collaboration (sharing of knowledge and results) contribute to the main design task.

Set-based concurrent engineering

Ward argues for a prominent role for sets in design synthesis systems; a position that follows from his earlier research into sets. An example of his earlier research is the co-authorship of an article describing the concurrent engineering methodology used by Toyota Motor Corporation (Sobek et al. 1999). In this article set-based design is presented as an engineering methodology that opposes *point based design*. In point based design a solution is picked quickly, and then the solution is iteratively modified until it meets design objectives. This may happen concurrently, developing multiple aspects of that same (single) solution at a time.

The set-based concurrent engineering method rather considers broad sets of possible solutions; which (by elimination) gradually converge into a final solution. The principles of set-based concurrent engineering are, taken directly from (Sobek et al., 1999, pg. 73) are described in table 2:

Step	Activity
Map the design space	Define feasible regions; Explore trade-offs by designing multiple alternatives; Communicate sets of possibilities;
Integrate by intersection	Look for intersections of feasbile sets; Impose minimum constrant; Seek conceptual robustness;
Establish feasability before commitment	Narrow sets gradually while increasing detail; Stay within sets once committed; Control by managing uncertanty at process gates;

Table 2. Principles of set-based concurrent engineering; adapted from (Sobek et al., 1999, pg. 73)

My commitment to a proposal that is consistent with Ward's view leads us to consider these principles of set-based concurrent engineering. The method as described almost directly leads to interactions we wish to support with our proposal, which we will describe at a later point.

Co-evolution for computational design synthesis and design cognition

An interesting intersection of design synthesis and design cognition research is Maher's co-evolution. The idea was first presented as a computational model for design (Maher, Poon 1996), and later developed into a cognitive model for design.

The computational co-evolution model for design was used to explore broad problem definitions as is often seen in conceptual design. A genetic algorithm is presented that can be used to explore the problem space. Even if this specific algorithm is not used in modern computational design synthesis, Maher's co-evolution as a computational model plays a role in how design synthesis tools operate. This is because:

- The design synthesis tool contains a description of the problem space in the form of a model, variables, constraints, scoring functions (performances).
- Adjustments in the model lead to alterations in the solutions that the tools creates.
- The tool may make alterations to the problem, based on aspects of the solutions. For example: synthesis methods usually add constraints or adjust their range, when seeking solutions.

This view of how design synthesis works computationally doesn't relate to usability per se: it only describes a mechanism applied by the computer. But a usability problem becomes apparent when a user tries to reflect on how the computer found the results. Can the user understand what mechanisms were applied, or is this unclear: is the tool effectively a black box? When a user wants to recover from a mistake, or improve his usage of the tool, it helps if he knows what happened and how it happened. He needs to know the inner workings of the machine. And to push that machine metaphor a bit further: can he repair it in case something breaks? Can he maintain it so that it keeps working properly? Is he able to replace or add parts? When the results of design synthesis are so dependable on how the synthesis tool operates, the tool is only usable if the user can investigate and change it's behavior.

The model of co-evolution can also be applied as a cognitive model and a model for problem solving. Maher later developed her ideas into a cognitive model for design (Maher, Tang 2003). And according to Drost and Cross creative problem solving can be understood in terms of Maher's model of co-evolution (Dorst, Cross 2001). In problem solving there is a constant repetition of analysis, synthesis and evaluation, leading to a constant reformulation of both the problem and solution. This reformulation is considered as co-evolution. Effectively this means that discovery of new knowledge in one domain leads to adaptations in the other domain. It is explained that, in creative events, an observation about the problem leads to a new idea for the solution, and vice-versa. This new idea leads to a reformulation, which is considered as an evolution. This establishes a kind of link. An observation in one domain, together with the reformulation in the other domain, can be considered as a problem-solution pair. The authors provide an explanation for how this pair is formed; how creative events occur. They recognize that in order for a creative event to occur, a creative bridge must be developed. Initially problem and solution spaces cannot be linked, but through exploration it becomes clear what parts of the problem relate to what parts in the solution. The manner in which the problem is interpreted can lead to problem solving, and it also affects what solutions are found: described by Schön as the problem frame.

Creative problem-solving happens as well using design synthesis tools. Drost and Cross's co-evolution view of creativity is relevant because:

- Within the design synthesis tool, the user can formulate his ideas about the problem using logic / a model
- The results of synthesis tool can lead the designer to form new ideas or knowledge about the solution.
- Observations about the results of the synthesis tool, aid the designer to develop new ideas about the problem.
- Therefore it can be understood that design synthesis tools help bridge creative gaps, and form problem frames.

This view of creative problem solving using design synthesis broadly describes interactions with it's user. The associated usability problem is about if the user can effectively share his ideas with the tool, and learn something new. It is about how effective the tool is in bridging the creative gap. This is a matter of expressiveness: can the user share how he understands the problem, and can he do so in a way that makes sense to him? Or restricts the design synthesis tool how the problem is framed? And accessibility plays a role: can the user get the information that leads to new understanding, and is it presented in a way that is useful to him? It cannot be expected that a design synthesis tool does all of the design: it becomes a useful tool if the user can incorporate it in his problem solving.

2.2 Design cognition

The focus of this research is on usability issues of design synthesis tools. Therefore we we consider it essential to understand how users of such software approach design situations. This approach includes many aspects about how a designer deals with design situations cognitively: how he understands design problems, how he relates solutions to these problems, and how he forms a strategy to solve these problems. Aspects of designerly thinking are described, which are later used to find factors in design synthesis software that cause usability problems.

There are two prevailing approaches to design cognition. Herbert A. Simon's symbolic information processing and the reflective approach of Donald Schön that introduced situatedness. Both approaches provide a suitable framework for reasoning about design cognition; but offer a very different focus. Rather than describing aspects of both perspectives on design cognition, we follow the position put forward in (Visser, 2004). Visser's stance is nuanced: she posits both approaches can be integrated and provides suitable argumentation for this. She presents a cognitive model of design that focusses primarily on the the cognitive processes and strategies a designer uses. The following table sums up the most important of these cognitive aspects in problem solving; both in the philosophy of Simon's and Schön's approaches.

Cognitive aspect	**Description**
Creative design	Decisions to a design problem are not known at the beginning of solving it. There is no way the problem can be broken into tasks or predetermined hierarchies, before it is solved. (Navinchandra, 1991)
Semantic richness	In design a lot of specific domain knowledge is used. This knowledge is used to construct problem representations. Domain-specific problem solving methods are used in addition to general methods. (Bhaskar and Simon, 1977)
Satisficing	This idea, put forward by Simon, states that design problems allow a satisfactory solution. This means that there no single correct solution exists, but actually many. Because a design problem can be interpreted and solved in various directions, optimising is impossible: it is only possible to find a solution that fits the problem description.
Atomization	Identifying the parts of which a design consists is the most crucial part of design. (Mayer, 1989)
Opportunistic organization of design activity	Designers often use methods from other domains and they use common knowledge even if many existing available for the design task they face. Also, the problem-solving strategies designers use are not known beforehand.

Table 3. Description of cognitive aspects in problem solving

Cognitive aspect	Description
Ambiguity	Aspects of a design problem, and possible solutions and improvements, are interpreted differently by different people. (Reitman, 1964)
Problem simplication	Designers reduce the complexity of a design problem by decomposing it. Even while this doesn't promise that subproblems can be solved without considering the other subproblems.
Spatio-temporal planning	When designers face a spatial form of a problem, they reach better solutions than when they face a temporal form of the same problem. This difference dissapears when representation aids (e.g. matrix representations) are provided for the temporal problem. (Thomas et al, 1980)
Constraint processing	When a designer is facing a design problem, he implicitly adds constraints in order to solve it.
Meta strategy	During problem-solving designers use different strategies for solving a subproblem and the overcoupling problem at the same time.(Best and Simon, 2000)
Framing	When designers face a design problem, they will formulate it in a way that allows it to be solved. This formulation will include setting boundaries, and selecting parts and relations in the problem that designer will pay attention to in his next steps. (Schönn 1988)

Table 3. (continued)

Visser's argument is that all of these aspects play an important role in how a designer understands and solves problems. These are behaviors that designers exhibit when faced with a design problem; which is different from descriptive and prescriptive design methodologies. A certain design methodology might not apply (or not be applied) to a design situation at hand; while these cognitive aspects most certainly will. The cognitive aspects we described will have implications on how a designer works with design synthesis tools. These implications are discussed in table 4:

Cognitive aspect	Implications for design synthesis
Creative design	Creative applications of design synthesis involves unknown synthesis methods, unknown parameters, unknown problem decompositions. Synthesis methods should be matched to the problem; or created specifically. Parameters will be added during the design process, and to the synthesis method. The hierarchy of the problem will change; meaning synthesis methods will be assigned to different parts of the problem.
Semantic richness	Situation specific knowledge in design situations (e.g. business logic, application related knowledge) make synthesis methods provide better results; but synthesis methods that use it will be hard to reuse.

