NTNU
Norwegian University of
Science and Technology

Knowledge Based Engineering and Metacognition for Creative Design

Eivind André Taftø

Master of Science in Engineering and ICT

Submission date:	February 2016
Supervisor:	Ole Ivar Sivertsen, IPM
Co-supervisor:	Ivar Marthinusen, IPM
	Christos Kalavrytinos, IPM

Norwegian University of Science and Technology
Department of Engineering Design and Materials

Knowledge Based Engineering
and
Metacognition
for
Creative Design

Eivind A. Taftø

Submitted as partial fulfillment of the
requirements for the degree of

Master of Engineering and ICT

Department of Engineering Design and Materials,
Norwegian University of Science and Technology

2016

THE NORWEGIAN UNIVERSITY
OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF ENGINEERING DESIGN
AND MATERIALS

**MASTER THESIS AUTUMN 2015
FOR
STUD.TECHN. EIVIND TAFTØ**

KNOWLEDGE BASED ENGNEERING (KBE) AND METACOGNITIVE PRINCIPLES FOR THE EARLY CREATIVE DESIGN PHASE

Knowledge Based Engineering (KBE) og metakognitive prinsipper for tidlig kreativ design-fase

Talking about KBE applications today, we often refer to mature technologies where well-defined product family design is automated through parameterization and Multi Model Generation (MMG). In this master thesis, we will investigate if and how KBE could support the early creative design phase. In particular, we will consider Eskild Tjalve's book "Systematic Design of Industrial Products". In addition, we will study how metacognition could apply to KBE and the creative phase. In summary:

- How could metacognitive principles benefit KBE and the creative design phase?
- How could KBE technologies support or automate the early creative phase of designing a simple product, for instance a coffee maker?
- How could these methods apply to the design of more complex products, for instance industrial process plants?

The assignment includes:

1. Litérature research on creativity, creative processes, metacognition, KBE and Tjalve's "Systematic Design of Industrial Products".
2. Identify how KBE along with metacognitive principles can support creativity.
3. Find an example product suitable for implementation in a test KBE creative system.
4. Develop a minimal KBE environment showing how KBE can aid in creative processes.
5. Investigate the possibility of using such an environment for design of a more complex product, for instance an industrial process plant.
6. If time allows, develop a generic pilot KBE application for creative design work with reference to Tjalve's design methodology.

Formal requirements:

Three weeks after start of the thesis work, an A3 sheet illustrating the work is to be handed in. A template for this presentation is available on the IPM's web site under the menu "Masteroppgave" (https://www.ntnu.no/web/ipm/masteroppgave-ved-ipm). This sheet should be updated one week before the master's thesis is submitted.

Risk assessment of experimental activities shall always be performed. Experimental work defined in the problem description shall be planed and risk assessed up-front and within 3 weeks after receiving the problem text. Any specific experimental activities which are not properly covered by the general risk assessment shall be particularly assessed before performing the experimental work. Risk assessments should be signed by the supervisor and copies shall be included in the appendix of the thesis.

The thesis should include the signed problem text, and be written as a research report with summary both in English and Norwegian, conclusion, literature references, table of contents, etc. During preparation of the text, the candidate should make efforts to create a well arranged and well written report. To ease the evaluation of the thesis, it is important to cross-reference text, tables and figures. For evaluation of the work a thorough discussion of results is appreciated.

The thesis shall be submitted electronically via DAIM, NTNU's system for Digital Archiving and Submission of Master's theses.

Contact persons:
Ivar Martinusen, IPM/NTNU
Christos Kalavrytinous, IPM/NTNU

[illegible]

Torgeir Welo
Head of Division

Ole Ivar Sivertsen

Ole Ivar Sivertsen
Professor/Supervisor

NTNU
Norges teknisk-naturvitenskapelige universitet
Institutt for produktutvikling og materialer

Preface

This thesis is submitted in partial fulfilment of the requirements for the degree of Master of Engineering and ICT. The degree is offered by the Engineering and ICT programme at the Norwegian University of Science and Technology in Trondheim.

My work was supervised by Professor Ole Ivar Sivertsen at the Department of Engineering Design and Materials, and co-supervised by Ivar Marthinusen and Christos Kalavrytinos in the same department.

Acknowledgements

To my supervisor, Ivar the Mountain Goat and Christos the Philosopher:
Thank you for your professional input and relentless interest in a flimsy coffee maker.

To my father, mother, sister and brother, in order of descending age, mind you:
Thank you for being an incredible asset to my life. May our bonds be everlasting.

To my grandparents, whom I never truly got to thank:
Your guidance and wisdom will always be remembered.

Epilogue

The remnants of war can be felt across the entire keyboard.

[illegible]

Eivind André Taftø
Trondheim
2016.02.29

Abstract

This study focuses on the possibility of using Knowledge Based Engineering (KBE) as a tool for the early creative design phase, using systematic design principles introduced by Eskild Tjalve. In addition, the concept of metacognition was explored to discover how it could relate to KBE and creativity. Specifically, the following research questions guided this research: 1) How can metacognition benefit KBE and creative design? 2) How can KBE technologies support the creative design phase? 3) How can Tjalve's principles be implemented with KBE for a simple product? 4) How can this environment be adapted to work for complex products?

This research study applies design research as a methodological approach. A literature review was conducted on previous research on KBE, creativity, and metacognition. In addition, this study has utilized Eskild Tjalve's work, Systematic Design of Industrial Products. The results of this review provided four learnings which became important for answering the main research questions. Firstly, existing research on KBE technologies indicated how the advantages of these systems could benefit creative design. Secondly, the research on creativity identified potential challenges with emulating human creativity, and revealed different ways of improving human creativity. Third, the study of metacognition delved into the arts of reflective practice and cognitive frameworks for creative design. Existing research suggested that computer systems could possibly embed some of these frameworks. Finally, Tjalve proposed a set of systematic product design principles that seemed feasible to implement using KBE technologies.

These theoretical results provided a knowledge base for the next phase of the study, and the practical part of this research utilizes Tjalve's principles to derive a cognitive framework for systematic design. This framework was then implemented in a pilot KBE environment for doing creative design work on a simple product. The environment was later evaluated to answer the following questions: a) *What are the systematic design constraints applied by the chosen framework?* b) *How do the framework constraints affect creativity?* c) *How feasible were the framework constraints to implement in KBE?*

Based on this evaluation, plans for prototype KBE environments were suggested for future development, where genericity and complex products were considered as advancements. In conjunction with this, HCI design principles were proposed to mitigate associated negative effects on creativity. A framework is proposed that illustrates how metacognition relates to KBE and creativity. In addition, positive and negative constraints for creative design are highlighted. Finally, implications for research and practice, along with limitations and suggestions, are correspondingly discussed in the thesis.

Sammendrag

Denne studien fokuserer på muligheten for å bruke *Knowledge Based Engineering* (KBE) som et verktøy for den tidlige kreative designfasen. I tillegg ble begrepet metakognisjon utforsket å finne ut hvordan dette kunne relateres til KBE og kreativitet. Forskningsspørsmålene som denne studien ønsker å svare på er: 1) *Hvordan kan metakognisjon være til fordel for KBE og kreativt design?* 2) *Hvordan kan KBE-teknologier støtte den kreative designfasen?* 3) *Hvordan kan Tjalves prinsipper implementeres med KBE for et enkelt produkt?* 4) *Hvordan kan dette verktøyet videre tilpasses for å arbeide med komplekse produkter?*

Denne studien benytter *design research* som metode. En litteraturstudie ble utført på tidligere forskning innen KBE, kreativitet og metakognisjon. I tillegg har dette studiet sett på Eskild Tjalves *Systematic Design of Industrial Products*. Resultatene fra dette litteraturstudiet ga fire lærdommer som ble essensielle for besvarelsen av forskningsspørsmålene. Først og fremst viser eksisterende forskning på KBE-teknologi indikerte hvordan fordelene med disse systemene kan være til nytte for kreativ design. Videre viser forskning på kreativitet identifiserte potensielle utfordringer ved å simulere menneskelig kreativitet, og forskjellige måter for å forbedre menneskelig kreativitet. Studiet på metakognisjon så nærmere på konseptet reflekterende praksis og kognitive rammeverk for kreativt design. Eksisterende forskning antydet at noen av disse rammeverkene kunne bygges inn i datasystemer. Eskild Tjalve foreslo et sett med systematiske designprinsipper for produkter, som virket mulig å implementere ved hjelp av KBE-teknologi.

Den teoretiske delen av studien gav resultater som bidro til å danne en viktig kunnskapsbase for neste fase av studien Den praktiske delen av masteroppgaven bygger på Tjalves arbeid vedrørende systematisk design av industriprodukter, hvorav et kognitivt rammeverk ble etablert. Dette rammeverket ble deretter implementert i en KBE testapplikasjon for å gjøre kreativt designarbeid på et enkelt produkt. Applikasjonen ble senere evaluert for å fastslå følgende: a) *Hva er de systematiske designrestriksjonene i det valgte rammeverket?* b) *Hvordan påvirker rammeverkets restriksjoner kreativitet?* c) *Hvor gjennomførbart var det å implementere rammeverkets restriksjoner i KBE?*

Basert på denne evalueringen, ble det foreslått KBE-prototyper for fremtidig utvikling, hvor det ble tatt hensyn til generiske egenskaper og støtte for komplekse produkter. Her foreslås også HCI design prinsipper for å nedtone negative effekter på kreativitet. Et rammeverk viser hvordan metakognisjon er relatert til KBE og kreativitet. Videre er positive og negative restriksjoner for kreativ design diskutert. Tilslutt diskuteres implikasjoner for forskning og praksis, begrensninger i studien, samt forslag til videre arbeid.

Contents

List of Figures

Notations

Abbreviations

AI	Artificial Intelligence
AML	Adaptive Modeling Language, by TechnoSoft Inc.
CAD	Computer Aided Design
CAI	Computer Aided Innovation
GUI	Graphical User Interface
HCI	Human Computer Interaction
KBE	Knowledge Based Engineering
KM	Knowledge Management
MMG	Multi Model Generation
OPM	Object Process Methodology
RQ	Research Question
SECI	Socialization, Externalization, Combination, Internalization

1 Introduction

1.1 Background and Motivation

Knowledge Based Engineering (KBE) technologies emerged in the 1980s, and have ever since provided invaluable automation of engineering design in various industries (Verhagen et al., 2012). The ever-increasing demand for product innovation is motivating the use of KBE and other computer aided technologies to do creative design (Cascini, 2008; Sanya and Shehab, 2014). The KBE environments of today are often mature technologies that automate product family design through parametrization and Multi Model Generation (Rocca, 2012). Although technologies that focus more on innovation do exist, the use of KBE for early creative design seems to be a less explored concept.

1.1.1 Research Motivation

The potential use of KBE in an early creative design context is the main motivator for our study. Additionally, the principles found in Systematic Design of Industrial Products by Tjalve (2003) serve as an inspiration and starting point for our research. The resulting scope has been expressed as an area of interest for the Department of Engineering Design and Materials at NTNU in Trondheim.

The main purpose of this research is to reveal how the advantages of KBE technologies can be leveraged to support creative design. In particular, we seek to adapt the principles of Tjalve (2003), and test-implement them in a pilot KBE environment. Having done this, we can evaluate how such a tool can affect design creativity.

An underlying assumption is that human cognition is closely tied to creativity (e.g. Arnsten et al., 2012; Goel, 2012; Howard et al., 2008), and that human cognition can have implications for the development and use of creative design tools. To explore this notion, we will study the field of metacognition[1], seeking to identify aspects that may relate to creativity and software engineering. Specifically, we will look for metacognitive concepts that can support designers to optimize their own creativity, and principles that can be implemented in a KBE environment. The inclusion of metacognition as a key part of our study is also motivated by literary works and otherwise, such as the podcast of Tim Ferriss (e.g. Ferriss, 2015), Waking Up (Harris, 2014), Thinking, Fast and Slow (Kahneman, 2013) and Marcus Aurelius: Meditations (e.g. Hays, 2002).

This study seeks to uncover potential challenges in the emulation or support of human creativity using computer technology. This leads us into a study of creativity in general. By better understanding its mechanisms, we aim to find ways in which people can improve their

[1] Metacognition is defined as knowledge and regulation of own cognition (elaborated in section 2.4).

own creativity. In addition, studies indicate that converting and embedding tacit knowledge is a challenge in any knowledge-based system (e.g. Alavi and Leidner, 2001; Dingsøyr, 2002; Nonaka and Konno, 1998; Rocca, 2012). With this in mind, we investigate key concepts within knowledge management to learn about such challenges within the scope of our research.

1.1.2 Research Contributions

New knowledge and understanding can further enlighten the possibilities of using KBE for early creative design. Moreover, the findings should provide some previously less explored connections between the fields of KBE, creativity, and metacognition. Lastly, an important aim and potential contribution is to develop the pilot KBE environment for creative design in a manner that can benefit or inspire future research and development.

1.2 Research Questions

The list below presents the main research questions addressed in this thesis. We elaborate on how we intend to approach these questions in *3.2*.

- RQ 1 How can metacognition benefit KBE and creative design?
- RQ 2 How can KBE technologies support the creative design phase?
- RQ 3 How can Tjalve's principles be implemented with KBE for a simple product?
- RQ 4 How can this environment be adapted to work for complex products?

1.2.1 Additional Research Scope

Since creativity is a broad concept that spans a variety of fields, it becomes necessary to apply a narrower scope that fits the focus and related aspects of the study. Referring to the research questions and the pilot KBE environment that is to be developed, *benefit* and *support* of the creative design phase are scoped by the following boundaries:

- The KBE pilot will implement a framework for systematic design that can affect the creative efforts of its users (e.g. Tjalve, 2003).
- The KBE pilot will *not* demonstrate creativity on its own.
- The KBE pilot *could* facilitate creativity by making non-creative tasks easier.
- The KBE pilot *could* support the users to focus and direct their creative efforts.

Having implemented the chosen framework for systematic design, the subsequent evaluation of the pilot KBE environment aims to answer the following:

a) *What are the systematic design constraints applied by the chosen framework?*
b) *How do the framework constraints affect creativity?*
c) *How feasible were the framework constraints to implement in KBE?*

1.3 Scenario

This scenario aims to contextualize the use of a KBE environment to do early creative design. However, the scenario will not be explicitly referenced or evaluated in this study.

Bobby is an engineer and product designer who works for a large corporation. Over the years, Bobby has been working with all kinds of computer-aided technology for product design. Nowadays, Bobby deals with parametrization and Multi Model Generation of complex industrial products.

All product designers in the corporation have received an invitation to participate in a renowned creative product design competition. The rules say that contestants are allowed to bring any tools to the competition, computational or otherwise, but that the product to be designed is not revealed in advance. On the day of the competition, contestants will receive a description of the product, along with its main components and functions. In addition, the product components will be supplied in the form of industry standard files for computer aided technologies. The contestant who can come up with the most creative design shall win.

Bobby's coworkers engage in vivid discussions of which product will be announced in the competition. Bobby is not so interested in these speculations. Bobby is an experienced developer of KBE environments, and has come up with a plan for how to win the competition. Using systematic design principles learned from a book by Eskild Tjalve, Bobby aims to develop a KBE environment that can assist in performing design variations on arbitrary products. This should allow Bobby to import the provided product definition files and quickly explore a large product solution space and choose the one that seems the most creative.

1.4 Structure of the Thesis

1.4.1 Chapter Overview

Chapter 2 presents the theoretical background for this study. *Chapter 3* explains the methodical approach for answering the research questions, and introduces the main software development tools that were used. *Chapter 4* provides details regarding the planning, development and evaluation of the pilot KBE environment that applies principles from Tjalve (2003). *Chapter 5* provides suggestions to the future development of a generic KBE environment for doing creative design work with arbitrary and complex products. *Chapter 6* provides a discussion of the research questions and an evaluation of the research study. *Chapter 7* presents the conclusion and implications of this study. Finally, *Chapter 8* provides suggestions to further research and development.

1.4.2 Chapter Coverage of Research Questions

A discussion of all research questions can be found in *Chapter 6*. The following list shows where relevant theory and results can be found for each research question:

- RQ 1 Theory: *Chapter 2* sections 2.1, 2.3, 2.4
- RQ 2 Theory: *Chapter 2* sections 2.1, 2.3
- RQ 3 Theory: *Chapter 2* sections 2.1, 2.3, 2.4, 2.5
  Results: *Chapter 4*
- RQ 4 Theory: *Chapter 2* sections 2.1, 2.3, 2.4, 2.5
  Results: *Chapter 5*

1.4.3 Assignment Focus and Adjustments

The research questions defined in *1.2* are selected and adapted from the original project proposal, which is the assignment text presented in the beginning of the thesis. This adaptation has been agreed upon as the revised assignment focus for this thesis. This implies that point six of the original assignment text will be presented as suggestions to future development in the same manner as point five.

2 Theory

This chapter introduces theoretical concepts that provide background information for discussions throughout the thesis. Some concepts are applicable to development of the KBE environment, or otherwise relevant for answering the research questions.

2.1 Knowledge Based Engineering

From the standpoint of an engineer or product designer, Knowledge Based Engineering (KBE) uses engineering knowledge embedded into computer systems to automate routine engineering design tasks (Rocca, 2012). This may include intelligent support that mimics the design process or simulates the presence of an engineer, providing advice, warnings, smart scaling and automated decisions.

As a consequence of these "smart" characteristics, KBE is labeled by some (Pinfold and Chapman, 2001; Rocca, 2012) as a merge between the disciplines of Artificial Intelligence (AI) and Computer Aided Design (CAD). Others mention that KBE has coexisted alongside CAD since the 1980s, both developing as individual entities (Verhagen et al., 2012).

Rocca (2012) points out that KBE is a less renowned discipline than for instance CAD, because it has mainly been used in a few competitive, non-mainstream industries, such as the aerospace and automotive industries. Aker Solutions[2] use their KBeDesign[3] framework to develop offshore oilrigs and platforms.

2.1.1 Advantages of KBE

The main advantage of KBE lies in its ability to automate routine engineering tasks, saving product development time and labor costs (Rocca, 2012). The ideas behind Knowledge Management (KM) approaches (e.g. Alavi and Leidner, 2001; Davenport and Prusak, 1998; Dingsøyr, 2002), is not only a central part of how KBE systems function, but these native KM capabilities are in themselves valuable for a knowledge dependent organization. Rocca (2012) suggests that one of the strengths of KBE lies in its programming language. He highlights how a language is advantageous, or even necessary, to perform or implement certain features, such as the capturing of design processes, defining consistent generative models, and the flexibility to do custom interoperability (Rocca, 2012, p. 173).

[2] Aker Solutions: http://akersolutions.com/
[3] KBeDesign: http://citrix.akersolutions.com/en/Global-menu/Media/Feature-stories/Engineering/KBeDesign/

2.1.2 KBE for Creative Design

Some features of KBE environments could be seen as native abilities to support design creativity. It has been found that 80% of engineering design consist of repetitive routine tasks, whereas only 20% of their efforts are innovative in nature (Skarka, 2007). Given the time savings that KBE systems provide, it should be possible to shift this balance in favor of creativity (Sanya and Shehab, 2014). Designers using KBE have more freedom to explore the product solution space (Verhagen et al., 2012). This could for instance be done by changing product parameters, or generating multiple models from a general set of rules. This is the case for generative systems, for instance, which do not contain specific geometrical instances, but defines generic rules to enable fast generation of new design instances (Verhagen et al., 2012).

2.1.3 Knowledge Modelling Frameworks for KBE

Two common frameworks (Sanya and Shehab, 2014) for embedding engineering knowledge are the Common Knowledge Acquisition and Design Support (CommonKADS), and the Knowledge Based Engineering Application methodology (MOKA). CommonKADS is a structured and generic-purpose framework, whereas MOKA focuses more specifically on the capture of engineering knowledge related to product design.

2.2 Essentials of Knowledge Management

This section contains supplementary information related to the knowledge management capabilities of KBE, and provides some perspectives for the following section on creativity. However, the findings in this section were not explicitly used in the pilot KBE development.

Knowledge Management (KM) can be defined as strategies and techniques that improve the transfer and use of knowledge in an organization (Dingsøyr, 2002). KM is a central part in the development and use of KBE systems. The most important aspects include identifying, capturing, converting, and finally embedding knowledge into computer systems for effective reuse (Alavi and Leidner, 2001; Rocca, 2012). Different types of knowledge, however, offer different challenges in a KBE context.

2.2.1 Types of Knowledge

Traditionally we distinguish between tacit and explicit knowledge, which likely originates from Gilbert Ryle (1949). Ryle further defined explicit knowledge as a theoretical *know-what*, while tacit knowledge is more of a practical *know-how*. Others (Nonaka and Konno, 1998) claim that knowledge is primarily tacit, which stems from Japanese philosophy, explaining that a focus on explicit knowledge has been more popular in western society.

Tacit knowledge is a complex and intangible concept, which makes it difficult for computer systems to grasp and formalize (Alavi and Leidner, 2001). For the same reason, tacit knowledge is hard to understand and define. Dingsøyr (2002) describes tacit knowledge as something that humans are unable to express, yet that guides their behavior. An everyday example is our innate human ability to recognize one face in a crowd, or to swallow a sip of water. At another level, tacit knowledge could be skills related to personal experiences or awareness of context (Polanyi, 1966). In any case, tacit knowledge is not only hard for humans to communicate, but is also difficult to encode into knowledge embedding technologies such as KBE.

Knowledge *within* a KBE system has by some scholars been divided into categories of facts, procedures, judgments, and control (Calkins et al., 2000). Of these, judgments are perhaps the most closely related to human tacit knowledge, since judgment is a cognitive skill that involves experience, observations, and reasoning. Control knowledge is meta-knowledge that manages the other three through pattern directed actions, anticipating developments and dealing with uncertainties.

2.2.2 Knowledge Conversion

The Concept of *Ba*

The Japanese philosopher Kitaro Nishida defined the concept of *ba* as a shared space for emerging relationships (Nonaka and Konno, 1998). Nonaka and Konno explain that knowledge exists within *ba* as a shared resource. It is attained from *ba* when we reflect upon personal experiences, or the experiences of others. For knowledge to be separated from *ba*, it must first be converted to information. They describe this information as explicit, tangible knowledge. With the traditional western view (Ryle, 1949), we can assume that it is only in this explicit form that knowledge can be incorporated into for instance KBE systems.

The SECI Process

The well-known SECI process introduced by Nonaka et al. (2000) and also discussed by Dingsøyr (2002), is a way to describe how knowledge is converted between tacit and explicit modes of knowledge in a circular pattern (see Figure 1). Each step can be summarized as follows (Nonaka et al., 2000):

- Socialization	Tacit knowledge transfers between people by shared experiences.
- Externalization	Tacit knowledge articulates into explicit sharable knowledge.
- Combination	Explicit knowledge combines to create new knowledge.
- Internalization	Explicit knowledge is embodied by people as tacit knowledge.

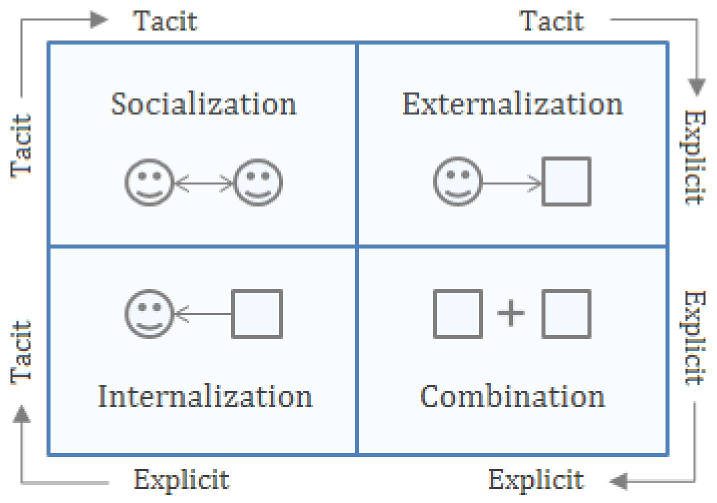


Figure 1: SECI process for knowledge conversion (adapted from Nonaka et al., 2000)

The SECI process has become popular since it explains an otherwise hard to define knowledge creation process. It has also been criticized because it takes a taxonomic perspective on the concept of knowledge (Walsham, 2001). In this study, however, we seek to embed relevant knowledge into a KBE environment which creative designers can then draw benefit from. Within this scope, externalization and internalization of knowledge are perhaps the most interesting phases of knowledge conversion.