Table 4. Implications of cognitive aspects for design synthesis tools

Cognitive aspect	Implications for design synthesis
Satisficing	A designer will try synthesis methods to assess if suitable solutions exist. If no suitable solutions are likely, the designer will try a different direction to solve: effectively trying another synthesis method for the same problem.
Atomization	It gradually becomes clear what components are used. Synthesis methods may be applied with some components unknown; and be reapplied when components are added.
Opportunistic organization of design activity	Designers may want to use design synthesis methods that are actually used in other domains.
Ambiguity	Results from design synthesis methods can be interpreted in different ways, and information fed to design synthesis methods may also be ambiguous. In order to resolve this ambiguity, explicit translation of information may be necessary.
Problem simplification	Designers like to create small design synthesis methods that provide simple functionality.
Spatio-temporal planning	Designers can find better solutions if results are presented visually and can be manipulated in this medium; or if the design synthesis tool allows him to create rich representations.
Constraint processing	Allowing the designer to make his implicit constraints explicit improves his problem solving process.
Meta strategy	Designers want to use different design synthesis methods in conjunction with each other, while working on the same problem.
Framing	Designers should be able to pick which part of the problem a design synthesis method should work on.

Table 4. (continued)

These implications are futher explored when they are applied in problem scenarios.

Analysis

3.1 Roles

A first step in describing the problem scenarios a design synthesis tool is exposed to, is finding the roles of users interacting with it. Rather than describing more high-level goals, our description of goals is on the level of *end goals:* what a user wants to do. This allows us to directly relate the goals to actual tasks performed by the user, and functionality in the product. The goals are based on the tasks that the user performs in his existing design process, and which are relevant for design synthesis tools.

The roles are split into a number of user classes, following the approach explained in (Cooper et al. 2014). These classes can be defined as follows: *primary users* use a design synthesis tool to create new solutions, model design problems, or review sets of solutions. A primary user is the intended end-user of the design synthesis tool. The results the design synthesis tool may be used to create reports, track progress, or be used in other non-design tasks: such tasks are usually performed by *secondary users*. This type of users contributes or benefits directly from the synthesis tool, but isn't the intended end-user. Another type of user, that usually isn't a single user at all, is the *organizational user*. This role doesn't actively interact with the tool at all, but plays a role because his interests and decision-making has consequences for the usage of design synthesis. The primary users of smart synthesis tools are listed in table 4:

Role	Goals
Design engineer	Quicker redesign after design decisions. Better understand effects of changing requirements, goals, product features. More flexibility in making design decisions.
Systems engineer	Better system architectures through insights and assistance from the tool. Easier to create and manage system budgets for systems and subsystems. Understand which (sub)systems are restricting certain performances. Better verification of system behavior Easier modeling of system behavior

Table 5. Primary users and their goals

The secondary users are considered in table 6:

Role	Goals
Researcher	Higher quality design synthesis methodologies. Easier implementation of synthesis ideas. Better ability to test and evaluate methodologies. Easier to apply synthesis methods in projects.
Product manager	Investigate product innovation opportunities. Have better on causes of product limitations and performances. Make informed decisions about design strategies (such as component reuse).
Supplier	Increase in usage of supplied parts because it is easier for designers to find these parts. Better possibilities to communicate the (side-)effects or abilities of specific product features.
Manufacturer	Increased use of services because of better discoverability. More cost-effective when designs are optimized for manufacturer.
Customer	Richer configuration options of desired product. Better integration across products through redesign.

Table 6. Secondary users and their goals

We also consider the roles that organizations or institutions may play in table 7

Role	Goals
Product division	More informed choices about product lineup. Reuse of knowledge between products.
Research institute	Realistic opportunity to apply design synthesis in research projects.
Industry wide	Standarization helps developing derived products.
Collaborating teams	Reuse of knowledge between teams. Better understand effect of design decisions other team. Reduce integration problems between systems.

Table 7. Organizational roles and their goals

3.2 Environments

The environment the designer operates in tells a lot about how he works. The artefacts that are considered are the software the designer interacts with; this is in the direct environment a design synthesis tool also operates in.

Software	Information and process implications
3D solid modeler	Models may be created from design synthesis results; or form a basis for design synthesis. Parametric modeling allows for models to be represented as sets, and is often used as a common interface between software.
Product data management	Product data is of use in configuration synthesis, for picking alternative parts. Requires a complete and well-specified dataset to be effective, and interopability is difficult because data is often ambiguous or used differently between organizations.

Software	Information and process implications
Engineering simulation	Can either be used in design synthesis itself, or directly on the results of design synthesis, or after further design tasks. Only when integrated into design synthesis it will be able to handle large sets; but less tight integrations still provide valuable information and are likely to trigger redesign (effectively repeating design synthesis).
Email	
Spreadsheet	Results or specifications are often reported and discussed in a spreadsheet format.

3.3 Scenarios

Problem scenarios are used to investigate the effect of particular features of the world on people's activities and experiences. In the case of this research: the effect of design synthesis tools on how a designer behaves and thinks. The problem scenarios do not necessarily describe unwanted situations; but rather the situations a design synthesis tool is exposed to. This leads to implications for needs, opportunities, and problems as well. Further design responds to the problem situations by improving them; reducing the negative characteristics. The problem scenarios are not a requirements specification, but together with the associated implications, they do establish needs and opportunities for design synthesis tools. Effectively, design proposals can be checked to be coherent with the problem scenarios, and how the proposals adress the implications. This makes problem scenarios, while not a requirements specification, a suitable method to establish a well-defined problem.

The scenarios integrate aspects from all previous analyses in this text, including the literature review. Effectively this leads to taking into consideration: roles, goals, environment aspects, tasks, cognitive aspects, principles of set-based concurrent engineering. The problem scenarios highlight aspects of the current practices; the manner in which design synthesis tools are used now. Descriptions of scenarios are followed by implications about usability problems and causes for them.

Scenario 1: Designer uses design synthesis with CAD

1.1> The designer creates a simple design for a bottle in a 3D solid modeler

1.2> While looking at the 3d model he assesses what measures and features to make parametric, and what are logical constraints

1.3> He models three design variants with different stylistic features; all three variants have parameters for height, diameter and scale of stylistic features.

1.4> The designer wants to use a design synthesis tool to help pick variants of the bottle.

1.5> He creates a new method in the design synthesis tool that uses all the parameters in his CAD model; and other performances he is interested in: weight, volume, heat transfer.

1.6> When he tries to relate these performances to the design he knows which formulas are important; but he cannot use them on the (parametric) bottle designs.

1.7> Subsequently he creates a spreadsheet to create instances of the parametric model in the 3D solid modeler, and he uses engineering simulation software to find the performances he cares for

Aspect	**Cause**	**Effect**
Integration	There is no integration with 3d modelers or engineering simulators.	Parameters and constraints on the 3d model cannot be viewed or instantiated by the synthesis tool. Performances of solutions of a synthesis task cannot be calculated by external software. Results of work on 3d models are not automatically available in the tool. But exporting of parameters to a 3d modeler; importing values from engineering simulation allows for some minimal integration.
3d models	Cannot be stored; no expressions exist that operate on geometry;	Any work done with 3d models happens outside the design synthesis tool.

Table 9. Implications of scenario 1

Scenario 2: Designer creates model for cooling component

2.1> The designer creates a simple simulation model for a cooling element in numerical programming software.

2.2> After verifying the simulation model works, he ports the model into the design synthesis tool by transcribing all formulas.

2.3> He is now able to synthesize a cooling component, but he wants the model to be more precise and tries to extends on the model.

2.4> When he tries to add differential equations, he finds out the design synthesis tool doesn't support these; and there is no way to have calculations performed by other software.

2.5> Instead he now runs the simulation model in the numerical programming software, and exports the results for use in the design synthesis software

Aspect	Cause	Effect
Expressions	The language used for synthesis methods is limited in functionality.	Not every kind of model can be made or used (even if it works in numeric programming software).
Integration	There is no integration with numeric programming software.	Models that are created in numeric programming software (which is a likely starting point) must be translated and transcribed into the design synthesis tool.