2.3 Creativity

In psychology, creativity is typically categorized into the creative process, product, person, and environment (Howard et al., 2008). Robert Franken (1994) states that creativity allows us to generate ideas or recognize possibilities that aid in problem solving and original creation. This last definition seems to be a suitable scope for engineering and creative design.

2.3.1 Innovation and Creative Design

Howard et al. (2008) describe innovation as a black box processing of design information to produce creative output. This may relate to the notion that it is hard to define exactly how our creative outputs are formed. Their study found that psychologists are split between romantic and non-romantic views, where romantics regard creativity as mysterious and subconscious, while non-romantics see it as a linear sequence of steps. We believe that the non-romantic view may be more directly applicable to the engineering design process, but that there are also ways in which we can optimize our subconscious creative powers.

The Challenge of Abduction

Creative design is regarded by some as a form of abduction (e.g. Dorst, 2011; Gero, 2013). We can understand abduction by considering deduction and induction from traditional science (see Dorst, 2011). In short, deduction lets us predict *results*, and induction lets us hypothesize *how* something works. With abduction, we imagine a desired *result* and know a means for *how* it can be done, but we do not yet know *what* can achieve this. This is typically what product designers and engineers do (Dorst, 2011). A second level of abduction is when both the *what* and the *how* are not yet known (see Figure 2). This effectively gives us an equation with two unknowns that need to be found in parallel (Dorst, 2011).

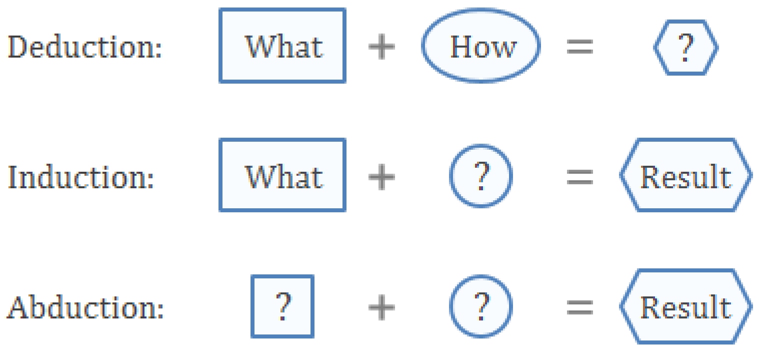


Figure 2: Deduction, induction and abduction (adapted from Dorst, 2011)

Gero (2013) argues that design is not like running deduction backwards. He explains that there is no single solution to a set of requirements. In other words, there is an entire solution space of *what* and *how* that can achieve one desired *result*. This philosophy may have implications for the design of computer systems that support or replicate creativity.

2.3.2 Computational Creativity

Computational Creativity is the art of using computer technology to either replicate human creativity, or augment the creativity of a human using such tools (Colton and Wiggins, 2012). This can be considered as a subfield of Artificial Intelligence (AI), which also has close ties to human cognition and philosophy (Colton and Wiggins, 2012).

Within the realm of CAx technologies, Computer Aided Innovation (CAI) is an emerging field of study (Cascini, 2008). CAI supports creative design by aiding the user in generating and evaluating new product variants. This includes the use of genetic algorithms along with traditional CAD tools. It seems that CAI has been researched and developed for over a decade (Cascini, 2008). Both innovative types of KBE systems and CAI technologies could be categorized as specialized tools for Computational Creativity.

2.3.3 How Knowledge Affects Creativity

Asimov (1959) claimed that a key criterion for optimizing team creativity is to ensure that each member were experts in their own fields. In such a case, their combined creative efforts would also depend on their ability to convey and share knowledge. This is closely related to shared knowledge spaces, or the concept of *ba* (Nonaka and Konno, 1998, p. 41):

> *"To participate in a ba means to get involved and transcend one's own limited perspective or boundary. This exploration is necessary in order to profit from the "magic synthesis" of rationality and intuition that produces creativity."*

This quotation supports that creativity stems from tacit knowledge and that individual creativity may flourish by immersing oneself into these shared knowledge spaces.

2.3.4 Fostering Creativity

Törnkvist (1998) proposes that in engineering education, there is a tendency towards "over-scientification", which discourages divergent, creative thinking (see also Wagner and Compton, 2012). Törnkvist also mentions a theory of situated cognition, where creativity cannot be taught out of its knowledge context or as a special skill, but rather needs some form of apprenticeship. In addition, he found that decisions made by design engineers were seldom based on the scientific calculations they learned as students.

In the study by Törnkvist (1998), many students reported that their best creative sessions were not driven by need or usefulness, but rather from curiosity and playfulness. He brings up the case of Richard Feynman, a former student at Cornell University, who in a state of depression started tossing cafeteria plates in the air. Feynman noticed that the plates were rotating twice as fast as they were wobbling (Törnkvist, 1998, p. 7):

"It was effortless. It was easy to play with these things. It was like uncorking a bottle; everything flowed out effortlessly. I almost tried to resist it! There was no importance to what I was doing, but ultimately there was. (...)"

Feynman did later receive the Nobel Prize in physics for his Feynman diagrams, which he attributes to this playful and obsessive research on wobbling plates (Törnkvist, 1998).

A study by Epstein et al. (2008) found that the creativity of individuals could be improved by training in certain related skills. These include idea capturing, challenging oneself with difficult tasks, seeking new knowledge, and seeking out new stimuli from the surroundings.

These findings could indicate that there is a need for greater focus on creativity in engineering education, and that teaching creativity per se comes with certain challenges. Epstein et al. (2008) suggest teaching supplementary skills to support creativity, and Törnkvist (1998) argues that creativity can mature during apprenticeship or flourish through playful curiosity.

2.3.5 Creativity as a Paradox

Creativity is paradoxical in the sense that it changes our world, and at the same time this world changes us. According to the field of ontological design (Willis, 2006), we design new technology which in turn designs us back. For instance, when people accept a new technology, it becomes a part of their social structure. The interaction between users and technology is mutual. This means that a user adapts to the technology, but also the other way around (DeSanctis and Poole, 1994). This seems true for inventions like the smart phone, and soon virtual reality. KBE environments that support creativity could fall into the same category.

Former research studies have revealed that many educational institutions actually remove creativity (Törnkvist, 1998; Wagner and Compton, 2012). At the same time, we require creative people in order to change how these institutions work. The people we seek are those that can think outside of established systems, and likely someone who is unconventional in his habits (Asimov, 1959).

2.3.6 Creative Processes

Creative Processes of Different People

Creativity applies to a wide range of professions, and different people have their own processes during which they are the most creative. Identifying some commonalities may be helpful for optimizing creative design processes, both in humans and computer systems.

"For me, creativity is a process of removing barriers, not of pulling something that's outside of me. The analogy I use is flow rate. If I'm busy, I think about lots of other things, but if I clear my mind, I can't stop the ideas from coming."

Paraphrase of Scott Adams, creator of Dilbert (Ferriss, 2015, ep 82)

"Discipline equals freedom. We have standard operating procedures for everything. You'd think this would restrain creativity, but it doesn't. When I give my team a task, I know what parameters they'll stay within. This makes things easier, and gives us freedom both on an individual level and as a group."

Paraphrase of Jocko Willink, Navy SEAL officer (Ferriss, 2015, ep 83)

"You should work in a casual environment like a living room or café, not in an office. It makes work not feel like work. An environment that feels safe, social and casual lets us pitch ideas that seem crazy. Nobody is judging us, measuring time or quotas. It allows us to be free creatively."

Paraphrase of Seth Rogen, filmmaker and comedian (Ferriss, 2015, ep 84)

For Scott Adams (Ferriss, 2015, ep 82), creativity is about maintaining flow and removing barriers. On the other hand, Jocko Willink (Ferriss, 2015, ep 83) prefers discipline and positive constraints to remove the burden of many small decisions and concerns. Finally, Seth Rogen (Ferriss, 2015, ep 84) benefits from being in a safe and casual environment, letting ideas come freely without judgment. This last notion resonates well with the belief that pure creativity requires isolation to avoid the embarrassment of bad ideas (Asimov, 1959). Although these people have different preferences, their end result is the same. They leave the creative mind free to focus on what is important. Even if they are not engineers or product designers, the principles behind their creative processes may still be applicable.

Night-Waking

In a study by Ekirch (2006), he argues that in the old times, before the invention of electricity, humans would sleep in two phases. This segmented sleep was divided by a period of night-waking, with nocturnal activities that people were too tired for at the time of "first sleep". Ekirch claims that during this period, our minds have dreamlike characteristics, which could be similar to how our brains function during optimal creative flow or subconscious creativity (Howard et al., 2008). The Norwegian novelist Knut Hamsun adapted segmented sleep to do creative work, and Francis Quarles explained how this period was optimal for concentration:

"Let the end of thy first sleep raise thee from thy repose: then hath thy body the best temper, then hath thy soule the least incumbrance; then no noise shall disturbe thine ear; no object shall divert thine eye."

Francis Quarles, 17th-centruty English poet

At night, activity in the frontal lobe of our brain is reduced, leading to less internal second-guessing of the ideas that emerge (Arnsten et al., 2012). Some people evaluate their ideas on the following day, and say that they often seem abstract or unreasonable. In the field of engineering, we require that ideas are at least scientifically possible. However, unfeasible ideas can sometimes be adapted to work, or they can serve as further inspiration.

2.4 Metacognition

Metacognition can be defined as higher order thinking that involves knowledge and regulation of own cognition (e.g. Brown, 1987; Schön, 1983; Schraw, 1998). It is also referred to as “thinking about thinking” or “cognition about cognition”. Metacognitive skills are considered to be directly teachable (Schraw, 1998), in contrast to for instance creativity, which seems to be more challenging in this regard (e.g. Epstein et al., 2008; Törnkvist, 1998).

A practitioner of metacognition could for instance monitor their own thoughts, evaluate current work performance, be aware of their own learning, or reflect upon past experiences. In most personal cases, these techniques are used to achieve some form of self-improvement. One could for instance become a more effective learner, find optimal work routines, or convert experiences into applicable knowledge.

> *“Whenever you find yourself on the side of the majority,*
> *it is time to pause and reflect.”*

Mark Twain, 1904[4]

2.4.1 Reflective Practice

Marcus Aurelius, Roman Emperor from 161 to 180 AD, made a series of personal writings that were later combined into the book, Meditations (e.g. Hays, 2002). His work demonstrates an early example of reflective practice with roots in stoic philosophy. Aurelius improved his personal and professional conduct by reflecting upon his philosophy apprenticeship and his continuous experience as emperor.

Also today, people who desire professional development can do so by reflecting upon their own professional experiences (Yanow and Tsoukas, 2009). We can evaluate and appraise our task performances and results in order to find better ways of doing them.

Self-Monitoring of Task Performance

The human mind has a naturally limited attention span, which affects our ability to retain optimal task performance over long periods of time. Aurelius encourages:

> *“Concentrate every minute (…) on doing what’s in front of you with precise and genuine seriousness, tenderly, willingly, with justice. And on freeing yourself from all other distractions (…), and stop being aimless (...)”*

(Hays, 2002, bk. 2 section 5)

Achieving this level of concentration requires that we become observers of our own minds. As the mind begins to wander, we can bring it back to focus on the task at hand. Distractions may come from internal or external stimuli, and should be handled before starting a task. If

[4] Mark Twain quote: http://twainquotes.com/Majority.html

we discover that our task performance is poor despite these attempts, it may actually be more productive to continue the task at another time or place. Our goal is to maintain motivation and sustained effort over time, so that we may efficiently and effectively see a task to its completion.

Metacognitive Strategies

Regulation checklists can be used to aid the use of metacognition (Schraw, 1998). These mental frameworks are helpful to ensure that we have covered all the aspects that we wish to monitor. Over time, a regulation checklist may become second nature, demanding less active monitoring. We can also imagine less abstract frameworks that are more directly applicable as professional work routines, such as the mandatory preflight checks done by pilots.

2.4.2 Cognitive Frameworks

We here define a cognitive framework to include thought systems and modeling techniques that can represent the real world or abstract concepts. The aim is to find frameworks that can contribute to improve design creativity, both in humans and computer systems.

Fast and Slow Thinking

Daniel Kahneman (2013) introduces a separation of the mind into fast and slow systems. He labels them *System I* and *System II*. *System I* takes care of fast and intuitive thinking, most of which happens without any explicit effort from our side. *System II* represents the more engaged thinking that is required when System I cannot automatically solve the problem.

Kahneman (2013) discovered that when a person meets some form of resistance in an otherwise intuitive task, the mind would shift from using *System I* to *System II* in order to solve the problem. This is also referred to as the "eyebrow response", which comes from the strict facial expression that humans often make when met with difficult or unordinary tasks.

System I is the part of our mind that is most active during states of creative flow. If *System I* is interrupted, for instance by a sudden "eyebrow response", the creative flow will likely suffer. This could have implications for the development of KBE environments that aim to support creativity.

Systems Thinking

Goel (2012) defines systems thinking as regarding systems in terms of their components and how they interact through processes. He claims that design is naturally done with systems in mind. Goel explains that systems thinking is challenging for humans when it involves a large number of components and processes. In such cases, the limitations of the human mind will inhibit design creativity. To better cope with this, the Structure-Behavior-Function (SBF) modeling technique was created (Goel et al., 2009). SBF uses various mental abstractions to divide a system into subsystems. The framework is supposed to aid the analysis and organization of a system, and provide a vocabulary for describing it.

Others have also mentioned structure, behavior and function as key aspects in design processes (e.g. Gero, 2013; Howard et al., 2008). Howard et al. (2008) propose that an SBF framework can be useful for integrating engineering design with the general creative process.

Dori and Crawley (2013) have developed the Object Process Methodology (OPM) since 1995, which uses objects, processes and states as modeling entities (see Figure 3). OPM also considers structure, behavior and form as complementary aspects from which systems can be viewed. They argue that the strength of OPM resides in its few and simple entities, while at the same time being capable of modeling virtually anything. They also claim that OPM can play a role in converting tacit knowledge into explicit knowledge, and that it is good for idea generation and rapid prototyping in the early design phases.

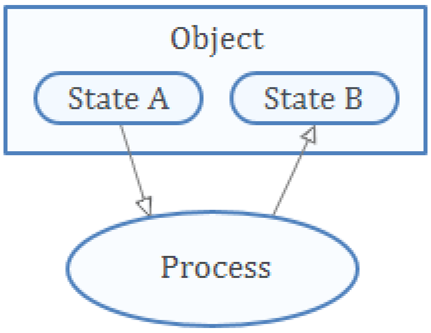


Figure 3: The basic entities of OPM (adapted from Dori and Crawley, 2013)

Judging from what OPM and SBF have to offer, it seems that they would both be suitable system modeling frameworks in the scope of creative design.

Analogical Thinking

Analogical thinking is regarded as fundamental in some aspects of creative design (Cross, 1997). We can transfer knowledge from one case and adapt it to another by using analogical thinking (Goel, 2012). In other words, we can learn from existing solutions and apply them by analogy to new problems. This could resemble lower level abduction where some general working principles, i.e. the *how*, are already known (Dorst, 2011). Goel (2012)states that this is cognitively challenging since we must conjure a representation of past solutions, and use pattern recognition to adapt the source knowledge to the target problem. KBE systems with powerful knowledge management capabilities could support or replicate these creative techniques.

Meta Thinking

Goel (2012) introduces meta thinking for creative design, and defines:

> *"Meta thinking entails processes such as goal spawning, suspension, and abandonment; strategy selection; belief revision; self-explanation; and design, diagnosis and revision of reasoning processes."*
>
> (Goel, 2012, p. 4)

He claims that creative design is not just about the design process itself, but also about meta design of this process. This means that we design the design process to optimize creative output. He points out that there is still little research done on metacognition and meta design.

Goel (2012) took part in the development of creative design frameworks that apply meta thinking. These include modifying own design processes that have led to a failure, adapting design processes to new tasks, and performing self-diagnostics and repair of defective knowledge upon failures. He further states that these frameworks are neither human nor computer specific. We can draw a parallel to KBE systems that rely on advanced levels of control knowledge to adjust its processes (Calkins et al., 2000).

Systematic Design of Industrial Products

Tjalve (2003) introduces a systematic and general approach to product design. We derive a cognitive framework from his principles in chapter 2.5. Although there is no mention of metacognition throughout his book, he is likely to have used some form of metacognitive practice to devise his systematic design methods. Tjalve provides a solid and generic framework. However, within the scope of KBE and creative design, it is possible that these methods can be tried and reflected upon to provide an even better framework for use within modern technology.

2.4.3 Cognition in Human Computer Interaction

The field of Human Computer Interaction (HCI) involves the design of interfaces through which humans can interact with computer technology (e.g. Preece, 1995). Because of the human aspect, HCI has close ties with human cognition. Through an understanding of human nature, we can design user interfaces that better pertain to human expectations.

Don Norman (2013) provides a deep analysis of the cognitive aspects of designing everyday things, many of which are directly transferrable to computer interface design. He argues that there is no such thing as human error, only bad design. For relatively simple cases, if an interface *requires* a user manual to be understood, there is something inherently wrong with the design.

User Interface Design for Creative Work

Norman (2013) explains how signifiers and constraints can be used for early error prevention to reduce user frustrations. We can relate these frustrations to the negative "eyebrow response" introduced by Kahneman (2013), and how it breaks creative flow. It seems that basic HCI design principles are essential for a system that facilitates creative work, by making sure that user creativity is not hindered by a poor interface.

A well designed interface can even offer a joyful user experience (Norman, 2013). This may contribute to the sense of playfulness that drives our native human creativity (Törnkvist, 1998).

User Awareness of Interface Design

Norman (2013) would argue that mainly the developers of a user interface are responsible for its user friendliness. However, if circumstances allow it, the user can give feedback to the interface developers so that they may improve it. This is a natural part of iterative development processes (e.g. Larman, 2004). The user would engage in reflective practice to determine whether the interface allows them to perform optimally, and help pinpoint any problematic areas.

2.5 Systematic Design of Industrial Products

This section is a summary of Systematic Design of Industrial Products by Tjalve (2003). Any reference to *Tjalve* herein implicitly refers to the works of Tjalve (2003). The sketch figures in this section are adapted from Tjalve's work.

Tjalve presents a set of principles for systematic product design. Of particular interest are methods that deal with defining requirements, assigning restrictions, and systematically exploring the solution space of a product.

2.5.1 Terminology

Some basic terminology should be established before discussing Tjalve's theories:

- Product	The product as a whole.
- Element	A single part or component of the product.
- Structure	The elements and their relationship.
- Form	The shape and dimensions of an element.

2.5.2 Product Properties

Before a product is designed, we have a set of *desired properties* which will influence our decisions during the design process. Upon completion, we have a set of *realized properties*. According to Tjalve, there are five *basic properties* that together completely define a product: structure, form, material, dimension and surface. The design process is about manipulating these five basic properties to get from desired to realized properties (see Figure 4).

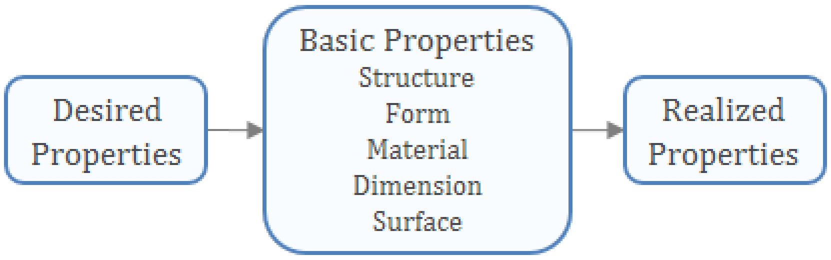


Figure 4: Manipulating basic properties (adapted from Tjalve, 2003)

2.5.3 Product Synthesis

Tjalve's *product synthesis* is his entire product design process divided into systematic steps (see Figure 5). An initial problem analysis establishes desired properties which act as criteria that affect decisions in all steps. In addition, the main functions are derived, and further divided into sub-functions. There can be many ways to realize the sub-functions, where each possibility gives a unique *basic structure*. At this point, we have decided *how* the product will perform its functions, so any alterations beyond this point can only be with structure or form.

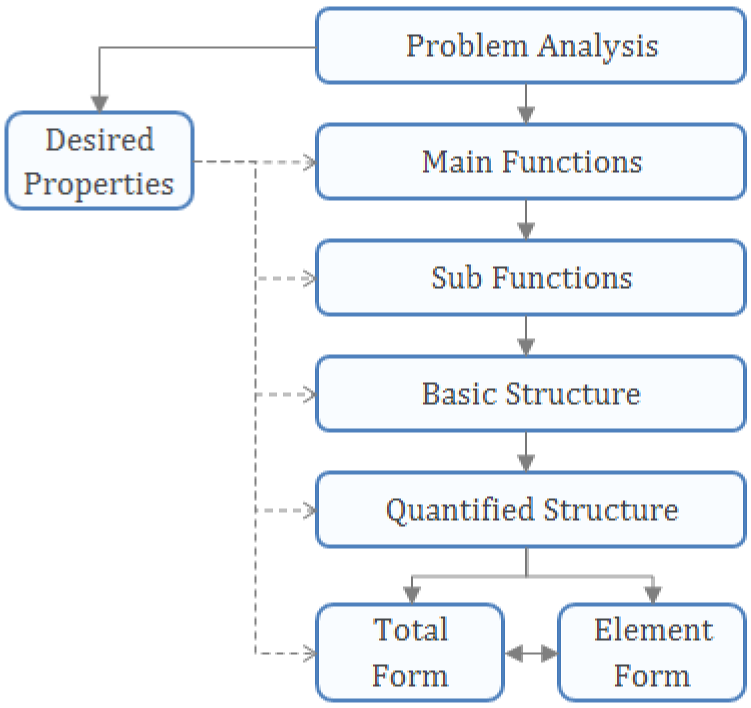


Figure 5: Product synthesis stages (adapted from Tjalve, 2003)

2.5.4 Functions and Basic Structure

When the product functions and basic structure are decided, we already have a specific type of product, fixed within a certain scope of possible mutations. It becomes impossible to change the entire product gradually in order to get another type of product.

Two products can also have the same functions, but different basic structures (see Figure 6). This is achieved by exploring different means of fulfilling the sub-functions. The basic structure of each product can be radically different, as long as the resulting main function is the same.

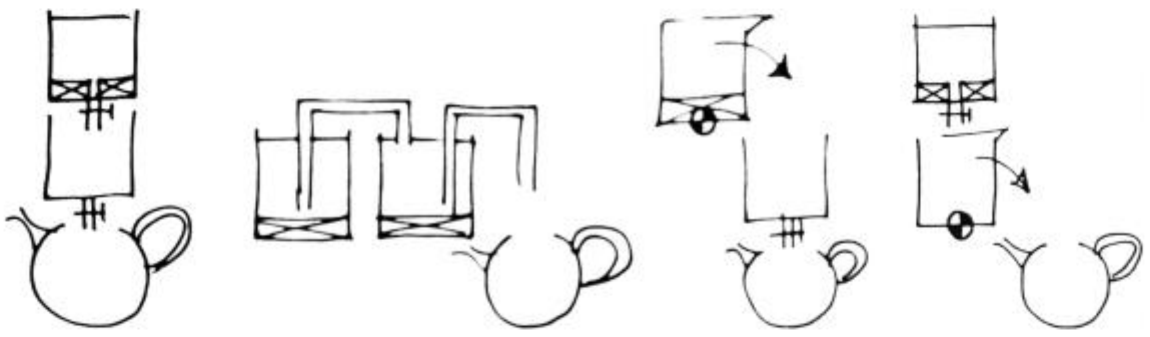


Figure 6: Different basic structures of a tea maker (adapted from Tjalve, 2003)

2.5.5 Quantified Structure

In this stage we look at the basic structure and explore different ways of arranging the elements (see Figure 7). The number of each element can also be changed (see Figure 8). Tjalve calls this the *structure variation* method. It can be viewed as exploring all possible statistical combinations. The desired properties and some common sense will help to limit the solution space. For instance, it is obvious that the wheels of a steam roller cannot be placed on its roof.

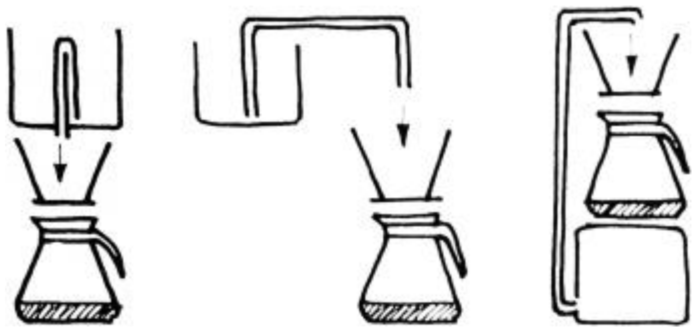


Figure 7: Quantified structure of a coffee maker (adapted from Tjalve, 2003)

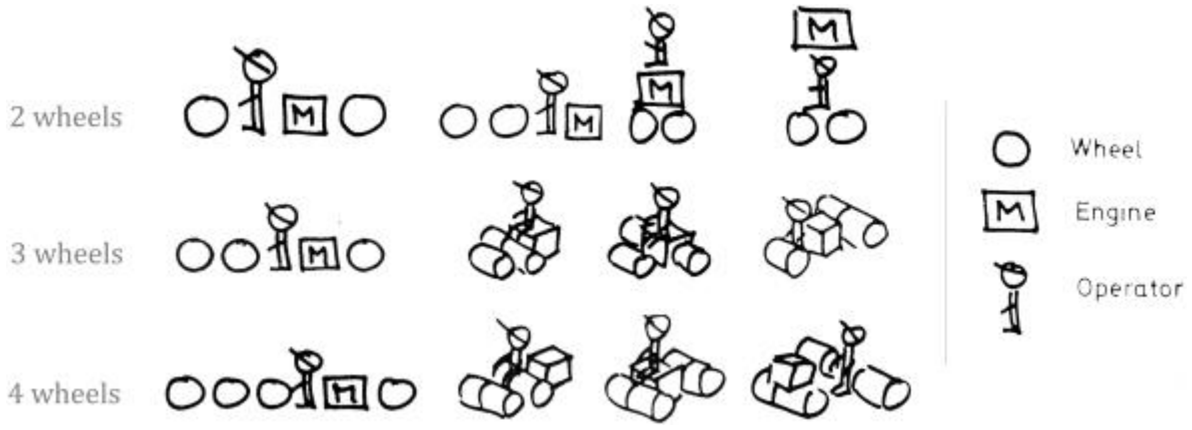


Figure 8: Quantified structure of a steam roller (adapted from Tjalve, 2003)

Isaac Asimov claimed that great ideas were made when connecting two things which might not ordinarily seem connected (Asimov, 1959). Systematically using Tjalve's structure variation method allows us to generate a large solution space. This might be one way of discovering such unforeseen connections without having to actually think of them.

2.5.6 Total and Element Form

The last step of the product synthesis is what Tjalve refers to as *form variation*. This means changing shapes, materials, dimensions and surfaces (see Figure 9). He further distinguishes between total form and form of the elements. The total form is considered when the product form as a whole is important, for example the aesthetics of a car. When technical or economic criteria dominate, the design of each element is considered. The total form depends on the form of the elements, and vice versa.

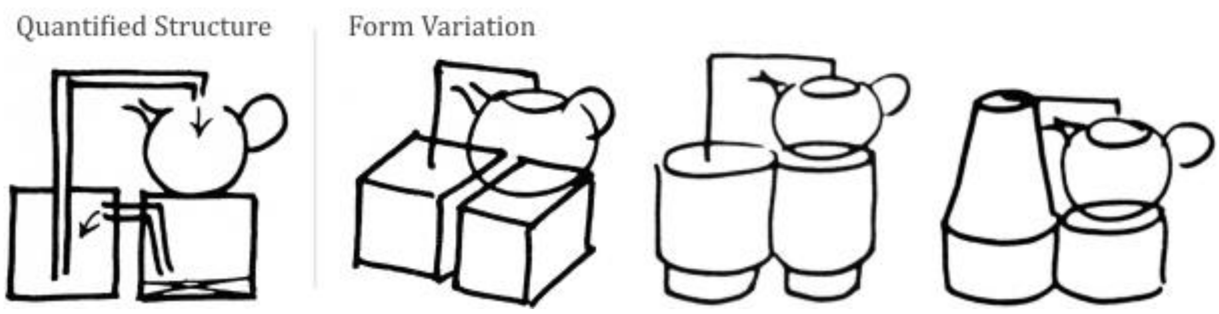


Figure 9: Form variation for a tea maker (adapted from Tjalve, 2003)

2.5.7 Functional Surfaces

The functional surface is a part of an element that has an active function during use. Internal surfaces relate to other elements in the product, while external surfaces have active functions to the surroundings. In form variation, functional surfaces let us define the most important surfaces and design the rest of the element with these in mind (see Figure 10). We can also modify the surfaces themselves by changing their number, arrangement, geometry and dimensions.

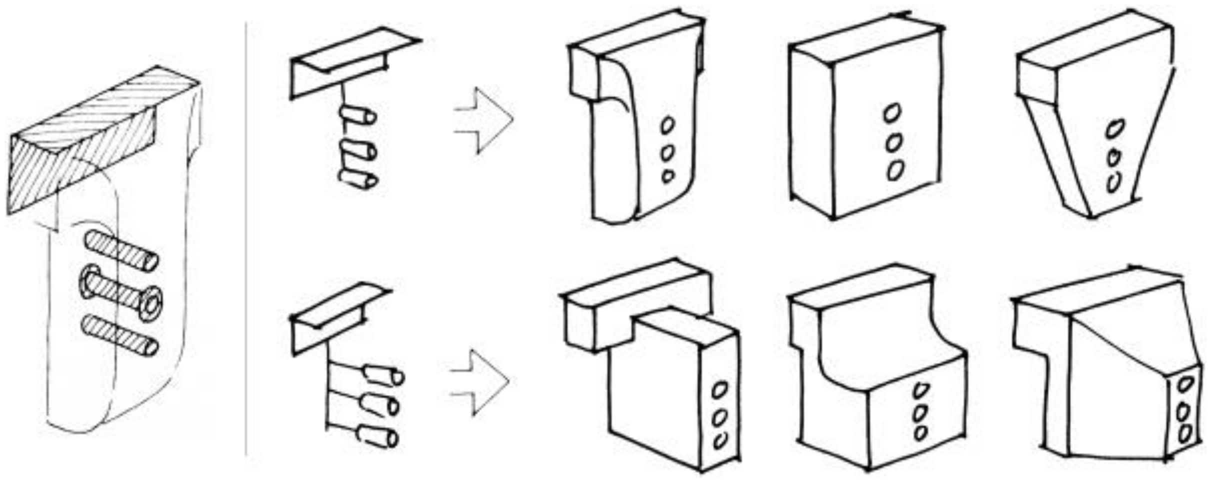


Figure 10: Functional surfaces and form variation (adapted from Tjalve, 2003)

Closely tied to form variation is the concept of minimum and maximum functional surfaces. This means creating a surface that is as small or large as possible while keeping its intended function. We can use this to optimize for certain properties, for instance maximizing adhesion between elements, or minimizing the use of materials (see Figure 11).

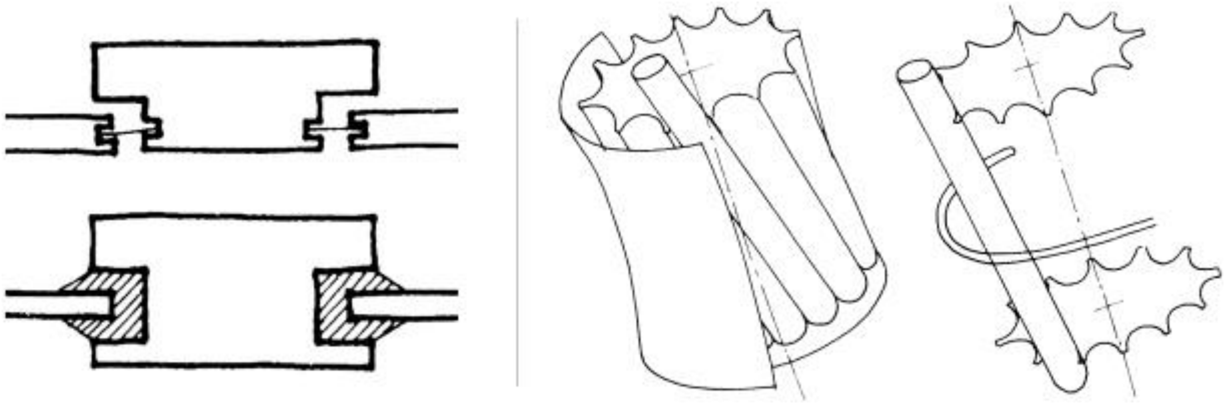


Figure 11: Maximum and minimum functional surfaces (adapted from Tjalve, 2003)

2.5.8 Summary

Systematic Design of Industrial Products (Tjalve, 2003) provides a framework that allows us to do systematic and precise design variations to a wide range of products. His techniques become increasingly more interesting with a creative KBE environment in mind. We have expanded our basic terminology with some concepts that may be relevant to this:

- Basic Structure	Product scope after functions and sub-functions are decided.
- Structure Variation	Explore all possible relative arrangements of elements.
- Form Variation	Explore different element shapes, dimensions and surfaces.
- Functional Surface	Part of an element which has an active function during use.

3 Methodology

This chapter explains our overall methodical approach to answering the research questions, and introduces the tools that were used for software planning and development.

3.1 Design Research Approach

Design science stems from the engineering tradition as an approach to problem solving (Hevner et al., 2004; Sein et al., 2011). The typical stages in design research approaches are 1) problem formulation, 2) iterative development and evaluation, 3) reflection and learning, and 4) formalization of learning (Sein et al., 2011).

Design research describes the process involved in designing the artifact (for instance a pilot environment), and the product itself (the artifact). Artifacts can be software-intensive systems that consist of terminology, models, methods, and system instances. KBE technologies represent typical software-intensive systems and the artifact in this case is a pilot KBE environment acting as a proof of concept. The artifact will embed systematic product design principles by Tjalve (2003) to facilitate creative design work. We describe the process and choices involved when implementing creativity into design. This involves planning, development, and evaluation of the pilot. From this proof of concept, we develop prototype interfaces as suggestions for future work.

In addition, the description of the artifact shows the relation between the KBE environment, creativity and metacognition. Thus, it represents an ensemble that goes beyond technological dimensions, and researchers emphasize that several aspects such as contextual factors, structure and goals, might be inscribed in the artifact (Sein et al. 2011). In case of this study, the artifact also includes theoretical assumptions of cognitive frameworks, metacognition, creativity, knowledge management and reflective practice. This knowledge base provided a fundament for developing and evaluation of the artifact, or pilot. It also includes research inspired from the practical use of Tjalve's (2003) principles.

Finally, the design process provides us with possibilities to attain learnings that can be formalized. For instance, this study proposes a framework that shows the relations between the KBE environment, creativity and metacognition. In addition, the study has also identified positive and negative constraints as an important outcome.

A limitation of this research study is the lack of iterative loops of development, feedback and evaluation. However, a discussion is provided about how future work should focus on participating design to enable iterative loops and necessary feedback from the end users.

3.2 Approach to Research Questions

RQ 1: How can metacognition benefit KBE and creative design?

First, we study the field of metacognition to identify which aspects can be relevant in the context of KBE and creativity. Secondly, we separate these findings into principles that can be *implemented* in a KBE environment, and concepts that the *users* can benefit from.

The users are engineers and product designers. We seek methods that with a reasonable amount of effort can be applied in a professional work space, for instance through training or change of environment. This could be work methods, awareness of own creative processes, and types of environments where people tend to be most creative.

As for the KBE implementation, we look for mental frameworks that can guide or optimize the creative design process, and software design principles that are derived from knowledge of human cognition.

RQ 2: How can KBE technologies support the creative design phase?

We suspect that some of the advantages that KBE has to offer can be leveraged to facilitate creative design. First, we look further into existing research on KBE to learn about these advantages. Secondly, we study the works of Tjalve (2003) to find systematic product design principles that can be implemented in a KBE environment.

RQ 3: How can Tjalve's principles be implemented with KBE for a simple product?

We will choose a simple example product, and develop a pilot KBE environment for doing creative design work with this product. Our main focus will be on the principles introduced by Tjalve (2003), and the choice of example product will be closely tied to which principles we want to test. The pilot will be specific to this product, and therefore non-generic. The overall purpose is to get an indication of how our findings from RQ 1 and RQ 2 can support creativity, and to what degree they are feasible to implement in a KBE environment.

RQ 4: How can this environment be adapted to work for complex products?

First, we will test and evaluate our pilot KBE environment developed for RQ 3. Secondly, we attempt to organize what we have learned from RQ 1, RQ 2 and RQ 3, in order to provide suggestions for future development. We will suggest features and prototype interfaces for a generic KBE environment for doing creative design work with arbitrary, complex products.

3.3 Development Tools

3.3.1 Adaptive Modeling Language

Adaptive Modeling Language[5] (AML) is a KBE modeling framework provided by TechnoSoft[6]. The development environment is geometry-centric, and the programming language is object-oriented with a syntax that resembles Lisp[7]. TechnoSoft provides their AML Reference Manual for looking up available classes and functions.

NTNU has a collaboration with Aker Solutions[8], where AML is central. Aker Solutions have over time developed their KBeDesign™ framework from an extensive track record of products[9]. This partnership made AML a natural choice for developing my KBE application.

With the installation of AML comes a customized XEmacs text editor, which is the main tool for writing AML code. From here the developer can run the AML environment and load models using the provided console window. There is also a Main Modeling Form, which already contains tools to inspect and edit models.

3.3.2 Notepad++

Notepad++[10] is a free source code editor with support for many languages. Using the built-in support for Lisp, it provided customizable syntax highlighting for AML code files. Syntax highlighting, auto-completion, and its arguably wider repertoire of editing commands were the main reasons for choosing Notepad++ to write AML code instead of XEmacs.

3.3.3 Visual Studio

Visual Studio[11] is an integrated development environment for the Microsoft .NET Framework[12]. For this project it was used to sketch class hierarchies and generate class diagrams. It was also used to create GUI prototypes for the pilot application and future development plans.

[5] Adaptive Modeling Language: http://technosoft.com/application-software/adaptive-modeling-language/
[6] TechnoSoft Inc.: http://technosoft.com/
[7] Lisp programming language: https://en.wikipedia.org/wiki/Lisp_(programming_language)
[8] Aker Solutions: http://akersolutions.com/
[9] KBeDesign: http://citrix.akersolutions.com/en/Global-menu/Media/Feature-stories/Engineering/KBeDesign/
[10] Notepad++: https://notepad-plus-plus.org/
[11] Microsoft Visual Studio: https://visualstudio.com/
[12] Microsoft .NET Framework: http://microsoft.com/net

4 Pilot KBE Environment

This chapter details the planning, development and evaluation of a pilot KBE environment for doing creative work with a simple product. The resulting proof of concept will provide guidelines for further development.

4.1 Purpose

The main purpose of developing the pilot KBE environment is to test-implement the principles for systematic product design introduced by Tjalve (2003) using a KBE technology. Specifically, we want to see if the following concepts are feasible to develop in AML:

1) Basic Structure
2) Structure Variation
3) Form Variation
4) Functional Surfaces

This selection of principles forms a cognitive framework, as we defined in *2.4.2*. When testing and evaluating the completed environment, we seek to obtain material for answering the questions listed in *1.2.1*:

a) *What are the systematic design constraints applied by the chosen framework?*
b) *How do the framework constraints affect creativity?*
c) *How feasible were the framework constraints to implement in KBE?*

4.2 Scope

4.2.1 Product Synthesis

We assume that we are at Tjalve's *quantified structure* stage of the product synthesis (see Figure 12). This means that the main and sub-functions are already decided, and that a basic structure has been derived from these. In other words, we have decided *how* the product will do its job and *what type* of components will be necessary, but not what form or structure it will have. The next stages thus involve *structure variation* and *form variation*, where the latter also includes defining *functional surfaces*.

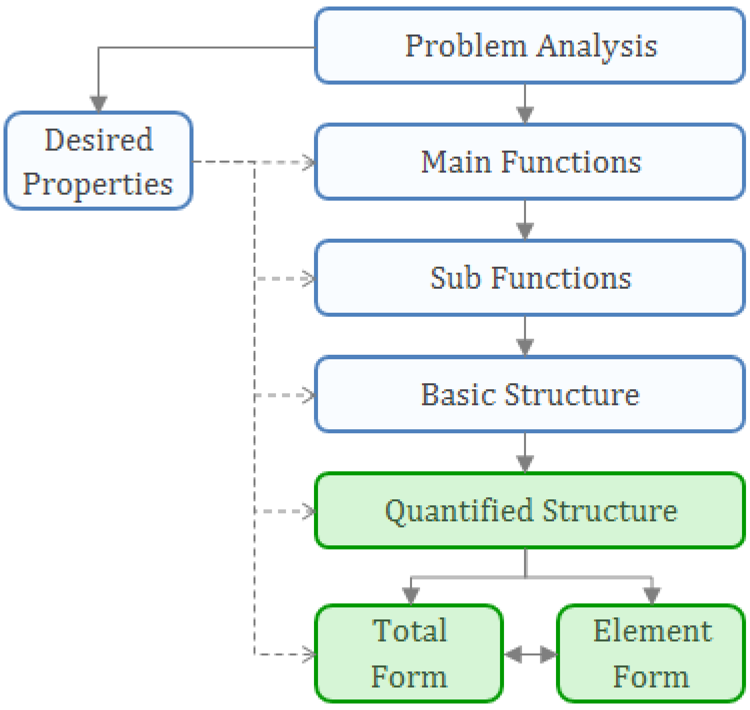


Figure 12: Scope of Tjalve's product synthesis (adapted from Tjalve, 2003)

4.2.2 Workflow

All the product components will be premade for the user. The application will first let them define and connect functional surfaces. It will then aid them in doing form and structure variation, where each of these components can be altered and rearranged.

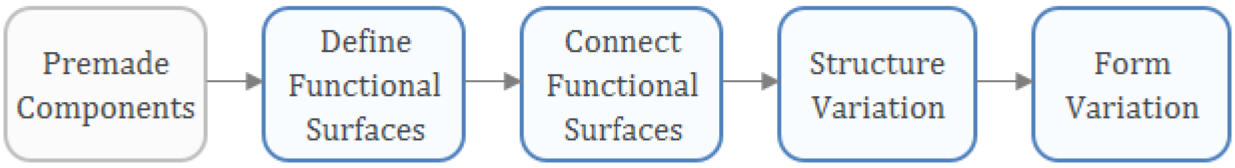


Figure 13: Pilot application workflow

4.2.3 Product Choice

The ultimate goal is to create a generic solution that could aid creativity for a multitude of products. However, the initial development and testing should be focused on a specific product. This removes the burden of having to think generically from the start, meaning less planning and faster coding. Also, if we are already familiar with the product we can more easily imagine various design alternatives and how it functions.

One such product that is commonly found in many homes and offices around the world is the coffee maker. Since coffee can be made in many ways, and we wish to simulate the stages after basic structure, we need to specify our product further. We choose a filtered coffee maker that uses a water heater, a coffee filter, and a pot for collecting coffee. This means that our product can never become an espresso machine, for instance. The coffee maker was chosen because it has few parts, yet many design variations, and its functions can be seen as a directional process through components. Tjalve also mentions tea and coffee makers as examples in his book.

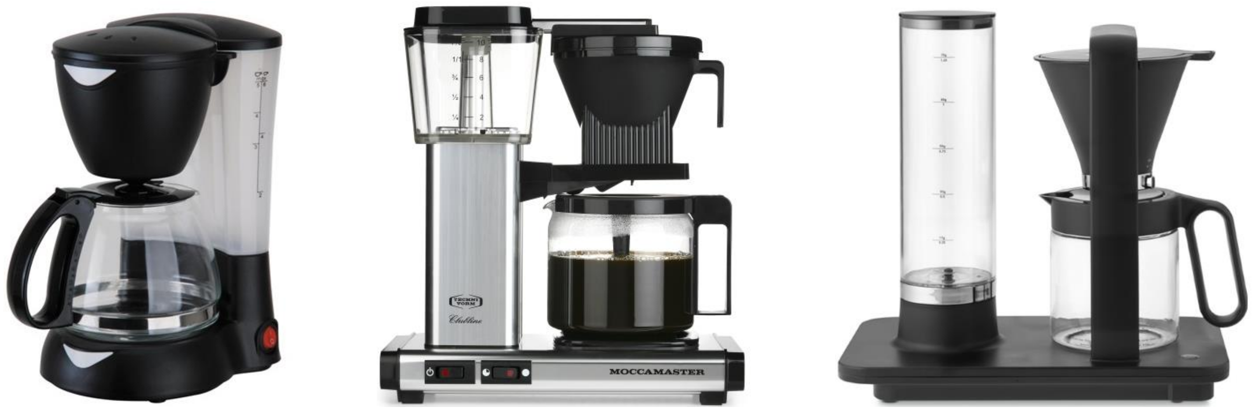


Figure 14: Some filtered coffee maker designs[13]

[13] Coffee makers, from left to right:

- http://topbestcoffeemaker.com/electric-coffee-makers/
- http://en.wikipedia.org/wiki/Drip_brew
- http://williams-sonoma.com/products/wilfa-precision-coffee-maker/

4.2.4 Alternative Products

A previous case study[14] dealt with a KBE bookshelf which had rules concerning shelves and dividing walls. For this pilot, the bookshelf was a less attractive option because it does not have any directional functions that could resemble industrial processes.

There are a range of possible industries to choose products from, such as the automotive, aerospace (e.g. Dahl et al., 2006; Rocca, 2012), and naval engineering industries (e.g. Skogsfjord and Rognseth, 2014). KBE has already proven itself as a strong asset to some of these industries (Rocca, 2012).

The automobile is a common product which was up for consideration. In this case, it would be hard to imagine how Tjalve's quantifiable structure methods would create many reasonable outcomes. Placing the wheels anywhere but under the car would make it dysfunctional. The windows are restricted by the driver's need to view traffic and other safety regulations. Tweaking with the interior or the engine is not an externally visible design alteration. Tjalve does have some good examples with a steamroller, but we preferred a product that most people are more intimately familiar with.

4.2.5 Scope Summary

Based on the product choice and the principles that are to be tested, the pilot application will have the following development scope:

- Only the main product elements will be used (heater, filter, pot).
- The elements will have simple geometry and no technical features.
- The elements will be premade for the user.
- Only a selection of Tjalve's principles will be demonstrated.
- The workflow and main functions will be as shown in Figure 13.
- The GUI will be simplified and less user friendly.
- The coding style will be mostly non-generic.

[14] The KBE bookshelf case study was a pre-master project at NTNU, supervised by Professor Ole Ivar Sivertsen.

4.3 Requirements Specification

4.3.1 Functional Requirements

Functional requirements determine which features should be available to the user. These will later be tested to verify which functions could be successfully implemented.

F 1	Functional Surfaces
F 1.1	Define functional surfaces. The user should be able to select a physical or spatial surface on an element. This must be labeled as a functional surface by the system, which should be visible to the user when inspecting the product.
F 1.2	Connect functional surfaces. The user should be able to select two functional surfaces, mark them as connected, and register the connection type. The system must store this information to work with the surfaces at a later stage.

F 2	Structure Variation
F 2.1	Do structure variation *(arrangement)*. The user should be able to rearrange an element relative to another element. This arrangement must be possible for all axes.
F 2.2	Do structure variation *(number of elements)*. The user should be able to choose how many instances there will be of an element. Any functional surfaces defined on the element (F 1.1) must also be duplicated. Relative positioning (F 2.1) of the individual instances is not a requirement for this pilot. However, arrangement relative to the entire element group must be possible.