Table 10. Implications of scenario 2

Scenario 3: Designer tries to identify matching components

3.1> The designer wants to pick a matching waterbottle and cooling element. He already has suitable synthesis methods for both the waterbottle and cooling element in the design synthesis tool.

3.2> The designer tries to understand how the cooling element affects the temperature of water in the bottle; bigger bottles need more cooling. He decides to add a simple heat transfer equation to the bottle synthesis, so that he can relate the both components.

3.3> He starts by picking a reasonable "first choice" for a small, medium and big bottle. For each of those points he broadens the selection to contain enough alternatives. He saves the three selections.

3.4> Then he opens a spreadsheet and transcribes the parameter ranges for each selection. He feels confident the ranges are broad enough that there's no need to update the spreadsheet when small changes are made to the bottle design.

3.5> Interpreating the heat transfer performances of the bottles; he can also make appropriate selections when he views synthesis results for the cooling element.

3.6> Again, he saves the selections and transcribes the parameter ranges to the spreadsheet. He now has three variants (small, medium, big) of bottle - cooling element combinations; but these are sets: no solutions have been picked.

3.7> The designer strategizes that he will pick the cooling element that is cheapest for it's cooling power; and then match bottles design so that they can be cooled within a certain maximum time. He does this by plotting the cost and cooling power of the cooling elements in the design synthesis tool; and selecting the best option by hand. By solving a simple linear equation he knows what the maximum volume he can pick for a bottle.

Aspect	Cause	Effect
Selecting	Selections can be stored; changed in size and dimensions; exported.	The designer can select sets of solutions that fit certain ranges, so that he can pick an actual point solution later.

Table 11. Implications of scenario 3

Aspect	Cause	Effect
Ranking	Ranking functions must be defined in the synthesis method; not in the results view.	When a designer wants to rank different components, he must add the ranking function to each synthesis method.
Comparing	No comparison can be made between selections; between results of different synthesis methods;	The designers resorts to an alternative workflow to make combinations between parts
Constraints	No relations (coupling effects) can be defined between synthesis methods	In order to satisfy constraints that exist because of interaction between components, the designer has to resort to manual comparing or repeatedly adding and chaning constraints.

Scenario 4: Designer uses parts from PDM

4.1> Opens the PDM and finds the parts that will be suitable for this. He exports the parts to a spreadsheet.

4.2> In the design synthesis tool he goes to the synthesis tool for that part; and imports the spreadsheet.

4.3> He has to create new parameters in the tool to translate the parameters that are known in PDF to parameters that are usable in the model.

4.4> He links the existing parameters to the new parameters; he does so via an expression

4.5> Now he can assign the columns in the spreadsheet to the newly created parameters in the model

4.6> And he can see the new results based on actual parts data in the screen

Aspect	Cause	Effect
Integration	Data from PDM is not automatically used in the design synthesis tool.	Changes and updates to (other) parts don't automatically lead to new synthesis results. The designer should manualy import information again. There is a chance design decisions are made based on outdated information.
Common interface	There is no common interface for the exchange of data; or still leaves ambiguity.	A user has to map data from one format to another; sometimes making interpretations and conversions. The procedure used for this is likely to be forgotten; the mapping introduces another "source of truth" which only adds to the already existing ambiguity.
Semantics	Expressions can use units (e.g. speed, surface); but lack further semantics.	Translations between different kinds of numerical data can be made; but it is easy to use wrong data (e.g. using the right kind of

Table 12. Implications of scenario 4

Aspect	Cause	Effect
		parameter (such as pace in min/km), but using it from the wrong kind of component (wheel instead of belt)).

Scenario 5: System engineer and product manager review a design

5.1> A system engineer and product manager sit down together to take a look at design results. The design synthesis tool is used to illustrate some of the effects certain design decisions would have.

5.2> For many of components a synthesis method was created in the software; the reviewers will evaluate results for all the componenents.

5.3> To explain and understand the effects of some design decisions, constraints are turned off and on, while reviewing the changes it has on the solution set.

5.4> Whenever design decisions across components are reviewed; matching parameters (that will create realistic and working configurations) are chosen for both components before viewing the results.

Aspect	Cause	Effect
Conditions	Results are displayed by value alone.	Results cannot be viewed conditionally (e.g. what results would change when a condition is changed); the designer only can acquire this knowledge by making changes to the tool and manually comparing results.
Combining	Results or selections can not be combined in one view.	Users who want to compare results of different synthesis methods (i.e. for different components) will have to do this via alternative workflows; e.g. by remembering or transcribing to a spreadsheet.

Table 13. Implications of scenario 5

Scenario 6: Systems engineer creates a system architecture

6.1> The designer is tasked with designing a watercooler. He will create a new design process unit (combination of analysis equation and design variables) in the design synthesis tool that simulates some of the important performances of the watercooler as a whole; while using coupling effects.

6.2> A new design process unit is created with a few parameters added: energy consumption, volume of bottle, cooling rate, cost of parts.

6.3> The designer divides the design into components; cooling element; bottle; power supply; microprocessor; tap; but doesn't model these parts in the synthesis tool. Rather he tries to determine coupling effects between the components, and put these into equations.

6.4> He adds these equations to the synthesis method and adds the parameters used in the equations

Aspect	**Cause**	**Effect**
Expressions	Multiple expressions, rules and parameters can be used in one synthesis method.	Coupling effects between objects can be modeled, even if the method has no understanding of what an object is.
Modeling	No objects or decompositions can be defined.	Questions such as "what is the energy consumption of subsystem A" are hard to answer because no such system or object can be defined.

Table 14. Implications of scenario 6

3.4 Usability problems found

This concluding paragraph lists the main causes and effects found in smart synthesis tools.

The use cases demonstrate a number of usability problems that are present in smart synthesis tools. The usability problems typically can be attributed to learnability, efficiency, memorability, errors, and satisfaction (Nielsen 1993). It seems that many of the usability problems lead to a negative sentiment in the user. This will likely worsen the attention, memory, performance and assessment of the user (Brave, Nass 2008). Usability problems identified through the use cases are:

- The user cannot complete the task in the tool, and is forced to use alternatives. This breaks the flow (of interactions) the user is in. Breaking this flow may lead the user to exit his task prematurely. (Cooper 2014, pg. 249)
- The user is forced to import and export data from other tools or manually transcribe it. Also any conversion of the data must happen per action of the user. The actions must be repeated when data changes. This adds to the physical work and turns the goal-directed task of the user into an excise task. (Cooper 2014, pg. 271)
- The way the a user programs his design problem into the software can be inconsistent with the modeling paradigm he uses. This makes it hard to learn using the tool, and cognitive work when translating the problem.

- The user cannot be sure if the data he uses in the tool is consistent with the data sources outside the tool. This contributes to a lack of trust, because this is likely to introduce errors (Cooper 2014, pg. 325).
- The user must use a plot to compare alternatives. This involves visual work: the user needs to find one object among many, decode the layout, distinguish between other visual elements. This turns the goal-directed task of the user into an excise task. (Cooper 2014, pg. 271)
- The user can not automatically repeat his review after starting a new problem; or easily access previous reviews. This leads to extra memory work, or using an alternative tool. This likely contributes to a lack of trust or breaks the flow.

3.5 Design requirements

A list of design requirements is created. This list is based on the found usability problems, the approaches used in smart synthesis tools and set-based design. Here a synthesis method is a programmed equation or procedure that the user can apply to his design problem. This is for example the analysis equation found in a DPU.
Creating synthesis methods:

- Particular equation or procedure used in a synthesis method can be changed by the user.
- A synthesis method can be created from another synthesis method.
- The user can apply transformations on the data he supplies to a synthesis method (such as unit conversion)

Using synthesis methods:

- The user can assign values to parameters
- The user can specify the type or relation of parameters
- The user can assign multiple synthesis methods to one design problem
- Synthesis methods are able to use other synthesis methods

Set based design:

- Parameters can be specified in ranges
- Multiple instances of the same design problem can be specified, resulting in a set of solutions.
- Multiple solution sets can be compared and reviewed in one view.
- Sets can be narrowed down: parameter values or constraints can be assigned to a complete set at once.
- After a user specified a set, it is remembered even when the design problem of synthesis method changes.