F 3	Form Variation
F 3.1	Do form variation *(element shape)*. The user should be able to choose different basic geometries for an element (sphere, cube, cone, etc.). It must still be possible to change the dimensions (F 3.2) in the same way as with the original shape.
F 3.2	Do form variation *(element dimensions)*. The user should be able to change the individual dimensions of each element. The system should update other closely tied dimensions.

4.3.2 Use Cases

Use cases are closely tied to functional requirements because they specify detailed scenarios that the user should be able to perform. These are helpful in the implementation process later. Use cases will be created for functional requirements that benefit from more specification, and will be omitted for further similar scenarios (F 2.1 resembles F 2.2, F 3.1, F 3.2). The following use cases have numbering that corresponds to their respective functional requirement.

Use Case U 1.1 – Define Functional Surfaces

The user will select an element, select one of its surfaces, and signify that they want this to be a functional surface. If there is no surface, for instance a hole, a virtual surface will be created.

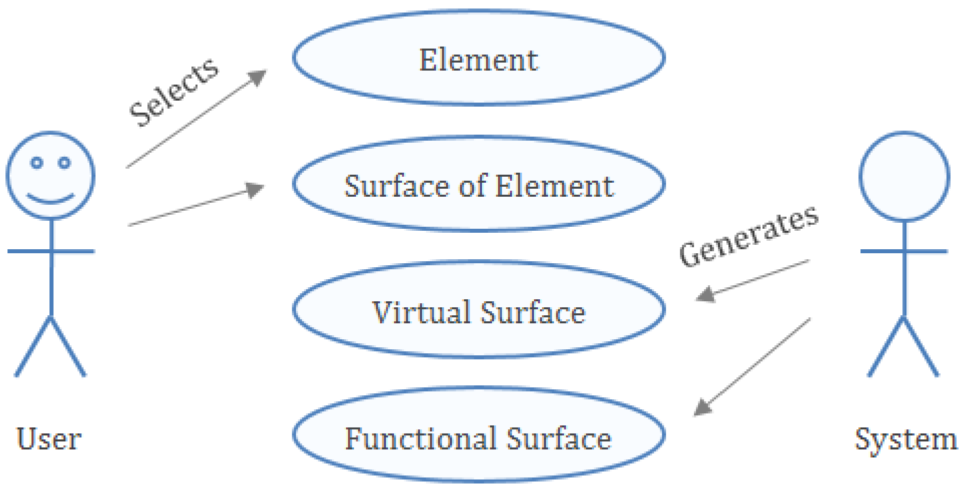


Figure 15: Use case U 1.1 – Define functional surfaces

Use Case U 1.2 – Connect Functional Surfaces

The user will select functional surfaces A and B, which have been created in U 1.1 – Define Functional Surfaces. The connection type and direction will also be chosen.

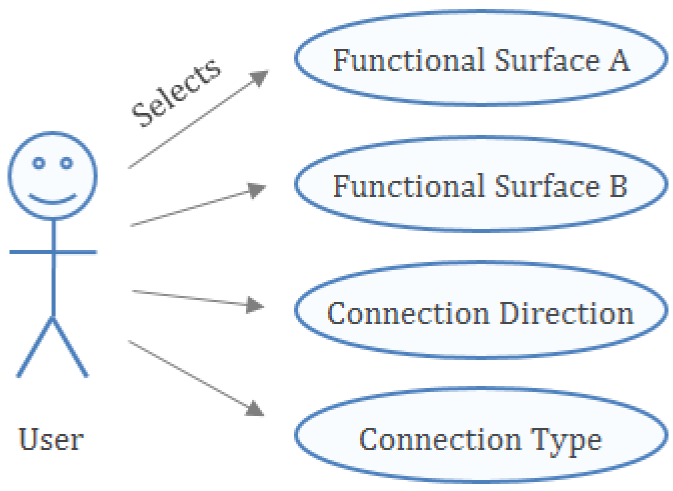


Figure 16: Use case U 1.2 – Connect functional surfaces

Use Case U 2.1 – Do Structure Variation

The user can select an element and easily rearrange it in relation to the other elements. The system will immediately update and display the new arrangement.

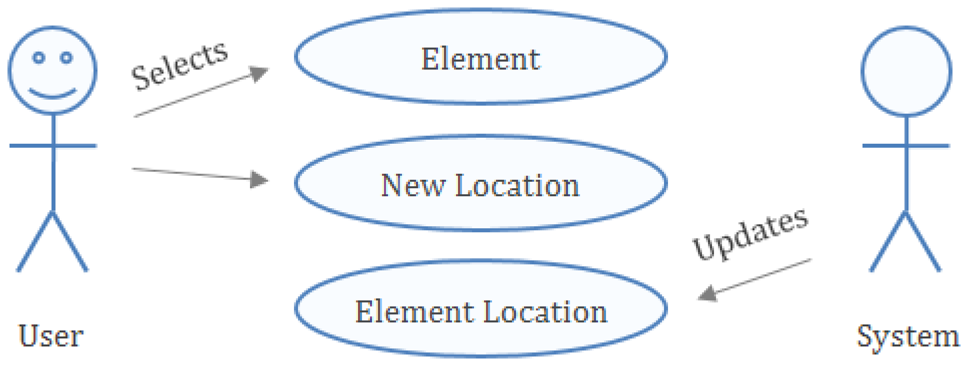


Figure 17: Use case U 2.1 - Do structure variation

4.3.3 Non-Functional Requirements

Many non-functional requirements are tied to the realm of software architecture, which is outside the scope of this pilot environment. However, with a future generic environment in mind, we should adhere to certain principles, such as reusability, modifiability, modularity and encapsulation. We will *not* evaluate these non-functional requirements.

4.4 Development

This section explains how the pilot KBE environment was developed, and shows incremental results. Code extracts are modified for brevity, meaning that basic or repeated similar code may be omitted. The entire source code is listed in Appendix A – Source Code.

4.4.1 Basic Structure

At the quantified structure stage of Tjalve's product synthesis, a basic structure has already been established. To achieve this in AML, we create a hierarchy where the coffee maker is a parent class, which contains the heater, filter and pot classes (see Figure 18). In Tjalve's terms, this corresponds to one *product* with three *elements*.

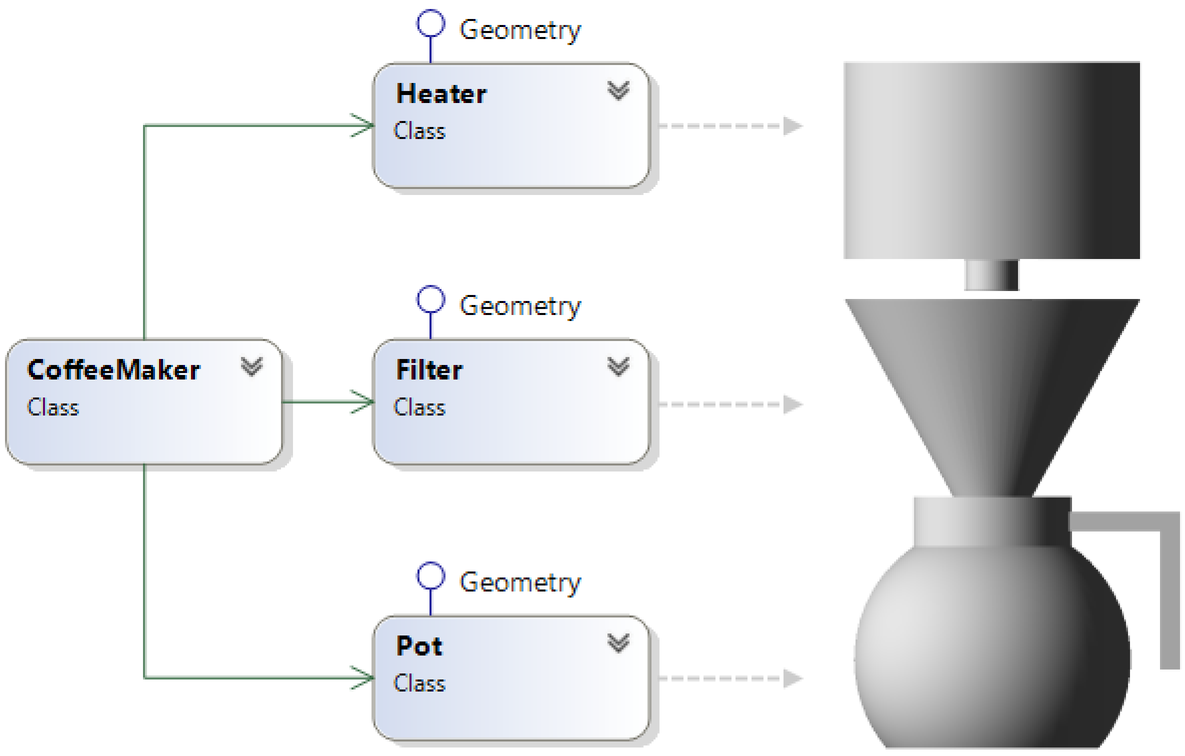


Figure 18: Class diagram and AML model of the coffee maker basic structure

4.4.2 Functional Requirement F 1.1 – Define Functional Surfaces

We need to be able to define functional surfaces which can be associated with specific areas of an element. Using the functional relationship between the heater and the filter as an example, we can define one functional surface for the heater exit pipe, and one for the filter entry area. These are not physical surfaces, but rather a cross section which water can pass through. To create these virtual functional surfaces in AML, we make a `functional-surface` class and define a subclass which inherits from this and holds a `disc-object`.

```
;--------------------------------------------------------------------
;    Class:          functional-surface-disc
;    Description:    Defines a functional surface in the form of a
;                    disc-object. Optional cylinder projection.
;    Reference 1:    Functional Requirement F 1.1
;--------------------------------------------------------------------
(define-class functional-surface-disc
     :inherit-from (functional-surface)
     :properties (
          height          (default 1.0)
          diameter        (default 1.0)
          color           (default 'blue)
     )
     :subobjects (
          (surface :class 'disc-object
               render    'shaded

               .....

          (projection :class 'cylinder-object
               render    'boundary
               orientation (list (
                    translate (list 0.0 0.0 (* 0.5 ^height))))

               .....
```

The `projection` component can be useful to better visualize functional surfaces while viewing the model from certain angles (see Figure 19).

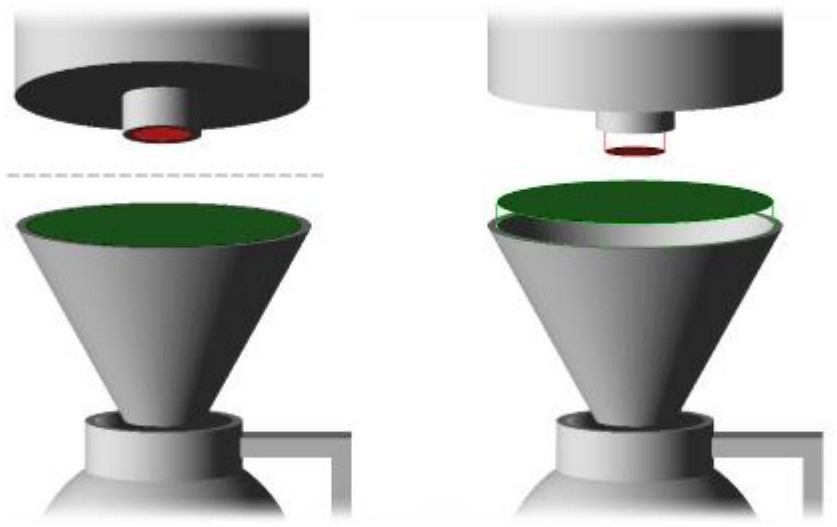


Figure 19: Functional surfaces in AML – heater pipe exit and filter entry

Use Case U 1.1

A custom user interface was created for the purpose of selecting physical and spatial surfaces and have the system create virtual functional surfaces (see Figure 20). However, we discovered that this type of user interaction would be too time-consuming to implement in AML. It was advised that for the remainder of the pilot, we should instead create default AML interfaces linked to geometry and properties. As such, the definition of functional surfaces is done at the development stage, and not by the end user. The classes necessary for these features exist as a proof of concept, but the custom GUI never became more than a prototype.

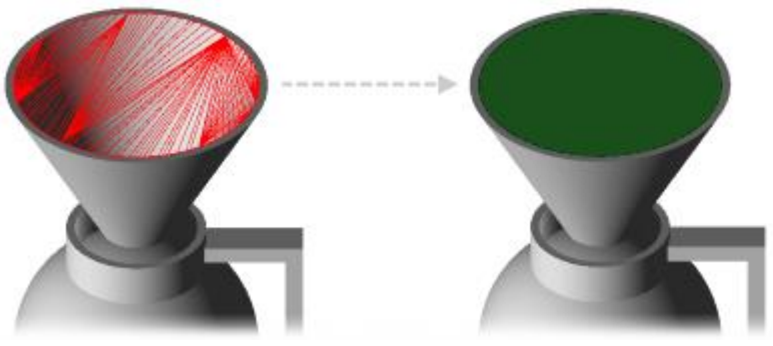


Figure 20: Functional surfaces in AML – created from a selected cross section

4.4.3 Functional Requirement F 1.2 – Connect Functional Surfaces

We need to store information about the functional relationship between two existing functional surfaces. In addition to the surfaces themselves, a connection type and direction will be useful at a later stage for dealing with the logics of element connectivity. The direction tells us *which way* the function naturally flows (`A-to-B`, `B-to-A`, `two-way`, `none`). The connection type says *how* the function flows over this cross section (`stream`, `gravity`, `contact`, `none`). More options can be added as required. The following `functional-surface-link` class contains these properties (see Figure 21).

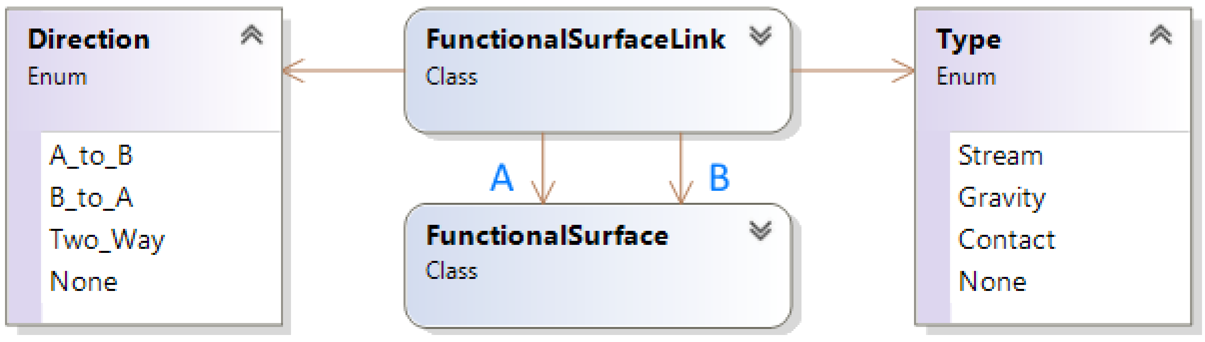


Figure 21: Class diagram for functional-surface-link

```
;-----------------------------------------------------------------
;    Class:          functional-surface-link
;    Description:    A connection between two functional-surface.
;                    The type and direction properties further
;                    specify the functional relationship.
;    Reference 1:    Functional Requirement F 1.2
;-----------------------------------------------------------------
(define-class functional-surface-link
     :inherit-from (object)
     :properties (
          functional-surface-a    nil
          functional-surface-b    nil
          connection-type         nil ; gravity, stream, contact, ...
          connection-direction    nil ; a-to-b,  b-to-a, two-way, ...
     )
)
```

The basic structure (seen in Figure 20), is expanded with `functional-surface` and `functional-surface-link` (see Figure 22). This effectively binds the geometrical elements through their functional relationships. Water exits the heater and enters the filter (`heater-filter-link`), and coffee exits the filter and enters the pot (`filter-pot-link`).

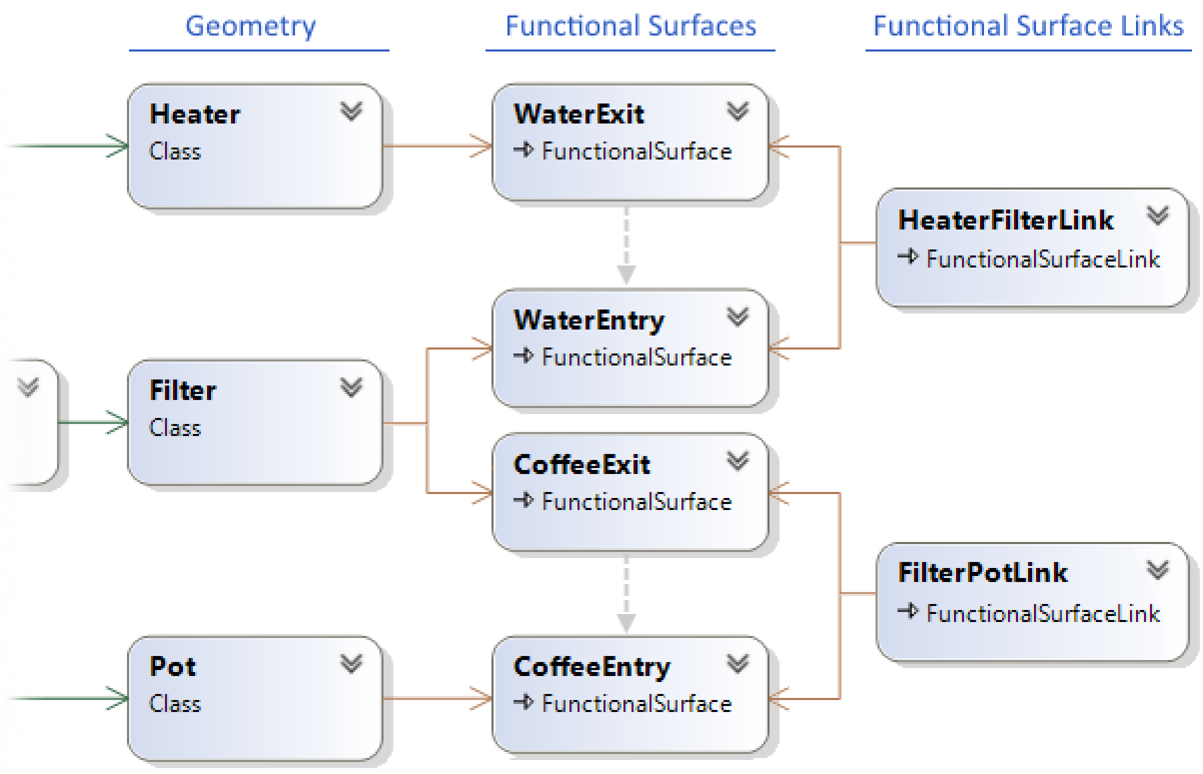


Figure 22: Class diagram of the coffee maker elements linked by functional surfaces

Use Case U 1.2

As with use case U 1.1, developing a custom AML interface for selecting functional surfaces proved too time consuming for this pilot. The preliminary GUI is shown below (see Figure 23). Using the `functional-surface` and `functional-surface-link` classes, along with some built-in AML classes, the feature is most likely possible to implement.

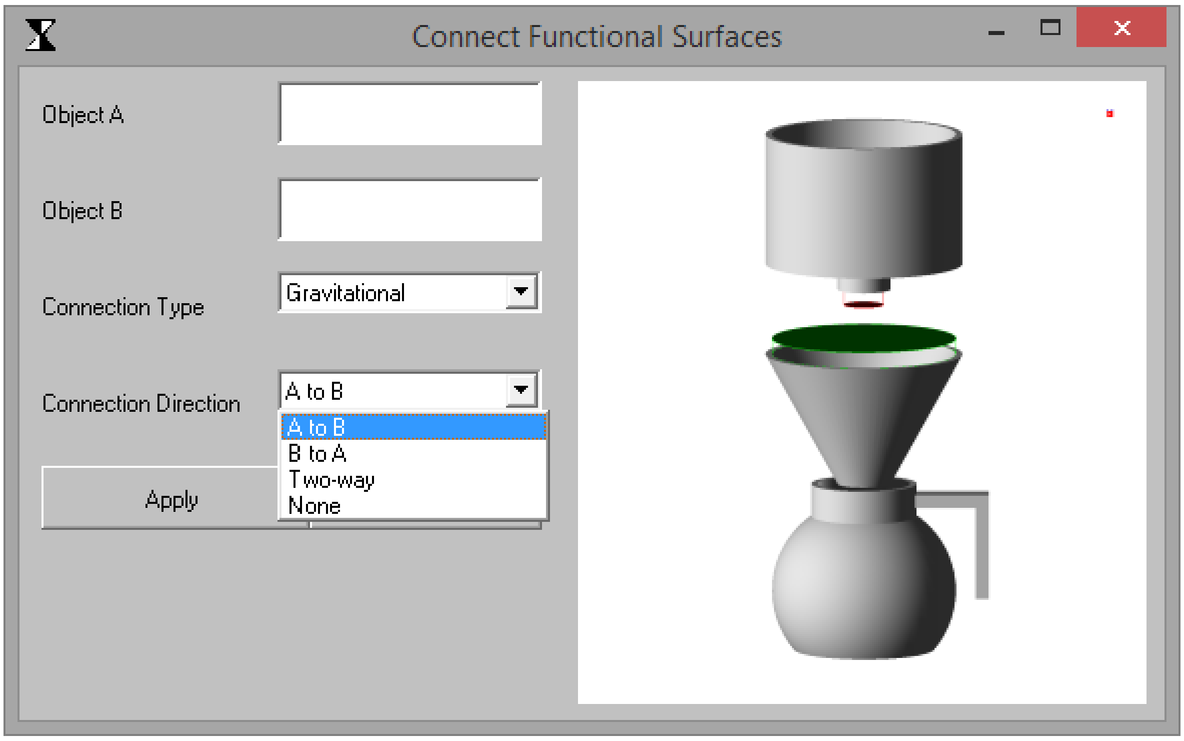


Figure 23: Custom GUI for connecting functional surfaces in AML

4.4.4 Functional Requirement F 2.1 – Do Structure Variation (arrangement)

Moving an element in AML can be done by adjusting its coordinate system's `origin` property, or by offsetting the element itself using `orientation` and `translate`. The former method keeps the code more modular. In structure variation, arrangement is done relative to other elements, and not just some fixed offsets. We need to devise a formula that can replicate this.

If two elements A and B are arranged adjacently along one axis, the distance between their origins must equal half of their dimensions along that axis. For instance, if the walls of two cylinders are touching, the distance between their centers must equal the sum of their radii or half-diameters (see Figure 24).

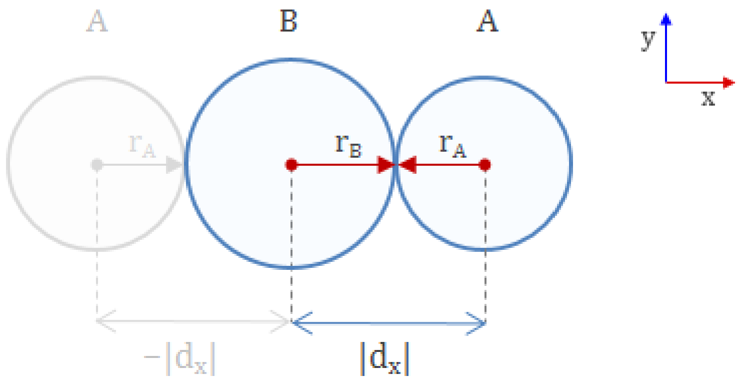


Figure 24: Distance between the centers of two adjacent cylinders

This rule is true for any geometrical element which is symmetrical along the axis in question. The distance formula for each axis is then:

$$|d_x| = \frac{(L_{x,A} + L_{x,B})}{2} \qquad |d_y| = \frac{(L_{y,A} + L_{y,B})}{2} \qquad |d_z| = \frac{(L_{z,A} + L_{z,B})}{2}$$

We can further generalize this to:

$$|d_i| = \frac{(L_{i,A} + L_{i,B})}{2}$$

This lets us arrange an element adjacently to another element along any axis. However, it limits us to exact adjacency along each axis, giving us (3 · 3) - 1 = 26 possible spatial arrangements when excluding the center. This can be imagined as a Rubik's Cube[15], where each outside block is a possible arrangement. If we multiply a distance $|d_i|$ by zero, the arrangement along that axis nullifies. Multiplying by any non-zero number gives us an offset that is a multiple of exact adjacency. For instance, a factor of ± 2.0 provides an open space between elements equal to the average of their dimensions. The resulting formula provides a dynamic way of assigning new coordinates to the element A that is being rearranged:

$$i_A = m_i \cdot \frac{(L_{i,A} + L_{i,B})}{2}$$

[15] Rubik's Cube: https://rubiks.com/

We can now apply this formula to the `origin` property of a coordinate system in AML:

```
;--------------------------------------------------------------------
; Function:         arrange
; Description:      Finds relative arrangement for element A vs B.
;                   Multiplier 0.0 = no offset, 1.0 = adjacent.
; Reference:        Functional Requirement F 2.1
;--------------------------------------------------------------------
(defun arrange (multiplier len-a len-b)
     (* multiplier (* 0.5 (+ len-a len-b)))

.....

; Heater Coordinates (Element A)
;--------------------------------------------------------------------
a-x (arrange ^x-multiplier ^a-diameter ^b-diameter)
a-y (arrange ^y-multiplier ^a-diameter ^b-diameter)
a-z (arrange ^z-multiplier ^a-height   ^b-height)

.....

; Heater Coordinate System (Element A)
;--------------------------------------------------------------------
(a-cs :class 'coordinate-system-class
     origin (list ^^a-x ^^a-y ^^a-z)
     reference-coordinate-system ^^b-cs
)
```

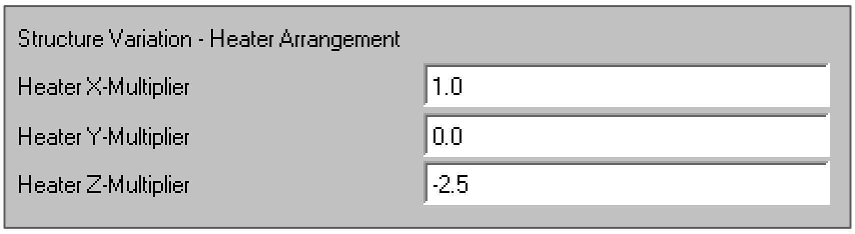
Use Case 2.1

We require an interface that lets the user perform these element rearrangements. With the equation in place, we can expose the multipliers through an interface to let the user do structure variation. This should be easier and faster than to directly input raw coordinates. AML supports automatic creation of input fields by inheriting the `data-model-node-mixin` into the object model and linking the desired properties using its `property-objects-list`.

```
;--------------------------------------------------------------------
;    Class:         coffee-maker-gui
;    Description:   Provides a GUI for the user to alter the
;                   coffee-maker through various inputs.
;    References:    Functional Requirements F 2.1, 2.2, 3.1, 3.2
;--------------------------------------------------------------------
(define-class coffee-maker-gui
     :inherit-from (coffee-maker data-model-node-mixin)
     :properties (

     .....
```

```
; Automatic Properties
property-objects-list (list
     "Structure Variation - Heater Arrangement"
     (the superior x-multiplier self)
     (the superior y-multiplier self)
     (the superior z-multiplier self)
```

The result is a set of labeled numeric input boxes (see Figure 25). Editing the multipliers will rearrange the elements drawn on the canvas (see Figure 26).



Structure Variation - Heater Arrangement	
Heater X-Multiplier	1.0
Heater Y-Multiplier	0.0
Heater Z-Multiplier	-2.5

Figure 25: Structure variation in AML – default GUI for arrangement multipliers

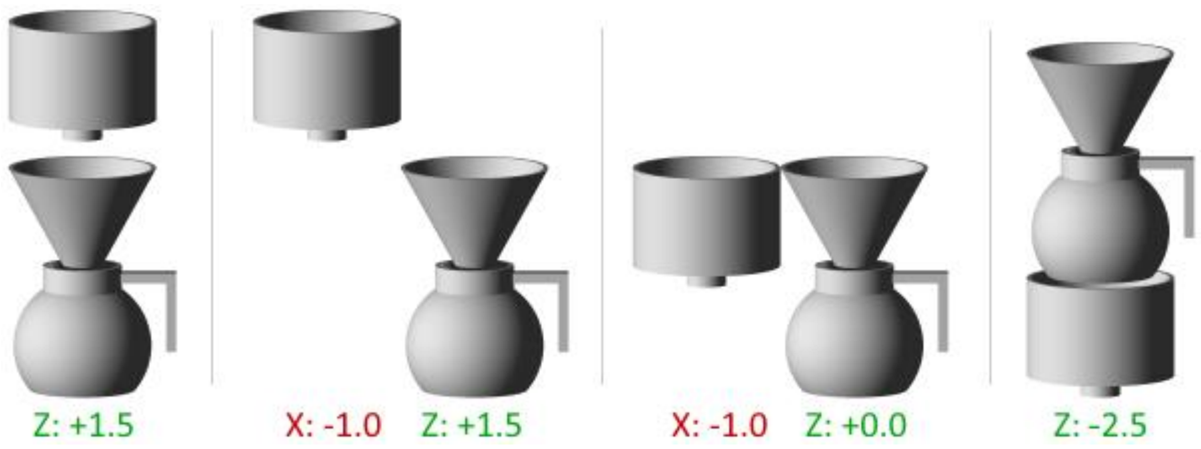


Figure 26: Structure variation in AML – rearranging the heater element

4.4.5 Functional Requirement F 2.2 – Do Structure Variation (number of elements)

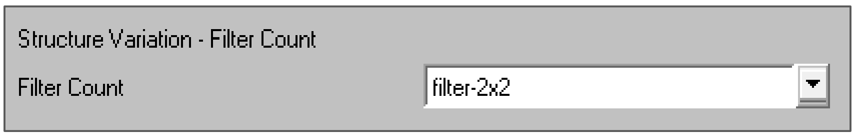
Let us assume that we want to design a coffee maker with separate filters for different types of coffee to avoid “contamination”. To achieve this second aspect of structure variation, we need to be able to duplicate elements. We can define a single element as a generic class in AML, and use it to create a parent class which contains multiple child elements. The parent and children can have linked properties, which lets us conveniently change all instances from one place. The following classes lets us choose between a single filter, or four filters in a two-by-two layout.

```
;-----------------------------------------------------------------
;    Class:         filter
;    Description:   Defines a single filter element. Uses two
;                   truncated-cone-object for wall thickness.
;-----------------------------------------------------------------
(define-class filter
     :inherit-from (difference-object)
     :properties (
          b-height       (default 1.0)  ; Auto-link
          b-diameter1    (default 1.0)  ; Auto-link
          b-diameter2    (default 2.0)  ; Auto-link
          wall-t         (default 0.25) ; Auto-link

          .....

     :subobjects (
          (outer :class 'truncated-cone-object
               height              ^b-height
               start-diameter      ^b-diameter1
               end-diameter        ^b-diameter2
          )
          (inner :class 'truncated-cone-object

          .....

;-----------------------------------------------------------------
;    Class:         filter-2x2
;    Description:   Defines a 2 by 2 grid of Filter elements.
;    Reference 1:   Functional Requirement F 2.2
;-----------------------------------------------------------------
(define-class filter-2x2
     :inherit-from (object)
     :properties (


          .....

          diameter1      (* 0.5 ^b-diameter1) ; Half lengths (2x2)
          diameter2      (* 0.5 ^b-diameter2) ; Half lengths (2x2)
          offset         (arrange 0.5 ^diameter2 ^diameter2)

     :subobjects (
          (filter-1 :class 'filter
               b-diameter1 ^diameter1 ; Specific, not auto linked
               b-diameter2 ^diameter2 ; Specific, not auto linked
               orientation (list (
                    translate (list ^^offset ^^offset 0.0)))
          )

          (filter-2 :class 'filter . . .)

          (filter-3 :class 'filter . . .)

          (filter-4 :class 'filter . . .)
```

We add an `option-property-class` to the `coffee-maker` and link the filter instance to this property. The user can then select between one and four filters from a dropdown menu to update the model (see Figure 27 and Figure 28).

```
; Filter Count Selection
;-----------------------------------------------------------------
(b-filter-selection :class 'option-property-class
     label          "Filter Count"
     options-list   '(filter filter-2x2)

     .....

; Filter (Element B)
;-----------------------------------------------------------------
(element-b :class !b-filter-selection
     ; All dimensions auto-linked
)
```

When duplicating the number of filter elements, we also need to ensure any related entities are scaled accordingly, for instance functional surfaces (see Figure 28). The code for duplicating functional surfaces can be found in Appendix A – Source Code (`functional-surface-disc` and `functional-surface-disc-2x2`).


Structure Variation - Filter Count

Filter Count: filter-2x2

Figure 27: Structure variation in AML – default GUI for number of filters

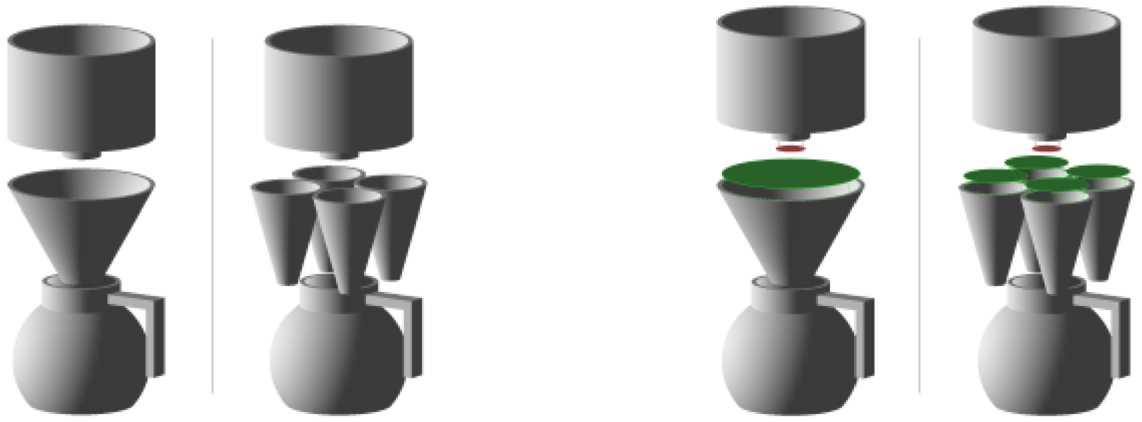


Figure 28: Structure variation in AML – number of filter elements

4.4.6 Functional Requirement F 3.1 – Form Variation (element shape)

An example of changing the shape of an element could be to choose a cubic water tank instead of a cylindrical one. This is equivalent to changing the base class of an element in AML. Since each class has unique properties for dimensions, we should add a class layer to genericize these. This lets us set the width of a cube and the diameter of a cylinder using the same property. The following classes define cubical and cylindrical water tanks.

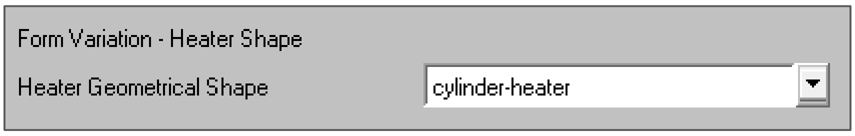
```
;-------------------------------------------------------------------
;    Class:         cylinder-heater
;    Description:   Defines a cylindrical shape for the water tank.
;    Reference 1:   Functional Requirement F 3.1
;-------------------------------------------------------------------
(define-class cylinder-heater
     :inherit-from (cylinder-object)
     :properties (
          a-height       (default 1.0)
          a-diameter     (default 1.0)
          height         ^a-height
          diameter       ^a-diameter

          .....

;-------------------------------------------------------------------
;    Class:         box-heater
;    Description:   Defines a cubical shape for the water tank.
;                   X and Y dimensions are set from the diameter.
;    Reference 1:   Functional Requirement F 3.1
;-------------------------------------------------------------------
(define-class box-heater
     :inherit-from (box-object)
     :properties (
          a-height       (default 1.0)
          a-diameter     (default 1.0)
          height         ^a-diameter
          width          ^a-diameter
          depth          ^a-height
```

To enable choosing between these classes, we add an `option-property-class` to the `coffee-maker` and link the heater element shape to this property.

```
; Heater Shape Selection
;-------------------------------------------------------------------
(a-shape-selection :class 'option-property-class
     label          "Heater Geometrical Shape"
     options-list   '(cylinder-heater box-heater)

     .....
```

```
; Heater (Element A)
;-----------------------------------------------------------------
(element-a :class 'difference-object
     object-list (list ^outer ^inner)

     ; Tank outer wall
     (outer :class !a-shape-selection
          ; All dimensions auto-linked
     )

     ; Tank inner space, floor intact
     (inner :class !a-shape-selection
          a-diameter (inner-diameter ^^a-diameter ^^wall-t)
          orientation (list (translate (list 0.0 0.0 ^^wall-t)))
     )
)
```

The results are shown below (see Figure 29 and Figure 30).


Form Variation - Heater Shape

Heater Geometrical Shape	cylinder-heater

Figure 29: Form variation in AML – default GUI for heater shape

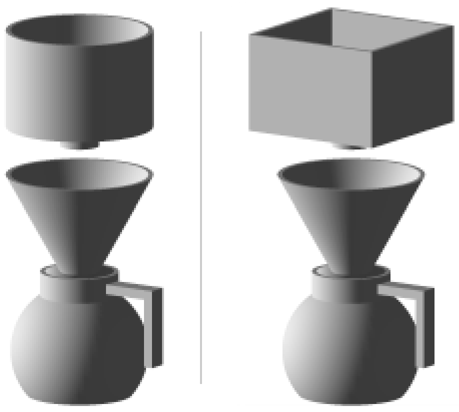


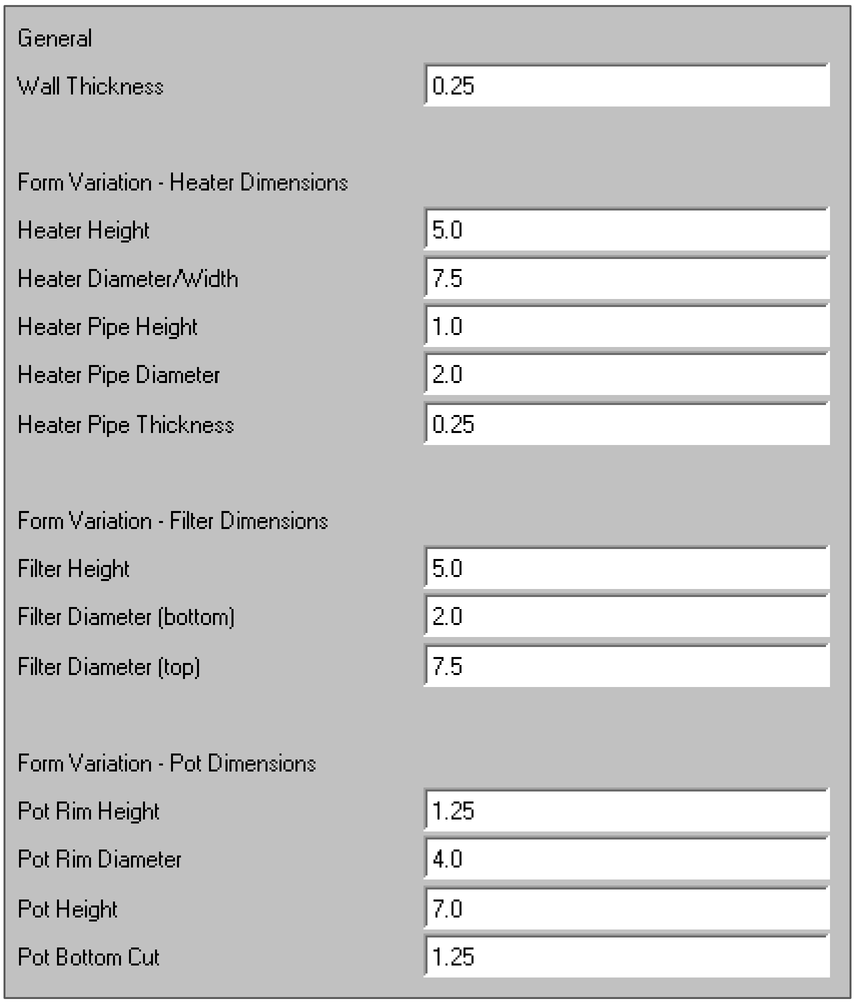
Figure 30: Form variation in AML – heater shapes

4.4.7 Functional Requirement F 3.2 – Do Form Variation (element dimensions)

Changing dimensions that are not bound by any knowledge is very straight forward to implement in AML. We simply need to link the element dimensions to the GUI using the `editable-data-property-class` as we did before.

```
; Pot Dimensions (Element C)
;------------------------------------------------------------------
(c-rim-height :class 'editable-data-property-class
     label "Pot Rim Height"
     formula 1.25
)
(c-rim-diameter :class 'editable-data-property-class
     label "Pot Rim Diameter"
     formula 4.0
)
(c-pot-height :class 'editable-data-property-class
     label "Pot Height"
     formula 7.0
)
(c-pot-bottom-cut-height :class 'editable-data-property-class
     label "Pot Bottom Cut"
     formula 1.25
)
```

Linking these properties in the `coffee-maker-gui` exposes them to the user in the default interface (see Figure 31 and Figure 32).



General	
Wall Thickness	0.25
Form Variation - Heater Dimensions	
Heater Height	5.0
Heater Diameter/Width	7.5
Heater Pipe Height	1.0
Heater Pipe Diameter	2.0
Heater Pipe Thickness	0.25
Form Variation - Filter Dimensions	
Filter Height	5.0
Filter Diameter (bottom)	2.0
Filter Diameter (top)	7.5
Form Variation - Pot Dimensions	
Pot Rim Height	1.25
Pot Rim Diameter	4.0
Pot Height	7.0
Pot Bottom Cut	1.25

Figure 31: Form variation in AML – default GUI for element dimensions

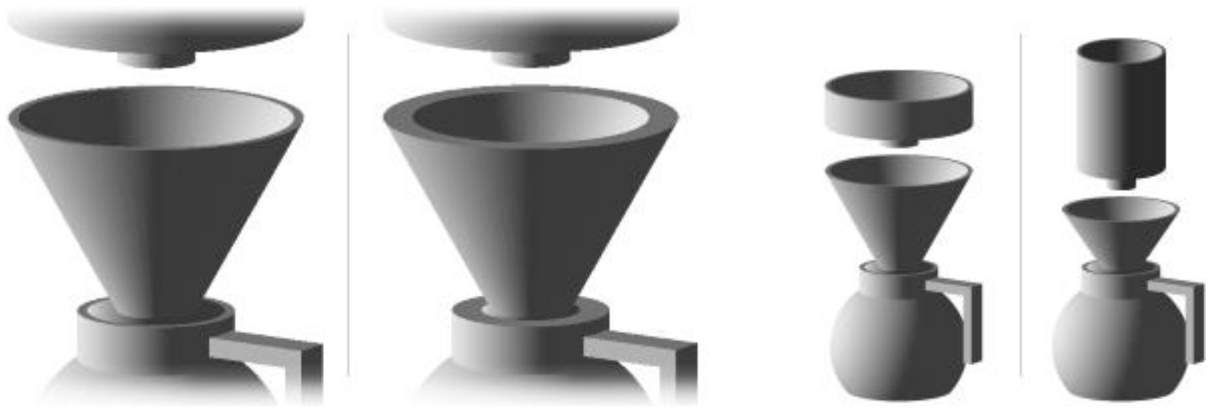


Figure 32: Form variation in AML – changing dimensions (thickness, size)

4.5 Final Implementation

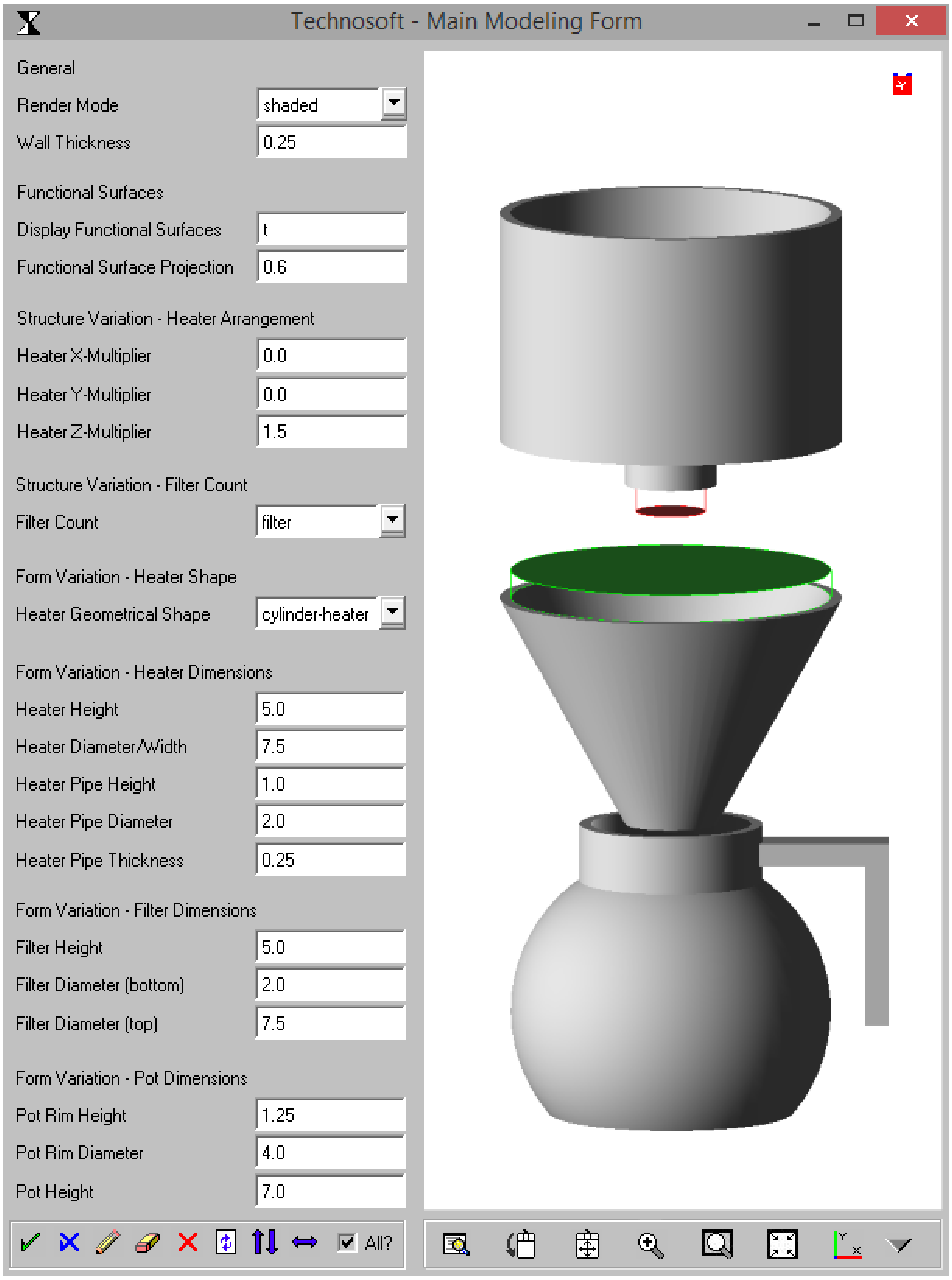
This section shows the final implementation of the pilot application with screenshots of the environment and examples of possible product variations.

4.5.1 Environment

The final pilot KBE environment has a control panel on the left for making design variations, and a graphics canvas on the right where the product is shown (see Figure 33). In reality, the AML interface is larger with more features, but has been cropped to show only what is relevant here.