Integration with external tools or data:

- A synthesis method can use external software to perform calculations
- On the availability of new data (from external sources), this data must be automatically imported, recalculation happen, and the synthesis method is reevaluated so that the results are in sync with this new data.
- On changes in the design problem or synthesis method, the synthesis method is reevaluated so that the results are in sync with these changes.
- Transformations applied to data (e.g. conversion of units, or an averaging function) are able to be saved and reused later.
- The user can specify transformations that are applied to data that is automatically imported.

Proposal

This section presents a proposal for a design synthesis tool. This proposal addresses the usability problems found in the analysis. The architecture and functionality of the proposal are described.

An overview of the proposed design synthesis tool can be seen in figure 3. This figure shows how the important components interact: there is a core component that orchestrates instruments to models; and queries these instruments for results. There are instruments, which are a way to implement design synthesis methods, which can be discovered by the core through the registry. And there is a automation environment, which is a GUI that implements all the functionality present in the core component.

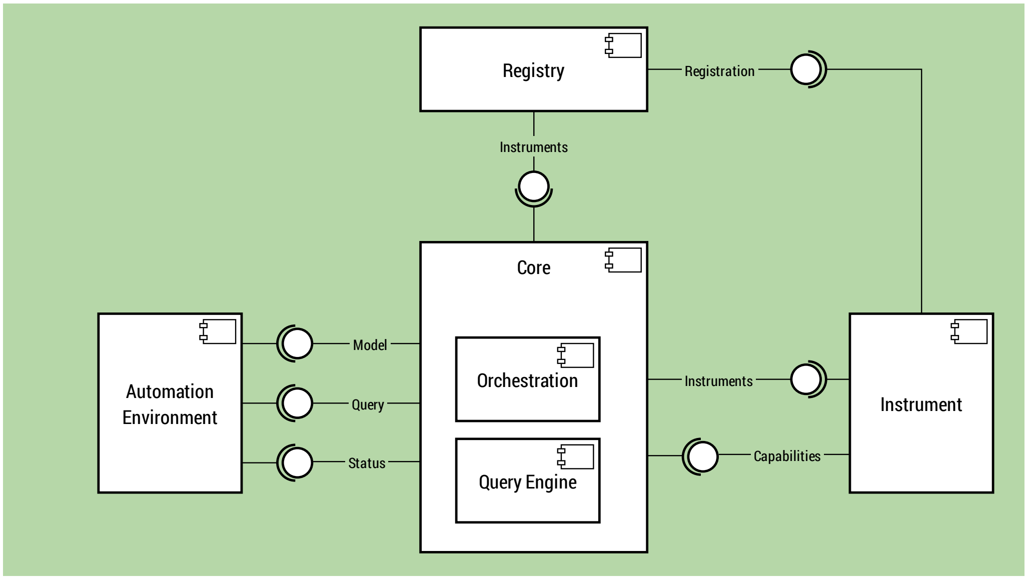


Figure 3. Component diagram

4.2 Design automation instruments

The instruments architecture is an approach to providing design synthesis methods as a combination of small services.[4] Each instrument is a service: a small program, that runs by itself, and can communicate with other services. Instruments are built to fulfill specific automation *needs* and are usable independent of other components. For example, an instrument may perform a synthesis tasks, while another

4. Services are an architectural style in writing computer software.

instrument is able to analyse solutions. It may be helpful to clarify the distinction with programming libraries and agents. Libraries are used by programmers as a means of code reuse, and are not working programs. Agents, from the field of artificial intelligence, are used to describe autonomous systems, focussing on the mechanisms that will lead to intelligent behavior. Agents and instruments can be seen as complementary ideas: it is certainly true that individual instruments can act as an agent, but the instrument architecture provides a way to build and deploy these instruments.

An instrument file is the description of an instrument that fullfils aspects of design. For example, an instrument can contain procedures that create a component layout and grade the associated reliability. Instrument files are used to create *live instruments:* running instances of instruments that can be interacted with. Editing instrument files is a way to create new instruments or modify existing instruments. An instrument being a file makes it portable: it can be send to others to be used in their environment.

To have an instrument perform operations it needs to run; this is done by creating a *live instrument* from an instrument file. Such a live instrument can be used as-is, it is not necessary to use it in conjunction with the *orchestration* or *registry* components. This behavior is realized by the *instruments api*.

Building instruments

A basic instrument consists of a number of methods and properties, as can be seen in figure 4. Every instrument uses this basic interface. The interface ensures that instruments can be queried: this is the manner in which they are used in a design situation. A user can create new instruments by creating new methods. These are pieces of code that follow a certain style so that they are usable in instruments: the methods define input and output variables, and can represent any kind of logic: rules, procedures, expressions, etcetera. The author of an instrument can then link to the method in the instrument file. This instrument file also contains a pipeline: the order in which methods are processed, and how their inputs and outputs relate to eachother. A pipeline does not have a very different appearance than a method: it has inputs, outputs, and does processing work.

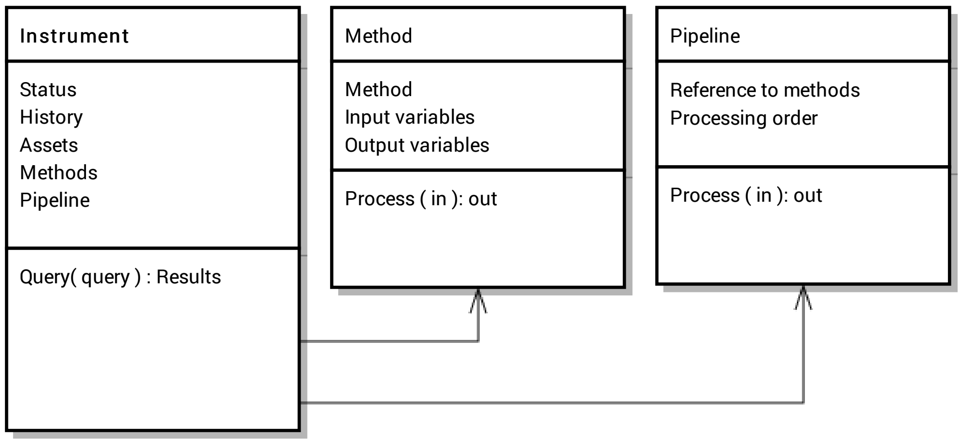


Figure 4. Class diagram of an instrument

The instrument can also contain actual assets that are used in the methods: tables or other types of information. An instrument accepts a query, a basic instrument interpretes this query and returns status or history information it remembered, or processes to acquire new results.

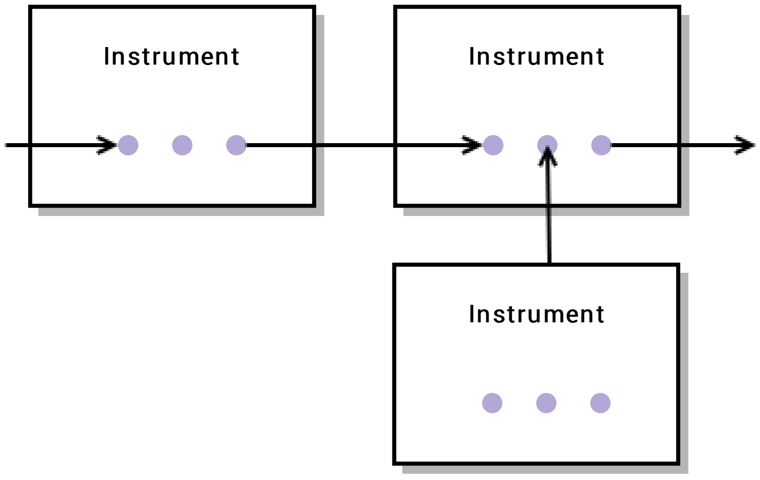


Figure 5. Showing composition of instrument pipelines

Composing instruments

The pipelines of instruments can be composed. An illustration of this can be seen in figure 5. This can be done by creating a new instrument file that references two (or more) other instruments. By placing two instruments after eachother in the pipeline, their individual pipelines are processed sequentially. We also consider that pipelines may be inserted at any point in the pipeline of another instrument, to modify its behavior.