Most of the features in the control panel have already been covered in 4.4 Development, but to summarize, the user can do the following:

- Change general properties such as render mode and wall thickness (form variation)
- Toggle functional surfaces on/off and change their projection height
- Change the arrangement of the heater element (structure variation)
- Change the number of filter elements (structure variation)
- Change the shape of the heater element (form variation)
- Change the dimensions of any element (form variation)



General

Render Mode	shaded
Wall Thickness	0.25

Functional Surfaces

Display Functional Surfaces	t
Functional Surface Projection	0.6

Structure Variation - Heater Arrangement

Heater X-Multiplier	0.0
Heater Y-Multiplier	0.0
Heater Z-Multiplier	1.5

Structure Variation - Filter Count

Filter Count	filter

Form Variation - Heater Shape

Heater Geometrical Shape	cylinder-heater

Form Variation - Heater Dimensions

Heater Height	5.0
Heater Diameter/Width	7.5
Heater Pipe Height	1.0
Heater Pipe Diameter	2.0
Heater Pipe Thickness	0.25

Form Variation - Filter Dimensions

Filter Height	5.0
Filter Diameter (bottom)	2.0
Filter Diameter (top)	7.5

Form Variation - Pot Dimensions

Pot Rim Height	1.25
Pot Rim Diameter	4.0
Pot Height	7.0

Figure 33: Final pilot KBE environment

4.5.2 Product Variations

Combining all implemented techniques for structure and form variation, we are able to produce a wide range of designs from the basic structure (see Figure *34*).

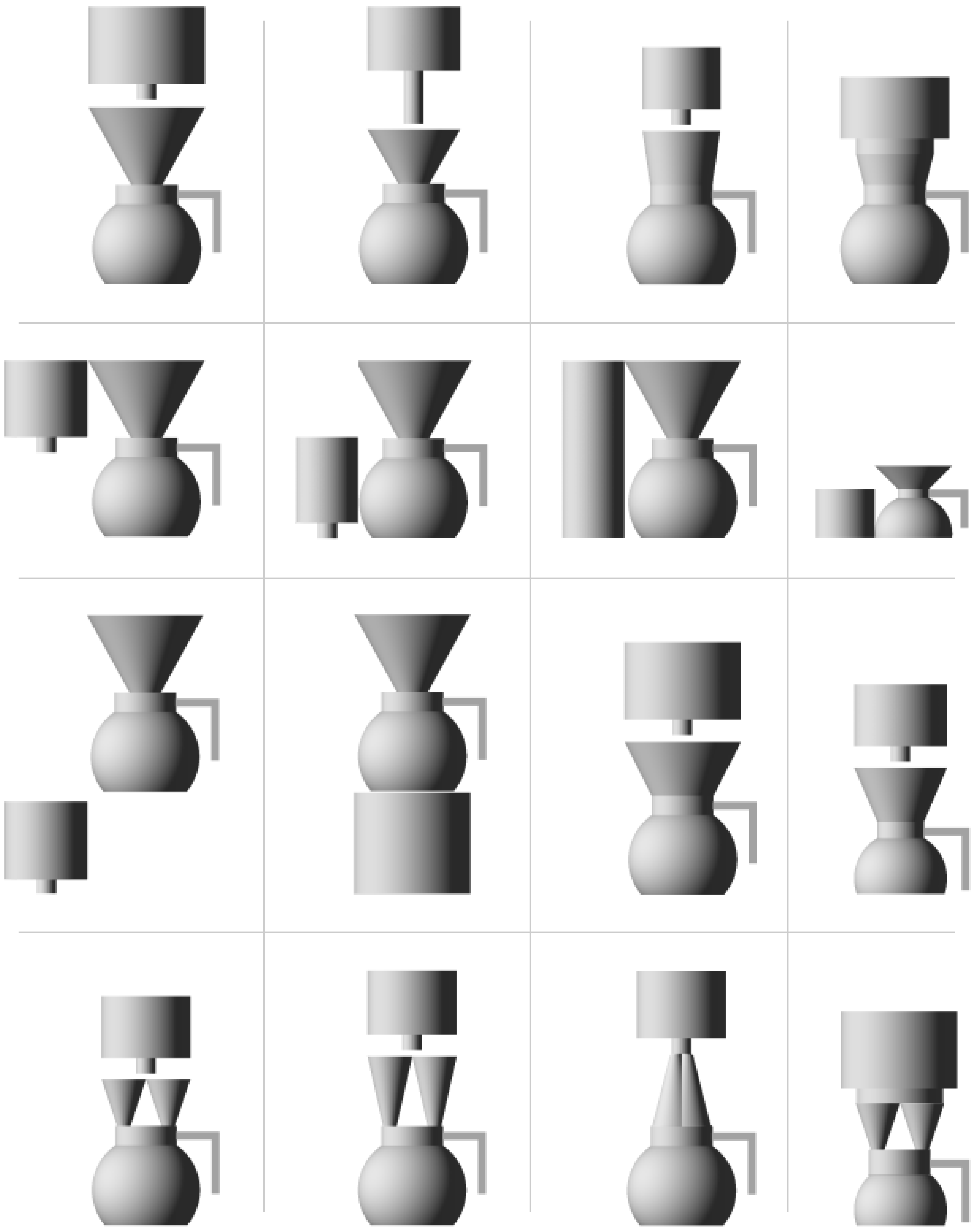


Figure 34: Product variations using Tjalve's principles in KBE

4.6 Test Report

All functional requirements are tested by the developer. In future development, we would include end users in any tests, to help verify user friendliness and receive other feedback.

4.6.1 Verification of Functional Requirements

F 1	Functional Surfaces
F 1.1	Define functional surfaces. The system has pre-generated virtual functional surfaces between the heater and filter elements. Once the base concept has been implemented, it can with reasonable amount of effort be added to other elements. The graphical user interface for selecting these surfaces was not feasible to implement in this pilot due to high development time (discussed in 4.4.2). Overall verification: Partial Success Initial development time: Medium Further development time: Medium (per added element/surface)
F 1.2	Connect functional surfaces. The system has the classes and mechanics needed to handle connection of functional surfaces. However, the initial GUI was discontinued for the same reasons as F 1.1. The GUI was reduced to a default AML interface where the user can link two functional surfaces by choosing connection type and direction. Overall verification: Partial Success Initial development time: Medium Further development time: Low (per added connection)

F 2	Structure Variation
F 2.1	Do structure variation *(arrangement)*. The user is able to move an element relative to another by entering positioning multipliers (devised in 4.4.4). It is implemented for the heater relative to the filter, and the general formula can easily be used on new elements by applying it to their coordinate systems.

Overall verification:	Success
Initial development time:	High
Further development time:	Low (per added element)

F 2.2 Do structure variation *(number of elements)*.

The user is able to select the number of instances of an element from a dropdown menu. This is achieved through the definition of a parent element that holds several original elements. We reuse a generic class, which is good, but it takes time to create the parent class for each new variation, and any belonging functional surfaces must also be multiplied.

Overall verification:	Success
Initial development time:	High
Further development time:	High (per added element)

F 3 Form Variation

F 3.1 Do form variation *(element shape)*.

The user is able to select a different geometrical shape for an element from a dropdown menu. This feature is implemented for the heater element, and can with a reasonable amount of effort be added to other elements, or include new shapes. If the new shape involves a change in functional surfaces, the required time is higher.

Overall verification:	Success
Initial development time:	Medium
Further development time:	Medium (per added element/shape)

F 3.2 Do form variation *(element dimensions)*.

The user is able to change several dimensions on all elements. This is done by numerical input of new sizes. This type of feature could be said to be native to AML, thus it requires little effort to implement.

Overall verification:	Success
Initial development time:	Medium
Further development time:	Low (per added dimension input)

4.7 Evaluation

This chapter evaluates the final implementation of the pilot application. We discuss what could have been done differently and suggest possible trajectories for future development. Parts of the evaluation is based on test results and theoretical findings. Other parts are based on the reflections and opinions of the developer and tester.

4.7.1 Development Issues

Functional Requirement F 1.1 and 1.2 – Custom Interfaces

The custom interfaces for functional surfaces could not be made as planned. Having pre-defined surfaces is decent enough for this simple coffee maker. Some of the work is shifted from user to developer, but leaves the user with less freedom in specifying their product. To make the application generic and user friendly, a custom interface should be a priority in future development.

Functional Requirement F 2.2 – Structure Variation (number of elements)

As noted in the test report, enabling this feature for new elements requires a high amount of development time. This is because we need to create a parent class for each new variation, and add a corresponding set of functional surfaces. We would also have to consider this added work load if the application were to automatically generate pipes between the heater and filter.

Looking at the class structure, if we had decided to incorporate the functional surfaces inside the original class instead, we could effectively eliminate this extra work per new variation. In addition, there are methods in AML for systematically repeating objects, for instance along curves. This approach may be useful to genericize the code to quicken further development.

4.7.2 Effects on Creativity

We wanted to evaluate how the application supports or inhibits creativity. During and after requirements testing, a wide range of product variations were made, some of which were shown in 4.5.2. Reflecting upon the creative flow during this design process allows us to identify some of the contributing factors.

Cognitive Framework Constraints

We have implemented our cognitive framework for creative design, derived from principles by Tjalve (2003). The four principles that we chose in section 4.1 have now become constraints that this framework applies to the design process:

1) The product *basic structure* is decided.
2) Elements can be rearranged and multiplied with *structure variation*.
3) Elements can be assigned new geometry and dimensions with *form variation*.
4) Variations must consider *functional surfaces* to retain product functionality.

Supporting Factor – Systematic Design

Having performed a large number of variations, there were times when no new ideas would emerge. When this happened, I would look at the user interface to find operations that I had not yet tried out. It could then happen that I suddenly entered a state of renewed flow, and more ideas kept emerging. I think that having systemized Tjalve's principles into a KBE environment can allow a designer to fall back to these guidelines for inspiration while the creative flow is poor.

Supporting Factor – Visual Feedback

Being able to continuously inspect the product on a 3D canvas was helpful to evaluate any new ideas. This is perhaps comparable to for instance cardboard mockups, but less tangible and more detailed. Still, I think that tangible mockups have the advantage of being faster than the current user interface. In order to compete, the interface needs to be better, and a future application should have some other knowledge-based advantages that a cardboard mockup cannot offer.

Inhibiting Factor – Limited Tools

I discovered that my creativity was sometimes hampered by the limitations of the pilot environment. There were operations that I foresaw in my mind that simply could not be done with the available variation tools. This would be an unfortunate situation for a designer, effectively stopping his creative flow. However, this is only a pilot application, and these flaws should be expected for several reasons:

1) Not all variation principles were to be implemented (Tjalve's or otherwise). One specific operation that I would find useful for certain arrangements is the ability to rotate individual elements. For a future application with more knowledge implemented, the system should account for potential issues such as functional surfaces with connection type gravity.

2) The implementation is not as sophisticated or detailed as the underlying theory. For instance, in the case of form variation, a person's ideas for new geometrical shapes may emerge with more specificity than the system is able to reproduce. A future version could have more basic shapes available, but also various extrusion and difference tools for creating more advanced geometry.

3) Some implemented principles were not available for the entire product. For instance, element rearrangement and basic shape change was only available for the heater.

4) The constraints of having a chosen basic structure. I found this limiting at times when more radical ideas emerged, until I realized that it would actually change *how* the product performed its task. In one sense, this limits creativity, while in another, it helps focus our creative efforts (as proposed in 2.3.6) and not get off track.

Inhibiting Factor – Graphical User Interface

Each incremental product variation can be done with relative ease. However, numeric input and dropdown menus are not optimal in terms of efficiency and user friendliness. Operations should be much faster in order to speed up the idea-to-realization cycle. I found that too much time and focus was spent on inputting values and redrawing the model. This mundane task got annoying over time, and I think it distracts our minds enough to hamper creative flow. I was likely having a taste of the negative "eyebrow effect", leading my task solving *system II* to dominate instead of the more creative *system I* (Kahneman, 2013).

4.7.3 Other Comments

Form Variation – Total and Element Form

Tjalve distinguishes between total form and form of the elements (described in 2.5.6 Total and Element Form). He also states that these are naturally connected. When changing the form of an element, the total form of the product must follow, and vice versa. If desired, we could implement more automation related to this in our KBE environment. Using AML, we can assign dependencies to dimensions and geometry. We could for instance have overall product dimensions like height or width, which automatically scale all individual element dimensions.

Product Variations – Degrees of Freedom

Let us consider our product variation tools in terms of degrees of freedom. For each element in the product, we have:

- Structural arrangement along three axes.
- Structural repetition of an instance.
- Different basic geometrical shapes.
- One or more dimension properties, such as height, width, thickness.

This gives us at least six degrees of freedom *per element*. For a user, more freedom equals more variation capabilities, but perhaps also fewer positive constraints, as Jocko Willink would have it (2.3.6). For complex products, the number of possible combinations would increase at an exponential rate for each new element added. If the application later evolved into some type of generative system, it would need manual or intelligent filtering to manage its solution space.

Product Variations – Viability of Outcomes

In the current state of the application, some of the product variants that can be created would not be immediately usable. There is currently no intelligent support if an element is displaced in some way that logically breaks its functional surface link. For instance, if the heater exit hole is placed outside or below the filter entry, we would require pipes and/or pumps to transport the water. Also, if we choose multiple filter elements, we would need individual pipe exits or some kind of rotational system.

5 Future Development

This chapter suggests the future development of a KBE environment for supporting creative design. We base these suggestions on the evaluation of the pilot application and any relevant theoretical findings. The GUI prototypes herein are made to illustrate some of these concepts, but they are not yet functional KBE environments.

5.1 Features

This kind of system competes with traditional methods for early creative design, like cardboard mockups. These are excellent because they are fast, tangible and inexpensive. The environment should offer some more advanced knowledge-based features that such tangible methods cannot.

5.1.1 Automatic Creation of Support Elements

When a product is changed through for instance structure and form variation, the functional surface connections will sometimes become invalid. This could be due to a gravitational flow between two elements that is no longer possible because the exit surface is below the entry surface. In the coffee maker case, moving the heater element outside or below the filter would require water pipes and/or pumps to be added. The KBE environment could be designed to automatically add these support elements when needed. This would let the user to creatively design his product without having to worry about breaking its functions.

5.1.2 CAD Import

AML supports import of industry-standard files like IGES, STEP, STL and DXF[16]. This could let us build modules using CAD, and import them into our KBE environment to do creative work. There could be some form of intelligence that helps with the logical arrangement of incoming objects, using functional surfaces or otherwise.

[16] AML support for import/export: http://technosoft.com/application-software/adaptive-modeling-language/

5.1.3 Product Stats

KBE gives the ability to implement cost functions. These could for instance calculate material or production costs as the product changes. Inspired by this, we could create a live updated display of various product stats. Similar calculated product stats are demonstrated in for instance TechnoSoft's AMRaven, *Adaptive Modeling for Rapid Air Vehicle Engineering* (e.g. Dahl et al., 2006), and in *Parametric Ship Hull Design in NX* (Skogsfjord and Rognseth, 2014).

Coffee maker stats could include material costs, production costs, element count, power consumption during use, brewing time, pot volume, and so on. Automatically added pipes and pumps would increase the material costs, production costs, and power consumption. The designer would receive immediate feedback on his changes, and could choose which stats to optimize for. This should further ease the cognitive load on the designer, possibly allowing more effort to be spent on being creative instead.

Radar Chart

Another way to more visually represent product stats could be through a radar chart[17]. This places three or more stats on individual axes in a two-dimensional diagram (see Figure 35). Higher stats are represented with further distance from the chart origin. It may be faster to determine the overall size of stats by glancing at a radar chart instead of raw numbers.

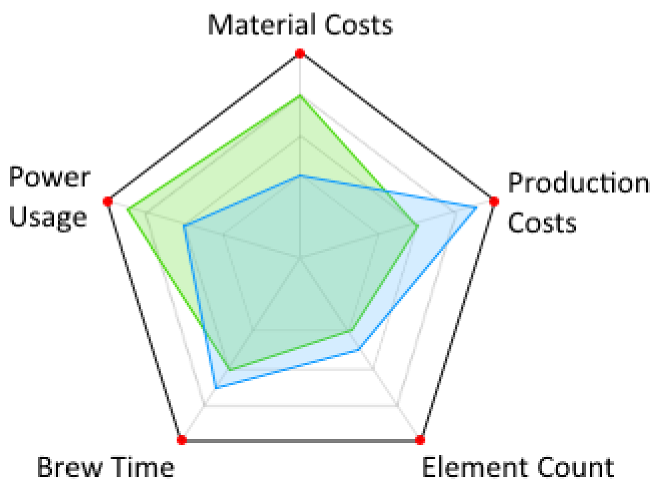


Figure 35: Radar chart with stats for two coffee makers

[17] Radar Chart: https://en.wikipedia.org/wiki/Radar_chart

5.2 GUI Prototypes

5.2.1 Defining and Connecting Functional Surfaces

The improved versions of these interfaces are more about interaction style than design. In the pilot application we failed to demonstrate a click-to-select method for element surfaces. Since this is not something we can explicitly show in a new image of a prototype GUI, we shall refer to the previously created pilot prototype (seen in Figure 23). The new interface would be similar to this, but with the ability for the user to click on physical and spatial surfaces to do their selection.

5.2.2 Manual Product Variation

In the pilot application, we created a default AML interface for product variation (Functional Requirements F 2.1, 2.2, 3.1, 3.2). The following prototype is a suggestion to how we can improve this interface (see Figure *36*).

Structure Variation

Arranging elements was previously done by entering positioning multipliers to make them adjacent, spaced or overlapping along an axis. Although the concept is relatively simple, the interface could be optimized to avoid hindering the creative flow. This version uses buttons to immediately reposition an element and show the update in the canvas. By simultaneously pressing the control, shift or alt/option buttons, the movement would be far, short or aligning.

Another possibility would be to have some kind of drag and drop system where the user could simply pull on an element to reposition. The system could then suggest aligned positions by letting the element “snap” into place depending on which modifier keys are held down.

The user can also press a button to align elements according to their functional surface connections. In the case shown below, the heater would then be centered directly above the filter with intersecting functional surfaces.

The second aspect of structure variation is done by choosing number of elements from a dropdown menu. This is the same as before, but we could now have more options and make sure that the model view instantly updates itself upon a new selection.

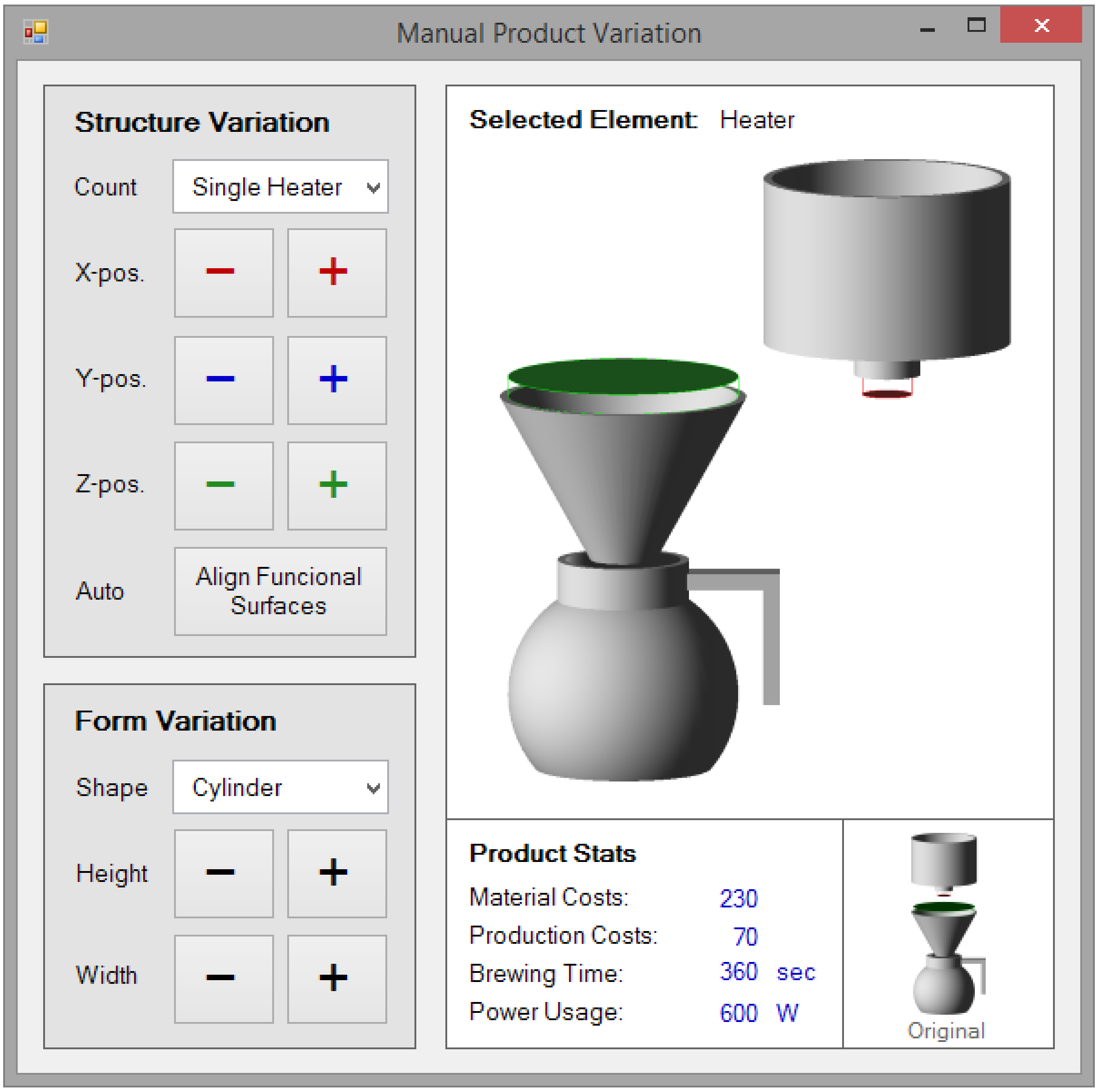
Form Variation

The form variation toolset follows the same behaviors as the controls for structure variation. Selecting new geometrical shapes is done from a dropdown menu, and element dimensions are adjusted incrementally using buttons. Holding the modifier keys determines the big or small change in dimensions, will determine how much the dimensions, or if they should auto align to other elements.

We could also add more advanced form variation options here, such as custom extrusions, difference objects, and edge rounding.

Other Features

Below the main canvas, the user can view live updated product stats, and a miniature version of the unchanged product. The user could optionally show these stats as a radar chart.



Selected Element: Heater

Product Stats

Material Costs:	230	
Production Costs:	70	
Brewing Time:	360	sec
Power Usage:	600	W

Figure 36: Prototype GUI for doing manual product variations

5.2.3 Automatic Product Variation

We recall Asimov (1959) who states that great ideas often stem from finding a connection between two entities which might not ordinarily seem connected. Using the product variation principles, we could let the system automatically generate a solution space for the user to explore. This means trying all possible combinations within reasonable parameters. A system with these characteristics could fall into the category of computational creativity systems that attempt to replicate human creativity (e.g. Colton and Wiggins, 2012).

Variation Filters

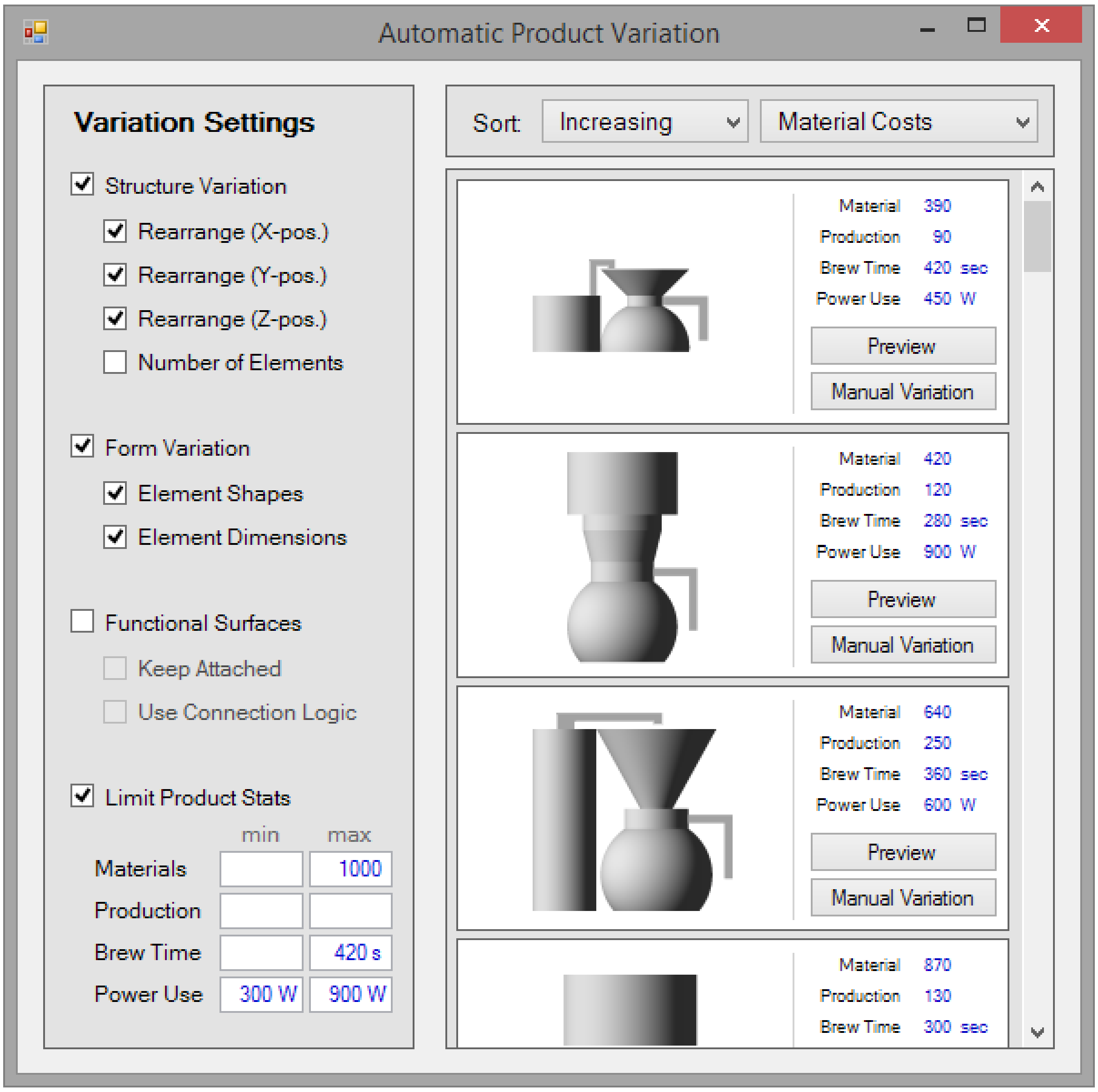
As discussed in the evaluation of the pilot application (*4.7.3*), for each new variation method, we add more degrees of freedom. An automatic variation environment like this would need a comprehensive filtering system which can be controlled by the user. These could include:

- Choosing to work with certain elements at a time.
- Choosing which types of variations to perform (arrangement, shape, dimensions, etc.).
- Preventing illogical relationships according to functional surface connections.
- Preventing physical element overlaps.
- Setting minimum and maximum limits to product stats.
- Sorting the solution space by product stats.
- Sorting the solution space by how much they differ from the basic structure.

This way, the user could choose to do precise exploration of outcomes, or allow the system to take its own questionable liberties. We could make a slightly farfetched comparison to the freely creative, dreamlike state of mind that results from human night-waking (Arnsten et al., 2012; Ekirch, 2006), discussed in *2.3.6*. Even if many of these generated solutions are unfeasible, the more radical combinations could aid user creativity by sparking new ideas.

Solution Space View

The user interface displays the solution space as a sortable list of products (see Figure *37*), with filtering options on the left side. Each preview also shows product stats, and has buttons that open full screen preview or the manual variation form to do further variations. Product stats could also be shown as a radar chart according to user preferences.



Variation Settings

- [✓] Structure Variation
  - [✓] Rearrange (X-pos.)
  - [✓] Rearrange (Y-pos.)
  - [✓] Rearrange (Z-pos.)
  - [ ] Number of Elements
- [✓] Form Variation
  - [✓] Element Shapes
  - [✓] Element Dimensions
- [ ] Functional Surfaces
  - [ ] Keep Attached
  - [ ] Use Connection Logic
- [✓] Limit Product Stats

	min	max
Materials		1000
Production		
Brew Time		420 s
Power Use	300 W	900 W

Sort: Increasing, Material Costs

Material	390	420	640	870
Production	90	120	250	130
Brew Time	420 sec	280 sec	360 sec	300 sec
Power Use	450 W	900 W	600 W	

Figure 37: Prototype GUI for automatic product variation

5.3 Genericity

5.3.1 Interoperability

The CAD import feature (*5.1.2*) would represent a significant contribution to genericity. This means that we would not necessarily have to program our basic structure in KBE. We could use other CAD tools to design products, or import already existing structures. This would also demand that our environment is in itself generic enough to handle arbitrary models.

5.3.2 Arbitrary Models

Dynamic User Interface

The user interface for product variation needs to dynamically adapt to the structure or element being edited. This means displaying all the dimensions that *can* be changed, and all basic geometrical shapes that are feasible choices.

Connecting Arbitrary Elements

The system should have some form of intelligence for connecting arbitrary elements together. This could mean that the system inherently understands certain component types, and would suggest pairings. Currently the user does all connections manually through functional surfaces. An interesting feature would be to have the system learn from the user, establishing an increasingly better understanding for what type of components typically go together.

General and Specific Product Stats

Some product stats can likely be calculated generically, such as material costs. More specific stats, such as the coffee maker brew time, would need to be developed for each special case.

5.4 Complex Products

Tjalve (2003) provides examples of moderately complex machines, but not for entire facilities such as industrial process plants. We seek to expand or adapt his principles for use with more complex products.

5.4.1 Number of Elements

Complex products have a significant amount of elements compared to a simple coffee maker. The original variation techniques were tested for low level variations, element by element. We would like to make these principles work for high level variations as well.

Product Sections

When doing form and structure variation, the user could choose to limit the variations to a selection of elements at a time, gradually finishing divisions of the product. It is also likely that such products are developed in teams. Each member or group could perform design variations on separate sections, while still being able to view other sections and take them into consideration. We could also have the ability to lock certain elements or sections, disabling any further editing or automatic variations there.

Group Operations

We could implement the ability to perform actions on entire product sections, for instance doing structure variation on sections by moving all elements as one group. This could be useful during an initial structural planning phase for the overall layout of an industrial facility.

Functional Surfaces

The functional surfaces in the pilot KBE environment were defined on single, function-specific areas of the product. For a complex product, we could also imagine large functional surfaces defined across multiple elements that serve as a group. This way, we could connect product sections with an overall connection type and direction. Whether using functional surfaces on low or high levels, we would argue that this concept scales well with increasing complexity.

Systems Thinking

The natural limits of human cognition come into play when thinking about large and complex systems (Goel, 2012). We could incorporate modeling frameworks into the KBE environment, such as OPM (Dori and Crawley, 2013) or SBF (Goel et al., 2009). This could help users get a better understanding of the system they are working on, and free more capacity for creative design. To get the most use of the modeling frameworks, the KBE environment could be able to create geometry and relationships from OPM or SBF models, and vice versa.

5.5 Other Aspects

5.5.1 User Interface Design

As discussed in *2.4.3*, HCI design principles may affect user creativity. The pilot application used the default generated AML interface with simple input mechanisms. Taking human cognitive aspects into consideration, it would be preferable to have a more progressive custom interface, designed in AML or otherwise. The prototype interfaces above attempt to improve some issues with the pilot interface.

Intuitiveness

The first issue is that the pilot interface is not intuitive enough to be used without reading a user manual, as Norman (2013) would recommend. A new user would not immediately know what is meant by structure variation and positioning multipliers, for instance.

Spacing and Grouping

Secondly, being met with an entire column of labels and textboxes is likely daunting to many users, and may trigger the "eyebrow response" (Kahneman, 2013). The common HCI design principle of spacing and grouping was applied in the pilot interface to improve the sense of belonging and make input sections easier to recognize (Norman, 2013; Preece, 1995). However, this was not enough to make input components stand out in the default AML layout. This is also likely to inhibit the creative process when users cannot immediately find the desired input component, which would require their *System II* to work instead of *System I* (Kahneman, 2013). A custom interface with frames and distinct headers improves separation. In addition, using different GUI components for each type of functionality makes them easier to locate. We can compare this to a well-designed TV remote control, where the volume button is clearly different from the other buttons.

Casual and Playful Creativity

The third issue is related to encouraging creativity through playfulness and curiosity (Törnkvist, 1998). This is perhaps more of a potential benefit of a good interface, rather than an actual issue (Norman, 2013). However, solving the former issues will provide some of these benefits as a byproduct. To improve this further, we should make the interface casual and "not feel like work", as Seth Rogen put it in *2.3.6* (Ferriss, 2015, ep 84).

6 Discussion

This chapter aims to answer the main research questions by structuring what has been learned from this study. In addition, we discuss and reflect upon the research study itself.

6.1 Research Questions

6.1.1 RQ 1: How can metacognition benefit KBE and creative design?

In our initial studies, we discovered branches of metacognition that could apply in a KBE setting or have implications for creativity in general. We defined the cognitive frameworks category to include systems of thought that could enable people or computer systems to improve design creativity. In addition, we investigated the art of reflective practice, and how it could improve the creative processes of people, or refine the effectiveness of existing computational creativity tools. The following figure suggests how metacognition and its derived concepts could relate to KBE and creativity (see Figure 38).

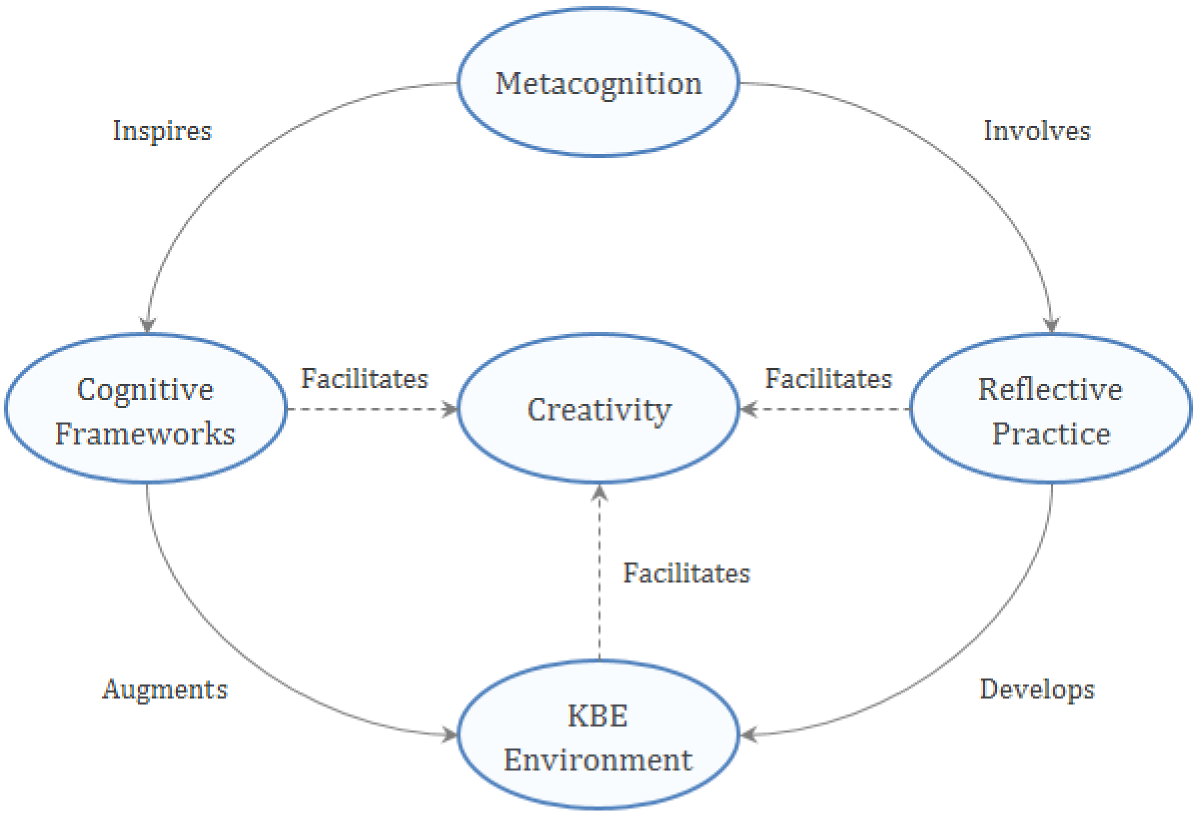


Figure 38: How metacognition relates to KBE and creativity

Cognitive Frameworks

The wording in Figure 38 may suggest that cognitive frameworks are exclusively beneficial for creativity. However, we found that cognitive frameworks can have both positive and negative constraints. Some constraints can also be positive and negative at the same time. The outcome depends on how the framework is used. For instance, a framework that is applied too strictly could limit freedom and produce negative constraints on creativity. Likewise, a framework with no boundaries could easily lead to derailment of the creative process.

The balance between freedom and restriction matters both for people who use cognitive frameworks as thought structure, and for computer systems that implement them. However, negative constraints in a computer system are more problematic, because once they are implemented, there is no way around them. On the other hand, when applying the same constraints in our minds, we can more easily cheat and escape the restraints if we notice that they are hindering our creative process.

Reflective Practice

We found that reflective practice allows for continuous development and refinement of existing creative tools and processes. With awareness of their own cognition and creative process, people can seek more optimal work routines, environments, times of day, and mental states in which they are the most creative. In the same way, people can reflect upon the tools they use for creative work, such as KBE environments, to consider whether they are optimal for what they want to achieve.

Having reflected upon our own pilot KBE environment, we found that our user interface had flaws that were strongly related to human cognition. Some HCI design principles were suggested for future development to mitigate or eliminate these issues. Our pilot evaluation also revealed that the cognitive framework that we had implemented offered both positive and negative constraints on creativity. Some prototype features for future development were suggested to shift the balance in favor of the positive aspects.

This is demonstrated by the concepts of reflection-*in*-action and reflection-*on*-action which were introduced by Schön (1983). These concepts have also been applied in software engineering to understand the importance of reflective practice in system development (Mathiassen, 1998).

Reflection-*in*-action refers to experiences and ideas that provide inputs to our actions in a learning situation. With respect to creativity, this can be stimulated if we have an intuitive user interface. On the other hand, reflection-*on*-action occurs after an experience that is perceived as complex or uncertain. For instance, a designer that has had problems with designing a new product in KBE, would need post-reflection before new actions can be taken. The designer has then become a reflective actor who combines his existing knowledge with new knowledge into a retrospective reflection. Because of this, reflection could be a vital activity in creative design, effectively turning experiences into explicit benefits.

Branching of Cognitive Frameworks under Metacognition

Within section *2.4*, we introduce some systems of thought that we label *cognitive frameworks*. Branching this under metacognition may have been slightly incorrect, because strictly speaking, not all of these frameworks really *apply* metacognition. However, we reckon they may have been uncovered through reflecting upon existing processes or a desire to formalize systems of thought into modeling frameworks.

6.1.2 RQ 2: How can KBE technologies support the creative design phase?

Through our initial study of KBE technologies, we found that some of the native advantages that KBE provides could benefit creative design. We highlighted that engineering design efforts typically consist of 80% routine and 20% innovation (Sanya and Shehab, 2014; Skarka, 2007). It is possible that KBE could shift this balance in favor of innovation through the automation of routine tasks. In addition, the knowledge management capabilities of KBE could allow knowledge to be combined in a creative manner, or apply existing knowledge to new designs by analogy.

Implementation of Systematic Design Frameworks

As part of this study, we created a cognitive framework based on the principles of Tjalve (2003), and were able to implement them in a KBE environment. Specifically, we start with Tjalve's *basic structure* and use *structure variation*, *form variation*, and *functional surfaces* to generate a product solution space from the basic structure. This way we have embedded a set of design constraints that allow us to steer and support the creative process of a user. Although these constraints have both positive and negative aspects, it shows that KBE enables us to structure the creative design phase and facilitate a certain kind of creativity.

The fact that a KBE technology such as AML can successfully implement a cognitive framework is a good indicator in itself. This leads us to believe that other types of frameworks could be implemented as well, tailoring the creative support as required. The following discussions on RQ 3 and RQ 4 mention other frameworks and theoretical considerations that are also valid for this question.

6.1.3 RQ 3: How can Tjalve's principles be implemented with KBE for a simple product?

Tjalve's Principles as a Cognitive Framework for KBE

Through our study of Tjalve (2003), we extracted a set of principles that seemed applicable to KBE and creative design. These principles make up what we shall refer to as *Tjalve's framework*, which we test-implemented in a pilot KBE environment. The entire implementation process is elaborated in chapter *4*. Tjalve's framework structures some of the mental creative processes, and applies systematic design constraints that can facilitate and inhibit creativity. The following figure explains how this cognitive framework relates to creativity (see Figure 39).

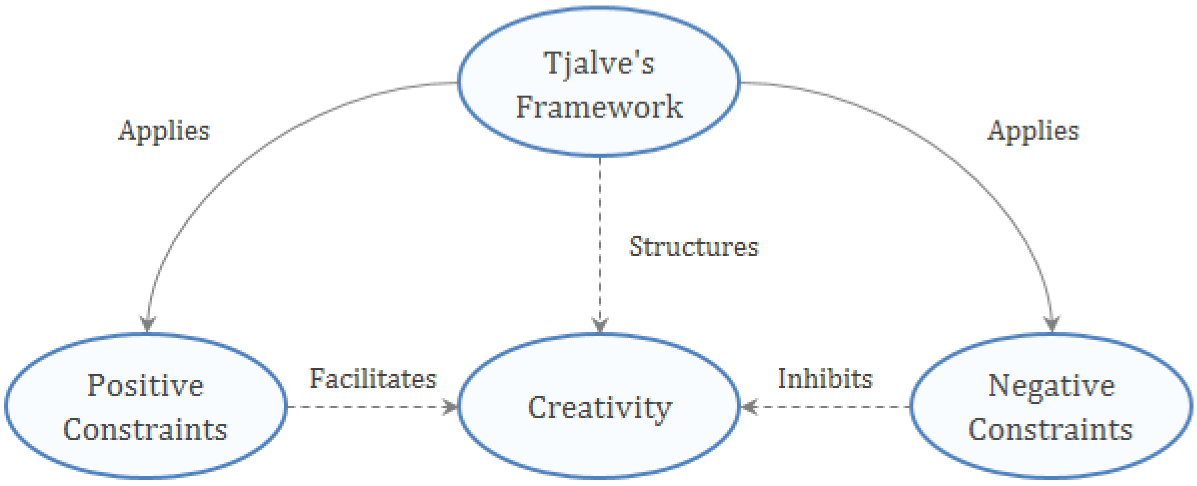


Figure 39: Tjalve's principles as a cognitive framework

The following sections will elaborate more on this figure, and aims to answer the questions asked in *1.2.1*:

a) *What are the systematic design constraints applied by the chosen framework?*
b) *How do the framework constraints affect creativity?*
c) *How feasible were the framework constraints to implement in KBE?*

a) Tjalve's Framework – Constraints

Tjalve's framework applies the following constraints on the creative design, as established in *4.7.2*. These constraints have positive and negative aspects, which can facilitate or inhibit creativity (as seen in Figure 39 above).

1) The product *basic structure* is decided.
2) Elements can be rearranged and multiplied with *structure variation*.
3) Elements can be assigned new geometry and dimensions with *form variation*.
4) Variations must consider *functional surfaces* to retain product functionality.

b) Tjalve’s Framework – Effects on Creativity

Having evaluated the pilot KBE environment, we got a better impression of how each constraint affected our creative process. The following table highlights the positive and negative aspects of the framework constraints. We have attempted to make this a conceptual evaluation, meaning that we try to look beyond how well or to what degree each constraint was implemented in the pilot. Some of the reasoning behind these results are covered in the pilot evaluation in *4.7.2*.

Constraint	Effects on Creativity
1) Basic Structure	Positive: Effective at steering the trajectory of the creative design process. Negative: Prevents ideas that would change *how* the product performs its function, i.e. altering the sub-functions that together fulfill the main function. We cannot return to an earlier stage of our product synthesis.
2) Structure Variation	Positive: Can inspire new and radical element combinations. A good support to fall back on when creative flow is poor. Negative: No adverse effects were found.
3) Form Variation	Positive: Experimenting with geometries and dimensions inspires completely different aesthetic designs. Changing these parameters can also spawn new arrangement ideas that were previously not possible. This creates an effective synergy with structure variation. Negative: No adverse effects were found.
4) Functional Surfaces	Positive: Easier to understand and visualize the product as a system of functional relationships. This frees more capacity for creative efforts with form and structure variation. Negative: The functional surfaces restrict how we can do form and structure variation. However, this is necessary to retain functionality and adhere to the basic structure.

c) Tjalve's Framework – Implementation Feasibility

We developed the pilot KBE environment using the Adaptive Modeling Language (AML) by TechnoSoft Inc. In terms of feasibility, we consider the implementation of Tjalve's framework in KBE to be an overall success. However, some interface-related features required more development time than anticipated. These were reduced to default input mechanisms or instead predefined for the user. The following table summarizes the implementation feasibility of the framework constraints in a KBE environment using AML. This reasoning behind these results are covered in *4.6*.

Constraint	Implementation Feasibility
1) Basic Structure	Implementation: Feasible Development time: Medium Basic geometry and element relationships were predefined for the user, setting natural boundaries for any further creative work.
2) Structure Variation	Implementation: Feasible Development time: High Elements were positioned relative to each other by applying an adjacency formula to their coordinate systems. Parent classes with multiple element instances were created to support a variable number of elements.
3) Form Variation	Implementation: Feasible Development time: Medium Different element classes were created for each new geometrical shape. Element dimensions and shapes were then exposed to the user through the default AML interface.
4) Functional Surfaces	Implementation: Partial Success (Feasible) Development time: Medium We created classes to represent functional surfaces and their relationships. These classes support the definition of arbitrary relationships, but the custom user interface was discontinued. Virtual functional surfaces were predefined for the user instead.

Further Comments on the Effects on Creativity

Given the fact that the pilot KBE environment was not tested with product designers, it is fully possible that the environment in its current state does not *improve* creativity, but merely structures it into a system. We could argue that the environment does not *impede* creativity either. During testing, ideas would still emerge that existed outside of the solution space enforced by the framework. A trained creative person could have routines for systematically capturing ideas that emerge (Epstein et al., 2008). The issue then might be that these ideas are not allowed to flourish and mature as they would in an unrestricted environment, for instance while sketching ideas on paper.

This brings us to the thought that the systematic design principles introduced by Tjalve (2003) are indeed very systematic and engineer-like, but does not take into account the potentials of unhindered creativity (e.g. Asimov, 1959; Ferriss, n.d., 2015, ep 82; Törnkvist, 1998). Based on these thoughts, one could argue that a creative environment should not rigidly enforce a such principles, and at the very least grant users the liberty to discard the framework if they notice that it is hindering their creative output.

Human Computer Interaction and Creativity

Throughout our study, we discovered that there could be a strong connection between the fields of Human Computer Interaction and creativity. We base this notion on the results of evaluating how the pilot KBE environment affected creativity, and connecting our separate learnings on human interface design, cognition and creativity (e.g. Brown, 1987; Kahneman, 2013; Norman, 2013; Preece, 1995). However, more research would be needed to determine whether there is a connection between HCI and creativity.

Assuming there is such a relation, it would be fair to assume that some aspects of user interface design are up to the preferences of the individual user, for instance colors and fonts. However, the more proven HCI design principles could guide the development of objectively better interfaces for creative design (e.g. Norman, 2013; Preece, 1995).

Potential Use of Modeling Techniques

During the study, we uncovered various modeling techniques that could be suited to represent a product structure and its functional relationships. Already mentioned in *2.4.2*, are OPM (Dori and Crawley, 2013) and SBF (e.g. Goel et al., 2009). These were also suggested for future development in *5.4*.