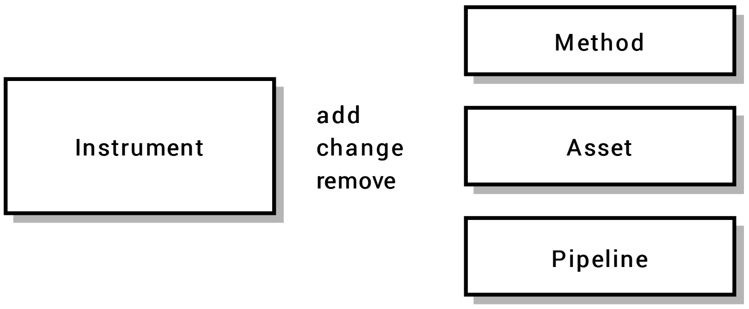


Figure 6. Showing modification possibilities of instruments

Modifying behavior

A design synthesis method may use rules, expressions, methodologies that the user wants to replace. Making a new instrument just because the user wishes to replace a rule is not a desirable granularity. It is better if instruments can be modified at the level of rules, expressions, procedures. But instead of making these changes directly to the instrument, which loses the original definition, a mechanism can be used that modifies parts of the instrument explicitly. An instrument file then contains a reference to an original instrument, and what modifications the user wishes to make. In fact this is a way too bootstrap from the most *basic instrument,* which contains logic to act as an instrument but no methods or pipeline: every instrument can be created by defining modifications to this basic instrument. Modification of an instrument is illustrated in figure 6.

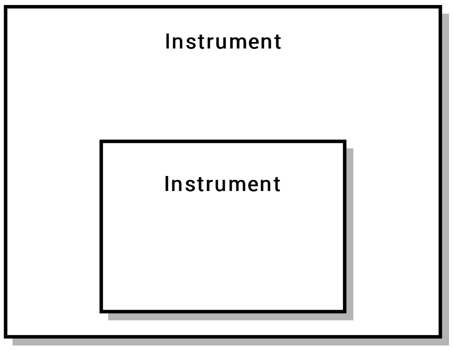


Figure 7. Showing a contained instrument

Reducing complexity of instruments

This paragraph highlights one kind of modification that can be very useful. As illustrated in figure 7, this modification consists of wrapping an instrument within another instrument, while not overriding any method of that instrument. What happens is that the outer instrument adds methods to the front and back of the wrapped instrument. Such a construction can function as a conversion of values for the inner instrument, guarding certain parameters, coupling effects; it is a good way of reducing the complexity of instruments.

Deploying instruments

After an instrument file is build, the file can be send everywhere and be used. The file contains all the knowledge, procedures and functionality to operate on its own. An instrument that is running, accepting queries and returning results is called a *live instrument.* It is shown later what kind of queries can be used.

This bare bones way of building and deploying instruments are not the only option. Some users may be more comfortable with an easier approach.[5] Editing the definitions of an instrument has expressive power, but requires expert knowledge—not something we can expect from every user. Some techniques that are used in instrument files can via another user-interface as well: by *orchestration.*

Taxonomy

5. This is concluded from the personas we described earlier. Roles that develop design synthesis methods are geared towards editing instrument files; roles that want to work on design problems are geared towards the orchestration UI.

Now, we will consider a basic taxonomy for instruments. This ordering is not meant to be complete or definite, but will give an overview of what functionality and *granularity* we believe to be effective.

Category	**Description**
Model providers	These are instruments that generate or modify the problem description or model.
Asset providers	May contain tables, 3d models, PDM data, or anything else that can be used by a method.
Rule / Logic providers	Contain simple predicates or expressions (greater than, smaller than, equals) that can be used to accept or reject results.
Synthesis providers	These are methods that are used for creating results from input parameters.
Analysis providers	These methods are used to find performances of results.
Reasoners	Interpret results and can decide to make adjustments in a strategic manner.
Solvers	Repeatedly create results until certain criteria are met, while gradually changing the input parameters with each iteration.
Translators	May translate between types, units, ontologies, parameter couplings

Table 15. Basic taxonomy of instruments

4.3 Registry

The registry is a component that is used to track which instruments are being used. It allows in instrument to be registered for use; so that the design synthesis environment can discover the instruments it needs when it is solving a problem. It is important from the user's perspective as well: he wants to know which instruments are available to use, and what their capabilities are. The registry is very simple: it maintains a list of instruments and registration and deregistration of instruments can take place. But even while it is simple, it is a necessary component to make instruments, which adhere to a microservice architecture, discoverable to the design synthesis environment.

4.4 Core

What we consider the *core* components, is the software that is responsible for using instruments on models. Which results in a easier user interaction with the instruments. The next section considers the usability concerns of using instruments without the core component.

Usability issues with using instruments directly

The usability of design automation is the main concern in this research, and working with instruments in a direct manner leads to usability issues straightaway. This is because of the amount of manual effort that is required to work with instruments this way, which consists of:

1. Providing the instrument with the information it requires to operate.
2. Displaying results in manner that contributes to actual the task at hand.
3. Repeating this for whenever multiple instruments are used.

This way of working is not an usability issue in itself, but elicits bad mannerisms—something we try to prevent. Applied in the context of a design task, to give a meaningful result, a user will have to repeat these interactions and plan them in relation to other actions. When considering this use case, the direct manner of working likely causes the following bad mannerisms:

1. Context switching:[6] not only does the long duration (even worse when multiple instruments are used) of the required interactions lead to an interuption, the actions are also very different in nature than design tasks. The frequency with which the actions need to be performed only makes things worse.

2. Delaying of the automation task: when instead of using the instrument at the earliest or most convenient opportunity, the user waits with using it. This happens because the information provided by the instrument may not be very usable for the user at all times; instead of repeating the actions at a later time, he rather waits until he actually needs the information. This may lead to reduced effectiveness of the tool: the user wastes the opportunity to get useful information early.

3. Hoarding: when the user keeps accumulating the information he wants to provide to the instrument—withholding incremental changes. This happens because the amount of work needed to use the instrument isn't in a 1:1 relation to the amount of information the user provides. Thus the user gains time by accomulating the information. The effect of this is that the user doesn't get feedback from the instrument on incremental information changes.

Orchestration

6. The considerations, understanding, planning, memory (amongst other things) of a user all relate to the context he is in. It takes time and effort for someone to adjust to new contexts. When someone changes the tool he is working with, his context changes as well. A forced switch in context (such as an interuption) is likely to have even worse effects, such as forgetting something.

Orchestration is the functionality to assign instruments to the problem descriptions. When a model is provided, the orchestration defines which instruments are assigned to which part of the model. This allows to send specific information of a part to a certain instrument, and also to maintain the results returned by an instrument in relation to the model. Assignment can happen through rules, for example: *"assign instrument 1 to all nodes of type N".* When multiple instruments are assigned to a part of the model, results from all instruments are returned. A query can then make the distinction between what results to display or use.

Query engine

The query engine executes queries. Many existing synthesis methods and tools are focussed on the aspect of showing solutions; which may actually not be the most important aspect. As is discussed earlier, design methods are only used to find intermediate solutions; to later adjust the problem that's focussed on, or find new solution directions. In order to do this a designer is not only interested in the solutions itself; but on other knowledge that is embedded in the set of solutions; the tool he used; he extracts this knowledge and tries to apply it to create new understandings. Querying is a proposal that will address this need for explanations.

This research will not touch on the exact working of querying. Queries are usually to a databse (in the case of the proposal to one or multiple instruments), and the database returns information that is asked for. In the case of this proposal, such results can be the results of design synthesis methods, but also other information. table shows a few types of information that can be queried for. What is more important is that queries allow these types of information to be requested in relation to eachother. Such as: *"which of all earlier results of instrument 1 has the highest performance in parameter X".*

Information type
Synthesis results
Earlier results
Used parameters
Rejected or accepted rules
Which instrument generated a result

Table 16. Information that can be returned by a query

Query languages used in databases are much more expressive than just looking up values. Query languages usually not only select certain values; but also filter them; group them; and select them conditionally to other values. A query in natural language might be *"select all results that were rejected by a rule in instrument 1"*. The operations allowed by query languages lead us to consider queries for the proposal: the expressiveness will make it possible for the designer to get useful information from the instruments: leading to explanation and understanding.

Design automation environment

Whereas the core provides all the functionality to use the proposal for a design synthesis tool; a design automation environment make the use of this functionality more pleasant. It was already discussed that the core functionality provides an improvement in the usability of instruments; similarly a design automation environment can provide such a improvement to all the core functionality.

A design automation environment will be a graphical user-interface to the design synthesis tool. It will present all functionality of the design synthesis tool in a graphical manner. This research does not concern itself with the graphical side of the design synthesis tool, but it is imperative that without proper plotting or display of results design synthesis stays hard to use.

Results

This section provides an overview of the results of the proposal. table 17 shows what functionality is present in the proposed tool as compared to the functionality present in the tested smart synthesis tool. These functionalities follow from the design requirements section earlier in this report. table 18 shows what functionality leads to an improvement of what problem.

Functionality	Proposal	SST
Creation and deployment of design synthesis methods		
Combining fragments	Yes	No
Overruling used procedures and rules	Without losing old rules and procedures	By permanent modification
Add parameters	Extending with traits	Yes
Divide responsibilities over multiple methods	Yes	No
Deployment	Microservice	From within tool
Execution	Always-on	At user action
Assignment of design synthesis methods to design problems		
Assign values to parameters	Yes	Yes
Describe relation to design problem	Yes	No
Use multiple methods on one design problem	Via orchestration	Not explicitly
Relate methods to eachother	Via orchestration or composition	No
Set based concurrent engineering principles		
Divide results into selections	With a query	Yes
Compare selections	With a query	No
Narrow down selections	With a query	Yes
Stay comitted to sets	Via history and branches	No
Integration		
With other software	Programmatically via instrument	No
Import and export of data	Automated via instrument	Manualy
Interpreting results		

Table 17. Comparison of functionality in proposal and SST

Functionality	Proposal	SST
Scatterplot for any dimension	Not described	Yes
In relation to concurrent versions of the design	With a query	No
In relation to older versions of the design	With a query	No
In relation to rejection by a rule	With a query	No
Knowing which rule has stable performance	With a query	No
Knowing which rule limits performances	With a query	No

Table 17. (continued)

It is clear that the proposal aims to provide more functionality in comparison with the smart synthesis tools. The functionality that is described in the proposal will improve a large number of identified usability problems:

	Instrument	Orchestration	Querying	Environment
Inability to organize methods	x			
Inability to reuse methods	x			
Inability to easily modify method	x			
Inability to interpret results			x	x
Inability to understand effect of changes			x	x
Inability to formulate design problem				x
Inability to relate method to problem		x		x

Table 18. Causes of usability problems adressed by proposal features.

Conclusion

Design synthesis tools have not been adopted by the industry at the same rate as other CAD solutions; with 3d parametric solid modelers being the main tool of the trade. A big promise is shown by design synthesis tools: they do well in making design knowledge explicit, and usable by automatizing design tasks. But they fail to attract much attention of engineers; the use of the tools is problematic for them because it is hard to fit them in their way of working. Changes need to be made that make design synthesis tools usable to designers: in their environment, their way working, and their way of thinking.

This research has shown features that design synthesis tools can use to let them provide more insight in design problems. A design synthesis method contains a lot of knowledge information about how a design is made. This information is used to automatically generate sets of solutions; but the knowledge is of much greater value when a designer can relate aspects of a solution (or set of solutions) to design decisions. By letting a design synthesis tool remember for each generated solution what steps are taken to create it; or what rules rejected a solution; the designer is provided with much richer information. He can put this information to use to understand aspects of the design and design method; e.g. what design decision made two alternative designs different from eachother? Or: which rule is badly affecting a certain performance aspect? Allowing users to ask such questions—relating generated solutions to steps of the design synthesis method—turns a design synthesis tool into a source of explanations.

Typically in design processes, designers make changes to intermediate solutions (or problem definitions) based on aspects they find in these solutions. We have described functionality that let designers redefine design synthesis methods as a reaction to found solutions. Knowing which step of a design synthesis method affects certain aspects of a solution is a start. The next step is allowing a designer to take action based on this new learned information; by letting him make changes to the used design synthesis method. With the differing natures of rules and procedures—some specific to the design problem, some are general methods, some originate from different domains, etcetera—a design synthesis method can become unwieldy, hard to make changes to or apply in new design situations. Composition allows the designer to organize (parts of the) design synthesis method: it explicitly describes how the method is created from (simpler) original parts. Reordering parts of a design synthesis method, swapping a used method with another, introducing new rules; all these alterations are described in an instrument file. It allows the designer to make design synthesis methods from general methods; and gradually introduce logic needed for the specific design situation; making the designer more responsive to design decisions.

Current design synthesis tools do not function in a way that reflects the way a designer thinks and behaves. This conclusions follows after researching the smart synthesis tools developed at University of Twente. Subjecting the tool to principles of set-based concurrent engineering, design cognition, and study on current practice, shows a number of shortcomings in the way a designer can interact with the tool.

There is no emperical evidence that the proposed tool is more usable than existing design synthesis tools. Simple experiments that do not require a fully functional prototype may give some early insight in major shortcomings of the proposed solution; this is standard practice in user-centered development. But it may not be until a fully functional prototype is used in real design situations that actual shortcomings or improvements can be shown. Also many of the implications made in this research; primarily on the relation between design cognition and design synthesis tools; are not verified. Any of these implications based on design cognition can be researched in an experiment; so that a stronger argument can be made for the importance of these cognitive aspects. Other research based on this work, other than emperically verifying the arguments made, may be in the direction of a graphical user-interface that is used with the proposed tool, or further specification of the interfaces, APIs and query language.

6.2 Self-assessment

I want to include a quick self-assessment about doing this research:

1. I'm happy and confident about the amount of new materials I've learned and familiarized with on the topics of: human-computer interaction, usability engineering, scenario-based design, user-centered design. I gained a lot of new knowledge by familiarizing myself with the research fields of computational design synthesis, design cognition and design methods.
2. Finding a good methodology for this kind of research was hard. I expect an empirical approach to have more scientific value, but such an approach is difficult to learn and apply, especially in a limited time.
3. Planning proved to be a complicated puzzle. Many tasks took more time than I expected. I expect to have improved in this aspect by the experience gained during this research; I have a better understanding of how much time certain

tasks cost. I think I can improve in the communication with my tutor by providing better agenda and being more punctual in delivery of results.

Bibliography

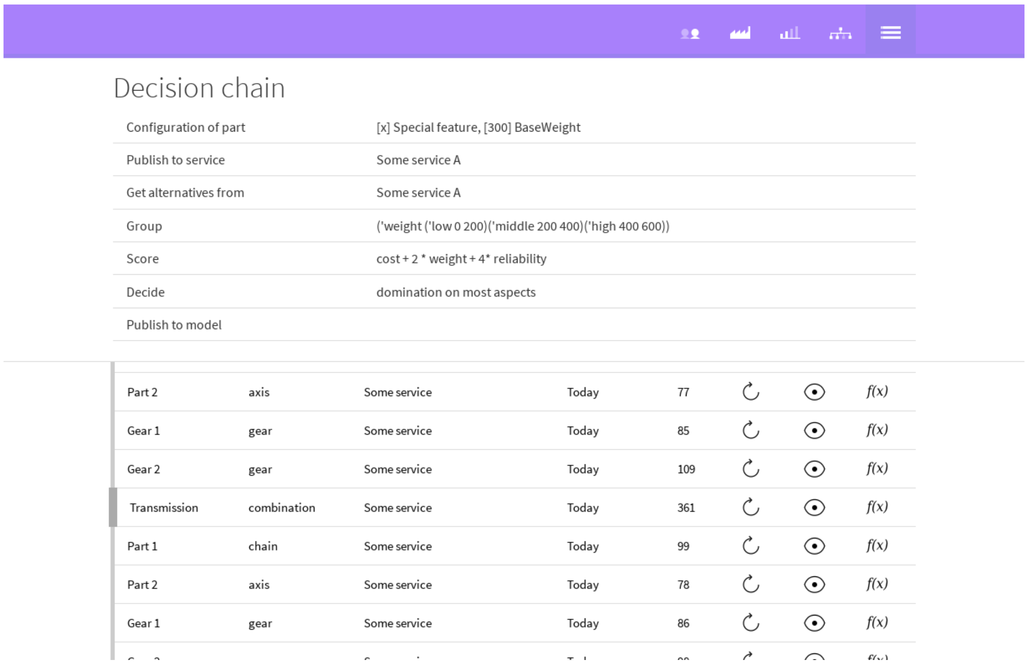
- Antonsson, Erik K., and Jonathan Cagan, eds. Formal Engineering Design Synthesis. Cambridge, UK; New York: Cambridge University Press, 2001.
- Best, Bradley J, and Herbert A Simon. "Simulating Human Performance on the Traveling Salesman Problem." In Proceedings of the Third International Conference on Cognitive Modeling. Universal Press, Groningen, 42-49, 2000.
- Bhaskar, R, and Herbert A Simon. "Problem Solving in Semantically Rich Domains: An Example from Engineering Thermodynamics*." Cognitive Science 1, no. 2 (1977): 193-215.
- Brave, Scott, and Clifford Nass. "Emotion in Human-computer Interaction." In The Human-Computer Interaction Handbook, Second edition., 77-92, 2008.
- Campbell, Matthew I, Jonathan Cagan, and Kenneth Kotovsky. "A-Design: An Agent-Based Approach to Conceptual Design in a Dynamic Environment." Research in Engineering Design 11, no. 3 (1999): 172-92.
- Campbell, Matthew I, Jonathan Cagan, and Kenneth Kotovsky. "The A-Design Approach to Managing Automated Design Synthesis." Research in Engineering Design 14, no. 1 (2003): 12-24.
- Chakrabarti, Amaresh, Kristina Shea, Robert Stone, Jonathan Cagan, Matthew Campbell, Noe Vargas Hernandez, and Kristin L Wood. "Computer-Based Design Synthesis Research: An Overview." Journal of Computing and Information Science in Engineering 11, no. 2 (2011): 021003.
- Chandrasegaran, Senthil K., Karthik Ramani, Ram D. Sriram, Imré Horváth, Alain Bernard, Ramy F. Harik, and Wei Gao. "The Evolution, Challenges, and Future of Knowledge Representation in Product Design Systems." Computer-Aided Design, Solid and Physical Modeling 2012, 45, no. 2 (February 2013): 204-28. doi:10.1016/j.cad.2012.08.006.
- Cooper, Alan. About Face: The Essentials of Interaction Design, 4th Edition. 4th edition. Indianapolis, IN: John Wiley and Sons, 2014.
- Deng, Y.-M. "Function and Behavior Representation in Conceptual Mechanical Design." AI EDAM 16, no. 05 (2002): 343-62.
- Dorst, Kees, and Pieter E. Vermaas. "John Gero's Function-Behaviour-Structure Model of Designing: A Critical Analysis." Research in Engineering Design 16, no. 1-2 (April 8, 2005): 17-26. doi:10.1007/s00163-005-0058-z.

- Draijer, Harry, and Frans Kokkeler. “Heron’s Synthesis Engine Applied to Linkage Design: The Philosophy of WATT Software.” In ASME 2002 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, 1405–9. American Society of Mechanical Engineers, 2002.
- Gero, John S., and Udo Kannengiesser. “The Situated Function-behaviour-structure Framework.” Design Studies 25, no. 4 (July 2004): 373–91. doi:10.1016/j.destud.2003.10.010.
- Hirtz, Julie, Robert B. Stone, Daniel A. McAdams, Simon Szykman, and Kristin L. Wood. “A Functional Basis for Engineering Design: Reconciling and Evolving Previous Efforts.” Research in Engineering Design 13, no. 2 (2002): 65–82.
- Jauregui-Becker, J. M., H. Tragter, and F. J. A. M. van Houten. “Structure and Models of Artifactual Routine Design Problems for Computational Synthesis.” CIRP Journal of Manufacturing Science and Technology, Design Synthesis, 1, no. 3 (2009): 120–25. doi:10.1016/j.cirpj.2008.10.002.
- Jauregui-Becker, J. M., and W. W. Wits. “Knowledge Structuring and Simulation Modeling for Product Development.” Procedia CIRP, 1st CIRP Global Web Conference: Interdisciplinary Research in Production Engineering (CIRPE2012), 2 (2012): 4–9. doi:10.1016/j.procir.2012.05.030.
- Jauregui-Becker, Juan Manuel. From How Much to How Many: Managing Complexity in Design Automation, 2010.
- Maher, Mary Lou, and Josiah Poon. “Modeling Design Exploration as Co-Evolution.” Computer-Aided Civil and Infrastructure Engineering 11, no. 3 (1996): 195–209.
- Maher, Mary, and Hsien-Hui Tang. “Co-Evolution as a Computational and Cognitive Model of Design.” Research in Engineering Design 14, no. 1 (February 1, 2003): 47–64. doi:10.1007/s00163-002-0016-y.
- Navinchandra, Dundee, Katia P Sycara, and S Narasimhan. “A Transformational Approach to Case-Based Synthesis.” Artificial Intelligence for Engineering, Design, Analysis and Manufacturing 5, no. 01 (1991): 31–45.
- Nielsen, Jakob. Usability Engineering. Academic Press, 1993.
- Reitman, Walter Ralph. Heuristic Decision Procedures, Open Contraints [sic], and the Structure of Ill-Defined Problems. Graduate School of Industrial Administration, Carnegie Institute of Technology, 1964.
- Rosson, Mary Beth, and John Millar Carroll. Usability Engineering: Scenario-Based Development of Human-Computer Interaction. Morgan Kaufmann, 2002.

- Schön, Donald A. “Designing: Rules, Types and Worlds,” 1988.
- Schotborgh, Wouter O., Chris McMahon, and Fred JAM Van Houten. “A Knowledge Acquisition Method to Model Parametric Engineering Design Processes.” International Journal of Computer Aided Engineering and Technology 4, no. 4 (2012): 373–91.
- Sobek, Durward K., Allen C. Ward, and Jeffrey K. Liker. “Toyota’s Principles of Set-Based Concurrent Engineering.” Sloan Management Review 40, no. 2 (1999): 67–84.
- Thomas, JC. “Problem Solving by Human-Machine Interaction.” In Human and Machine Problem Solving, 317–62. Springer, 1989.
- Visser, Willemien. “Dynamic Aspects of Design Cognition,” 2004. https://hal.inria.fr/inria-00071439/.
- Ward, Allen C. “Mechanical Design Compilers.” In Formal Engineering Design Synthesis, edited by Erik K. Antonsson and Jonathan Cagan, 428–41. Cambridge University Press, 2001.
- Ward, Allen C, and Warren Seering. “The Performance of a Mechanical Design’Compiler’,” 1989.

Appendix A: Sketches for UI

Various sketches for an UI helped shape the vision for the final system architecture. It shows ideas for modeling the design problem, and automatic review / evaluation of the generated solutions.

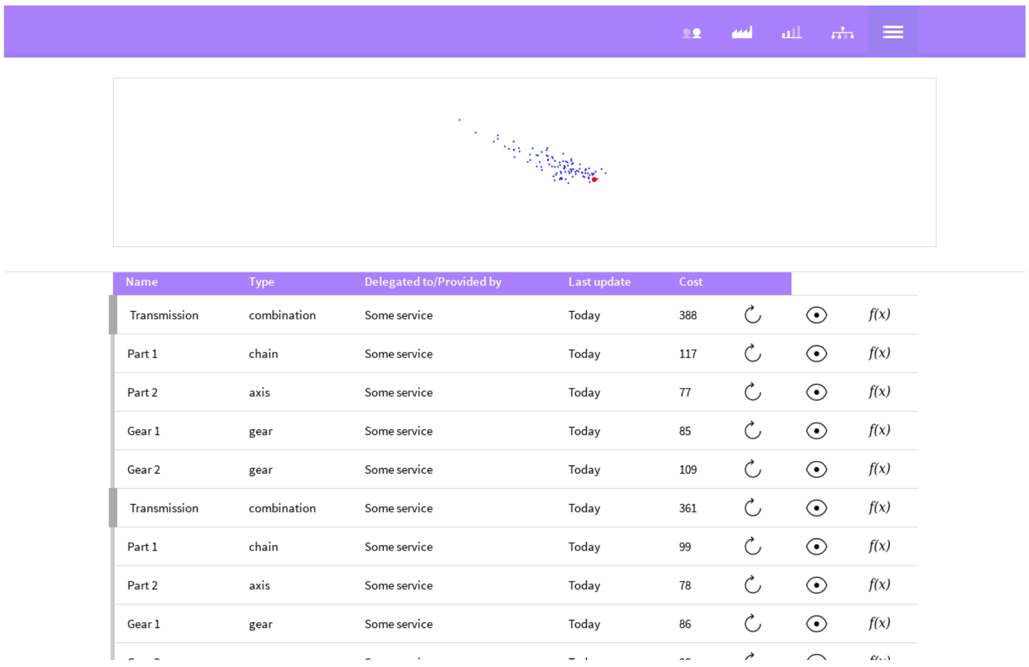


Decision chain

Configuration of part	[x] Special feature, [300] BaseWeight
Publish to service	Some service A
Get alternatives from	Some service A
Group	('weight ('low 0 200)('middle 200 400)('high 400 600))
Score	cost + 2 * weight + 4* reliability
Decide	domination on most aspects
Publish to model	

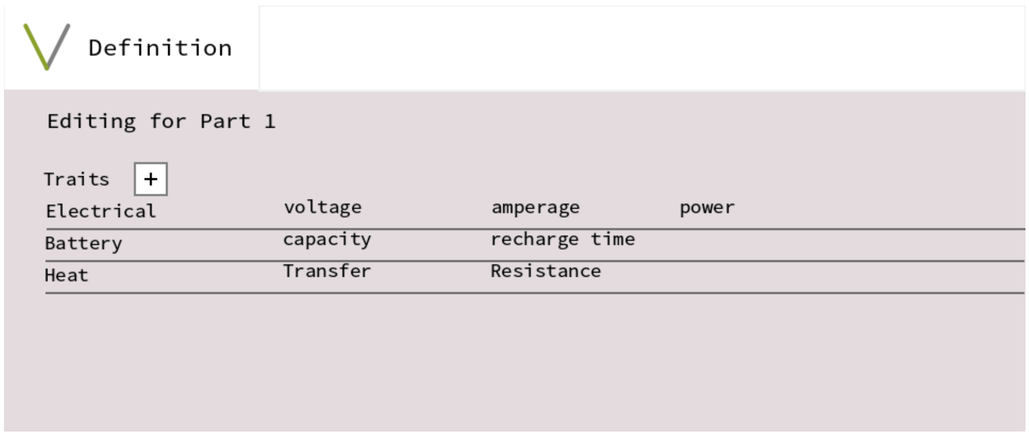
Part 2	axis	Some service	Today	77	↻	⊙	$f(x)$
Gear 1	gear	Some service	Today	85	↻	⊙	$f(x)$
Gear 2	gear	Some service	Today	109	↻	⊙	$f(x)$
Transmission	combination	Some service	Today	361	↻	⊙	$f(x)$
Part 1	chain	Some service	Today	99	↻	⊙	$f(x)$
Part 2	axis	Some service	Today	78	↻	⊙	$f(x)$
Gear 1	gear	Some service	Today	86	↻	⊙	$f(x)$

Figure 8. An UI showing automatic review by a decision chain



Name	Type	Delegated to/Provided by	Last update	Cost			
Transmission	combination	Some service	Today	388	↻	⊙	f(x)
Part 1	chain	Some service	Today	117	↻	⊙	f(x)
Part 2	axis	Some service	Today	77	↻	⊙	f(x)
Gear 1	gear	Some service	Today	85	↻	⊙	f(x)
Gear 2	gear	Some service	Today	109	↻	⊙	f(x)
Transmission	combination	Some service	Today	361	↻	⊙	f(x)
Part 1	chain	Some service	Today	99	↻	⊙	f(x)
Part 2	axis	Some service	Today	78	↻	⊙	f(x)
Gear 1	gear	Some service	Today	86	↻	⊙	f(x)

Figure 9. An UI showing evaluation by scatter plot



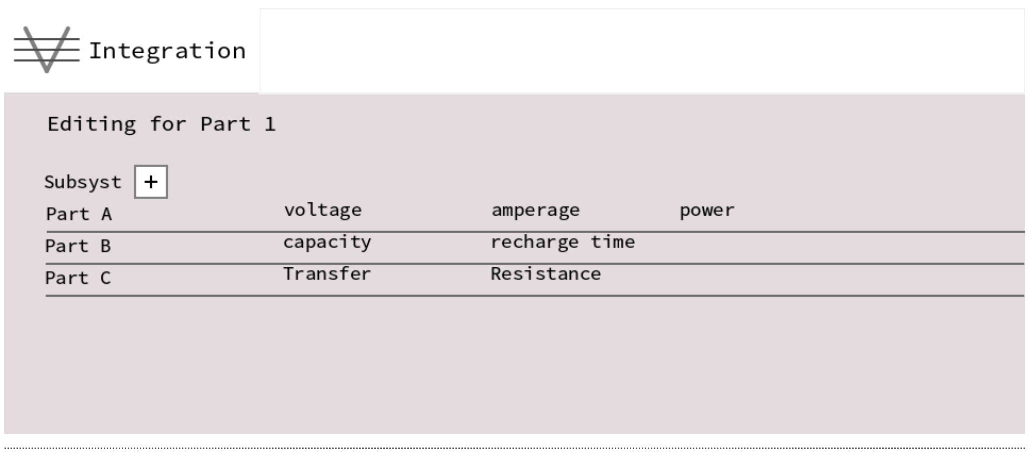
Definition

Editing for Part 1

Traits +

Electrical	voltage	amperage	power
Battery	capacity	recharge time	
Heat	Transfer	Resistance	

Figure 10. An UI for setting the type of a part



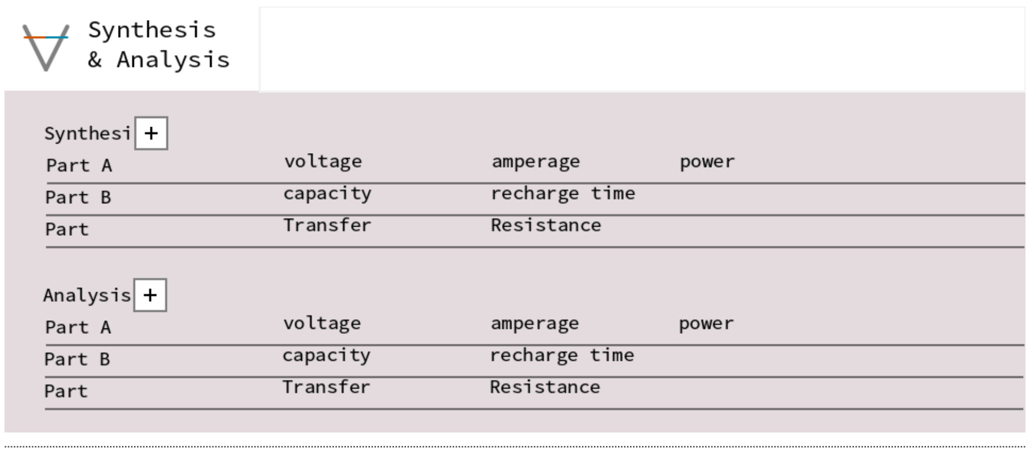
Integration

Editing for Part 1

Subsyst +

Part A	voltage	amperage	power
Part B	capacity	recharge time	
Part C	Transfer	Resistance	

Figure 11. An UI for combining systems



Synthesis & Analysis

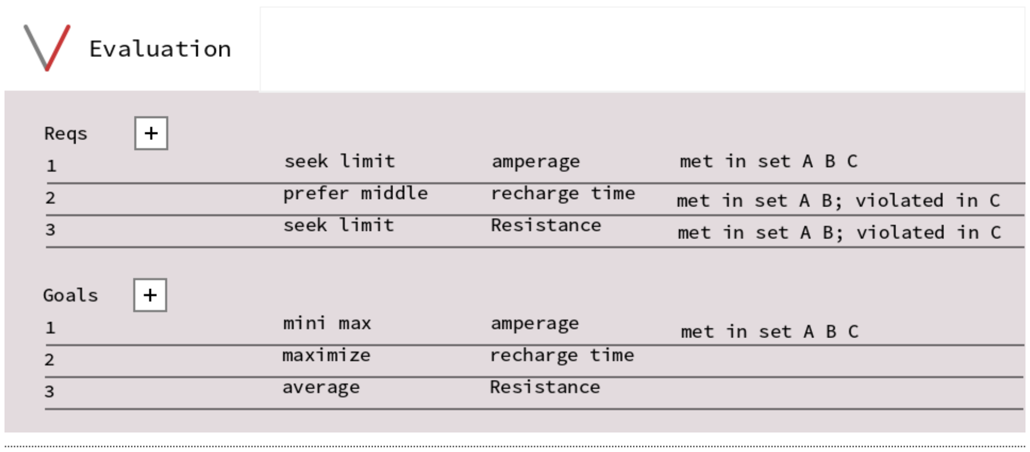
Synthesi +

Part A	voltage	amperage	power
Part B	capacity	recharge time	
Part	Transfer	Resistance	

Analysis +

Part A	voltage	amperage	power
Part B	capacity	recharge time	
Part	Transfer	Resistance	

Figure 12. An UI for setting design parameters



Evaluation

Reqs +

1	seek limit	amperage	met in set A B C
2	prefer middle	recharge time	met in set A B; violated in C
3	seek limit	Resistance	met in set A B; violated in C

Goals +

1	mini max	amperage	met in set A B C
2	maximize	recharge time	
3	average	Resistance	

Figure 13. An UI for automatic review of solutions