In addition, we looked into the possibility of using Oriented Networks to represent the functional relationships of a product (Sheth, 1972). This modeling technique uses a network of nodes with directional connections. We considered using it in conjunction with the concept of functional surfaces by Tjalve (2003). However, due to the simple, linear nature of the coffee maker example, this technique was not used in the pilot KBE development.

Alternative Workflow

In *4.2.2*, we established a workflow where functional surfaces were defined and connected before doing structure variation. This is not strictly in accordance with the product synthesis proposed by Tjalve (2003), and could introduce some challenges. Originally, functional surfaces are a part of form variation, which is done after structure variation. Having already defined functional surfaces and their relationships, our options for rearranging and multiplying elements become restricted. However, as suggested in *5.1.1*, these functional surface relationships allow us to implement things like automatic pipes and pumps as required. This increases the development cost, but eventually gives us a more powerful KBE environment that can retain product functionality while doing unrestricted structure variation.

The Origin of Positive and Negative Constraints

The word constraint arguably has an inherent negative feel to it. However, in order to discuss the good and bad aspects of cognitive frameworks, we established the terms *positive constraints* and *negative constraints*. By positive constraints we mean boundaries that eventually yield some form of gain, for instance more focused creative efforts. Negative constraints would have the opposite effect, for instance limiting creative outputs.

In an attempt to formalize this concept, we looked for other studies that had used similar terms. Our initial search showed few relevant results for positive constraints, except for occasional mentions on the podcast of Tim Ferriss (e.g. Ferriss, 2015, ep 83), and in various blogs[18, 19, 20]. It is possible that the concept itself has been discussed in previous research, but with different terminology. Further study would be required to determine whether this is a previously explored concept with respect to creativity.

The Role of Knowledge Management

Throughout our study, the concept of Knowledge Management (KM) emerged as an interesting perspective on both KBE and creativity. We discovered that tacit knowledge in particular, has challenges associated to its conversion and embedment into knowledge based systems (e.g. Alavi and Leidner, 2001; Nonaka et al., 2000; Rocca, 2012). Combined with the fact that creativity could stem from tacit knowledge and experiences (Asimov, 1959; Nonaka and Konno, 1998), we wanted to investigate whether this could be relevant for implementing KBE systems that support or emulate human creativity. Although we did uncover interesting relationships between the fields of KM, KBE and creativity, we could not make any solid conclusions without further study. Therefore, we did not include KM as part of our main inspirations for the development of the pilot and future KBE environments.

Blog mentions of positive constraints:
[18] http://jamesclear.com/dr-seuss
[19] http://thinkspace.com/the-power-of-positive-constraints/
[20] http://innovationexcellence.com/blog/2013/04/01/the-power-of-positive-constraints/

6.1.4 RQ 4: How can this environment be adapted to work for complex products?

In chapter *5*, we suggested features and prototype interfaces for a potential generic KBE environment for doing creative work with complex products. These proposals were based on the evaluation of the pilot KBE environment and other findings in our study. However, we would highlight that many of these suggestions also bear the subjective assessments of the author. As such, until further research is conducted, we cannot not make any solid claims regarding the feasibility or effectiveness of the suggested features.

Prototype User Interfaces

We proposed an updated version of the product variation form that applies Tjalve's framework, which would allow users to more easily perform structure and form variation (Tjalve, 2003). Based on Tjalve's framework, we also suggested an automatic product variation form, which would produce an entire solution space of products, controlled by filters chosen by the user. We argued that such an environment would converge towards the realm of computational creativity, attempting to emulate human level creativity (Colton and Wiggins, 2012). This automated exploration of the product solution space was inspired by the principles of Tjalve (2003), combined with the notion that great ideas often stem from discovering connections between things that ordinarily do not seem connected (Asimov, 1959).

Genericity and Complex Products

To create a more generic environment, we suggested features such as interoperability through the import of industry standard files to do creative work in KBE, intelligent connection of arbitrary elements, and a more dynamic interface for product variation.

To have a system that can scale with complex products, we proposed doing variations with entire sections of the product, definition of overall functional surfaces to link product sections, and incorporating systems thinking modeling frameworks such as OPM or SBF (Dori and Crawley, 2013; Goel et al., 2009).

Meta Thinking and Analogical Thinking

In *2.4.2*, we mentioned meta thinking and analogical thinking as cognitive frameworks for structuring design processes. These could be options to consider for future development, but the underlying principles do not seem to pair well with our chosen principles by Tjalve (2003). Analogical thinking (Cross, 1997; Dorst, 2011; Goel, 2012) seems to resonate well with knowledge conversion (Nonaka and Konno, 1998) and the potential for KBE environments to reuse existing knowledge in new designs (Verhagen et al., 2012). Meta thinking (Goel, 2012), on the other hand, would inspire systems that rely on artificial intelligence and control knowledge to perform self-diagnostics and adaptation of processes (Calkins et al., 2000).

6.2 Limitations and Suggestions for Improvement

6.2.1 Research Scope

The originally suggested thesis assignment was focused on incorporating the principles of Tjalve (2003) into a KBE environment for creative design. With the addition of metacognition, the research scope widened, and also led to a greater focus on the cognitive aspects of creativity. This ultimately led to a more diverse, but challenging thesis. We believe that the original scope would have allowed for a more in-depth research on KBE, and possibly to have experimented with a working generic KBE environment using Tjalve's principles. Despite these limitations, we hope that the broader scope has uncovered some new perspectives with potentials for additional research. These could include how knowledge and KM relates to KBE and creative design, the effect that positive constraints have on creativity, and how HCI design principles can affect human creativity.

6.2.2 Theory and Practice

This study involved a combination of theoretical research and software development. The latter could be seen as more practical than theoretical. However, the pilot KBE environment was ultimately developed as a proof of concept with the theoretical research in mind, and not for immediate use in a specific industry. In light of former development experiences within other industries, we would argue that there is often a notable difference between theory and practice when it comes to software development and its integration into organizations. In retrospect, we would have preferred to incorporate some more practical aspects into our research and development, knowing that what is found in theory might come with unexpected challenges when attempted in practice later.

6.2.3 Evaluation Process of the Pilot KBE Environment

The findings in *4.6* and *4.7* were attained by the developer alone, who is neither a product designer by profession, nor the end user of the application. This compromises the objectivity and quality of the test results. Including the end users in the process would yield a more solid evaluation of the effects on creativity, and would provide invaluable feedback on potentials for improvement. The Scandinavian tradition of Participating Design includes the users in the design process of system development (Ehn, 1993). A collective resource approach suggested by Ehn (1993) with participating design in iterations could improve the evaluation stage (Schuler and Namioka, 1993).

6.3 Discussion on the AML Development Environment

Rocca (2012) mentions how some find a programming language to be tedious compared to simply using a graphical user interface. However, he also points out several advantages of having an underlying language, and that it is even necessary to achieve certain advanced capabilities. Some cases where a KBE language is required or advantageous: when we need to capture the design *intent* or *process*, when we need to define a *consistent* automated generative process, and when we need to handle custom interoperability with non-standard file formats (Rocca, 2012, p. 173).

Rocca (2012) gives further support to KBE, which he says has been labeled by many as "just CAD systems", and highlights that, in fact, some KBE technologies are merely CAD systems with augmented KBE-like capabilities. He argues that the Adaptive Modeling Language (AML) by TechnoSoft, and the General-purpose Declarative Language (GDL) by Genworks, are "the only true KBE systems on the market".

Assuming that having a powerful language is one of the key advantages of KBE technologies, we would argue that the language and its facilities would be a vital area of focus for the providers of KBE development environments. As mentioned in *3.3*, we used both AML XEmacs and Notepad++ to develop the pilot KBE environment. The reason for supplementing with Notepad++, was that it had certain assistive features that led to a faster and less error prone coding experience. This may be wholly subjective, as the preferred choice of code editor varies greatly in the software development community. Even so, given our experience from this, and other development projects, we would like to suggest some key features that we think would enhance the AML XEmacs environment. It is possible that some of these features already exist without our knowledge. We would also like to propose that some of these features would yield benefits for the developer, referring to HCI design principles (Norman, 2013; Preece, 1995), and metacognitive aspects (Brown, 1987; Kahneman, 2013) which were identified in our theoretical studies. These benefits include faster code writing, less routine work, reduced cognitive load, early error catching, and ultimately being able to redirect these spared efforts into coding. We propose the following:

1) Auto-completion of text would allow faster code output, less typing errors, and not having to remember exact names of already declared classes and properties.
2) The editor could highlight other code that is relevant to the currently selected term, for instance a class name or a declared property.
3) The editor could highlight the target of long references such as `(the superior superior ...)`, when possible to determine.
4) The environment code-to-test-cycle could be shortened so that the developer could more quickly view the results of minor code changes.

7 Conclusion

This study investigates the possibility of using Knowledge Based Engineering (KBE) as a tool for the early creative design phase, and examines how the concept of metacognition could relate to KBE and creativity. Design research was used as an overall methodological approach as a basis for the research and development processes.

A literature review on KBE and creativity revealed that KBE can facilitate creativity by automating routine tasks in favor of innovative efforts, and by exploration of product solution spaces via parametrization and generative procedures. A study of Eskild Tjalve's work suggested that his design methodology could be implemented in KBE to support the creative design phase. A literature review on metacognition suggested that aspects such as reflective practice play a role in the creation, evaluation, and refinement of cognitive frameworks that can ultimately be suitable for implementation in systems for computational creativity.

A pilot KBE environment was developed in AML using Tjalve's systematic design principles. The resulting tool could perform Tjalve's design variations on a simple product, by rearranging or multiplying components, and by changing the dimensions or base geometries of components. Tjalve's functional surfaces could be defined on specific areas of components and connected with functional surface on other components, effectively defining their functional relationships. It was determined that all of the chosen principles from Tjalve were feasible to implement using a KBE technology with a strong modeling language such as AML.

An evaluation of the KBE environment found that Tjalve's framework imposed positive and negative constraints on creativity. The positive constraints facilitated creativity by guiding the trajectory of the design process, and by providing a set of design principles to fall back to when the creative flow is poor. The negative constraints inhibited creativity when the framework was unable to reproduce the ideas of the designer. Other negative aspects were identified as being related to Human Computer Interaction (HCI) and human cognition.

A future development plan was proposed to advance the pilot KBE environment. Prototype interfaces were suggested to demonstrate how HCI design principles could mitigate the associated negative effects on creativity. Live updated product stats would allow the tool to better compete with traditional methods for mockup design. The concept of interoperability and handling of arbitrary models would promote a generic environment. Finally, suggestions were made for how to scale Tjalve's design principles to work for complex products.

The paper has demonstrated that KBE technologies with powerful language capabilities are well suited to incorporate a systematic design methodology as a framework to guide the early creative design process. The resulting effects on user creativity is largely dependent on the positive and negative constraints applied by the framework, and how strictly these constraints are enforced.

8 Suggestions for Further Work

This chapter offers propositions for further work. The primary focus should be on 1) further development of the KBE environment, and 2) the use of participating design to reveal more potential effects on creativity. Secondary aspects to investigate include 3) getting a deeper understanding of how metacognition can benefit creative design, and 4) how HCI design principles can benefit human creativity.

8.1 Development of the KBE Environment for Creative Design

The next logical step after this study is to consider the suggestions in *5 Future Development,* and conduct further research and development of a KBE environment for the early creative design phase using Tjalve's framework. First, this adaptation should focus on genericity and the ability to handle arbitrary product models. Secondly, the environment should be aimed towards more complex products, such as industrial process plants.

8.2 Participating Design for the KBE Environment

A limitation of this study was that the potential end users of the KBE environment were not included in the testing and evaluation processes. Further research along with for instance product designers is needed to more accurately determine the effects that this tool has on creativity. The Scandinavian concept of Participating Design may be of relevance here.

8.3 Metacognition and Creativity

The study discovered potential ways to improve human creativity through reflective practice, which involves refinement of existing creative processes, adaptation of work environments to encourage casual and self-motivated idea generation, and evaluation of existing tools for creative design to uncover flaws and potentials for improvement.

The concept of cognitive frameworks can be explored further with respect to the refinement of creative processes and to their applicability in tools for computational creativity.

8.4 Human Computer Interaction and Creativity

The study revealed potential close ties between Human Computer Interaction (HCI), human cognition, and creativity. This could have implications for not only KBE and creative design, but for the development of computational creativity tools in general. Further literature reviews and research must be conducted in order to substantiate this notion.

9 References

Alavi, M., Leidner, D.E., 2001. Review: Knowledge Management and Knowledge Management Systems: Conceptual Foundations and Research Issues. MIS Q 25, 107–136.

Arnsten, A.F., Wang, M.J., Paspalas, C.D., 2012. Neuromodulation of Thought: Flexibilities and Vulnerabilities in Prefrontal Cortical Network Synapses. Neuron 76, 223–239.

Asimov, I., 1959. Isaac Asimov Asks, "How Do People Get New Ideas?" MIT Technol. Rev.

Brown, A., 1987. Metacognition, Executive Control, Self-Regulation and Other More Mysterious Mechanisms. In F. E. Weinert & R. H. Kluew (Eds.), Metacognition, Motivation and Understanding (pp. 65-116), The Psychology of education and instruction. L. Erlbaum Associates, Hillsdale, N.J.

Calkins, D.E., Egging, N., Scholz, C., 2000. Knowledge-Based Engineering (KBE) Design Methodology at the Undergraduate and Graduate Levels. Int. J. Eng. Educ. 16, 21–38.

Cascini, G., 2008. Computer-aided innovation (CAI). Int. Fed. Inf. Process., IFIP International Federation for Information Processing 277, 7–18.

Colton, S., Wiggins, G.A., 2012. Computational Creativity: The Final Frontier? Front. Artif. Intell. Appl. 21–26. doi:10.3233/978-1-61499-098-7-21

Cross, N., 1997. Descriptive models of creative design: application to an example. Des. Stud., Descriptive models of design 18, 427–440.

Dahl, J., Hill, S., Chemaly, A., 2006. AMRaven - An Integrated Air Vehicle Design and Analysis Environment.

Davenport, T.H., Prusak, L., 1998. Working knowledge: how organizations manage what they know. Harvard Business School Press, Boston, Mass.

DeSanctis, G., Poole, M.S., 1994. Capturing the Complexity in Advanced Technology Use: Adaptive Structuration Theory. Organ. Sci. 5, 121–147.

Dingsøyr, T., 2002. Knowledge Management in Medium-Sized Software Consulting Companies (Doctoral Thesis). Norwegian University of Science and Technology, Tapir, Norway.

Dori, D., Crawley, E.F., 2013. Object-process methodology: a holistic systems paradigm, softcover repr. of the hardcover 1. ed. 2002. ed. Springer, Berlin.

Dorst, K., 2011. The core of "design thinking" and its application. Des. Stud. 32, 521–532.

Ehn, P., 1993. Scandinavian design: On participation and skill. ResearchGate.

Ekirch, A.R., 2006. At day's close: night in times past, 1. ed., 1. publ. as a Norton paperback. ed. Norton, New York.

Epstein, R., Schmidt, S.M., Warfel, R., 2008. Measuring and Training Creativity Competencies: Validation of a New Test. Creat. Res. J. 20, 7–12.

Ferriss, T., 2015, ep 82. Podcast - The Tim Ferriss Show, Scott Adams: The Man Behind Dilbert.

Ferriss, T., 2015, ep 83. Podcast - The Tim Ferriss Show, The Scariest Navy SEAL Imaginable…And What He Taught Me.

Ferriss, T., 2015, ep 84. Podcast - The Tim Ferriss Show, Comedy's Dynamic Duo, Seth Rogen and Evan Goldberg.

Franken, R.E., 1994. Human motivation, 3rd ed. ed. Brooks/Cole Pub. Co, Pacific Grove, Calif.

Gero, J., 2013. Understanding Designing: What Are Designers Doing When They Design (Video Presentation, https://youtu.be/TRiETj2ujvQ).

Goel, A.K., 2012. Analogical Thinking, Systems Thinking, Visual Thinking and Meta Thinking: Four Fundamental Processes of Design Creativity. Proc. Des. Soc. SIG Des. Creat. Workshop Coll. Stn. Tex.

Goel, A.K., Rugaber, S., Vattam, S., 2009. Structure, behavior, and function of complex systems: The structure, behavior, and function modeling language. Artif. Intell. Eng. Des. Anal. Manuf. 23, 23.

Harris, S., 2014. Waking up: a guide to spirituality without religion, First Simon & Schuster hardcover edition. ed. Simon & Schuster, New York.

Hays, G., 2002. Marcus Aurelius: Meditations. Modern Library, New York.

Hevner, A.R., March, S.T., Park, J., Ram, S., 2004. Design science in information systems research. MIS Q. Manag. Inf. Syst. 28.

Howard, T.J., Culley, S.J., Dekoninck, E., 2008. Describing the creative design process by the integration of engineering design and cognitive psychology literature. Des. Stud. 29, 160–180.

Kahneman, D., 2013. Thinking, fast and slow, 1st pbk. ed. ed. Farrar, Straus and Giroux, New York.

Larman, C., 2004. Agile and iterative development: a manager's guide, Agile software development series. Addison-Wesley, Boston, Mass.

Mathiassen, L., 1998. Reflective systems development. Scand. J. Inf. Syst. 10, 12.

Nonaka, I., Konno, N., 1998. The Concept of "Ba": Building a Foundation for Knowledge Creation. Calif. Manage. Rev. 40, 40–54.

Nonaka, I., Toyama, R., Konno, N., 2000. SECI, Ba and Leadership: A Unified Model of Dynamic Knowledge Creation. Long Range Plann. 33, 5–34.

Norman, D.A., 2013. The design of everyday things, Revised and expanded edition. ed. Basic Books, New York, New York.

Pinfold, M., Chapman, C.B., 2001. The application of a knowledge based engineering approach to the rapid design and analysis of an automotive structure. Adv. Eng. Softw. 32, 903–912.

Polanyi, M., 1966. The Tacit Dimension. Routledge, London.

Preece, J., 1995. Human-computer interaction. Addison-Wesley Pub. Co, Wokingham, England; Reading, Mass.

Rocca, G.L., 2012. Knowledge based engineering: Between AI and CAD. Review of a language based technology to support engineering design. Adv. Eng. Inform. 26, 159–179.

Ryle, G., 1949. The Concept of Mind. Penguin Books, Ltd, London.

Sanya, I.O., Shehab, E.M., 2014. An ontology framework for developing platform-independent knowledge-based engineering systems in the aerospace industry. Int. J. Prod. Res. 52, 6192–6215.

Schön, D.A., 1983. The Reflective Practitioner: How Professionals Think in Action. Basic Books.

Schraw, G., 1998. Promoting general metacognitive awareness. Instr. Sci. 26, 113–125.

Schuler, D., Namioka, A. (Eds.), 1993. Participatory Design: Principles and Practices. L. Erlbaum Associates Inc., Hillsdale, NJ, USA.

Sein, M.K., Henfridsson, O., Purao, S., Rossi, M., Lindgren, R., 2011. Action Design Research. MIS Q 35, 37–56.

Sheth, P.N., 1972. A Digital Computer based Simulation Procedure for Multiple Degree of Freedom Mechanical Systems with Geometric Constraints (Doctoral Thesis). University of Wisconsin.

Skarka, W., 2007. Application of MOKA methodology in generative model creation using CATIA. Eng. Appl. Artif. Intell. 20, 677–690.

Skogsfjord, M.B., Rognseth, C., 2014. Parametric Ship Hull Design in NX. Institutt for produktutvikling og materialer.

Tjalve, E., 2003. Systematic Design of Industrial Products. Institute for Product Development, Technical University of Denmark, Lyngby.

Törnkvist, S., 1998. Creativity: Can It Be Taught? The Case of Engineering Education. Eur. J. Eng. Educ. 23, 5–12.

Verhagen, W.J.C., Bermell-Garcia, P., van Dijk, R.E.C., Curran, R., 2012. A critical review of Knowledge-Based Engineering: An identification of research challenges. Adv. Eng. Inform., Network and Supply Chain System Integration for Mass Customization and Sustainable Behavior 26, 5–15.

Wagner, T., Compton, R.A., 2012. Creating innovators: the making of young people who will change the world, 1st Scribner hardcover ed. ed. Scribner, New York.

Walsham, G., 2001. Knowledge Management: The Benefits and Limitations of Computer Systems. Eur. Manag. J. 19, 599–608.

Willis, A.M., 2006. Ontological Designing. Des. Philos. Pap. 4, 69–92.

Yanow, D., Tsoukas, H., 2009. What is Reflection-In-Action? A Phenomenological Account. J. Manag. Stud. 46, 1339–1364.

Appendix A – Source Code

A.1 Coffee Maker Pilot

```
;==========================================================================
;
;   Module:        Coffee Maker Pilot
;   Version:       1.0.3.2
;   Date:          2016.02.08
;   Authors:       Eivind A. Taftø
;   Description:   This pilot was made as part of a master's assignment at
;                  the Norwegian University of Science and Technology.
;                  The thesis explores the use of Knowledge Based
;                  Engineering technology to support an early creative
;                  design phase. The purpose of this pilot was to
;                  test-implement theoretical findings from Systematic
;                  Design of Industrial Products, by Tjalve (2003).
;                  These techniques include systematic variation
;                  of element arrangement, number of elements,
;                  element dimensions, element geometrical shapes,
;                  as well as creation of virtual functional surfaces that
;                  represent functional relationships between elements.
;
;==========================================================================

(in-package :aml)


;--------------------------------------------------------------------------
;   Function:      arrange
;   Description:   Calculates a relative arrangement coordinate for two
;                  elements. The multiplier determines the spacing between
;                  them, where 0.0 is no offset, and 1.0 is exact adjacency.
;   Reference:     Functional Requirement F 2.1
;                  Structure Variation (relative arrangement)
;--------------------------------------------------------------------------
(defun arrange (multiplier len-a len-b)
   (* multiplier (* 0.5 (+ len-a len-b)))
)


;--------------------------------------------------------------------------
;   Function:      inner-diameter
;   Description:   Calculates diameter excluding wall thickness
;--------------------------------------------------------------------------
(defun inner-diameter (diameter thickness)
   (- diameter (* 2.0 thickness))
)
```

```
;------------------------------------------------------------------------
;   Class:          functional-surface
;   Description:    General mixin class for functional surfaces.
;   Reference 1:    Functional Requirement F 1.1
;                   Functional Surfaces (define)
;------------------------------------------------------------------------
(define-class functional-surface
   :inherit-from (object)
   :properties (

   )
)


;------------------------------------------------------------------------
;   Class:          functional-surface-link
;   Description:    Defines a connection between two functional-surface.
;                   The type and direction properties further specify
;                   the functional relationship.
;   Reference 1:    Functional Requirement F 1.2
;                   Functional Surfaces (connect)
;------------------------------------------------------------------------
(define-class functional-surface-link
   :inherit-from (object)
   :properties (
        functional-surface-a    nil
        functional-surface-b    nil
        connection-type         nil ; gravity, stream, contact, none
        connection-direction    nil ; a-to-b,  b-to-a, two-way, none
   )
)
```

```
;---------------------------------------------------------------------------
;   Class:          functional-surface-disc
;   Description:    Defines a functional surface in the form of a disc.
;                   The surface can optionally be projected above its origin.
;   Reference 1:    Functional Requirement F 1.1
;                   Functional Surfaces (define)
;---------------------------------------------------------------------------
(define-class functional-surface-disc
   :inherit-from (functional-surface)
   :properties (
        height    (default 1.0)
        diameter  (default 1.0)
        color     (default 'blue)
        origin    (default (list 0.0 0.0 0.0))
        display?  (default t)
        reference-coordinate-system nil
   )
   :subobjects (

        ; Coordinate System
        (cs :class 'coordinate-system-class
             reference-coordinate-system (default)
             display?  nil
             origin    (default)
        )

        ; Virtual Surface Disc
        (surface :class 'disc-object
             reference-coordinate-system ^^cs
             diameter  (default)
             color     (default)
             render    'shaded
             display?  (default)
        )

        ; Surface Projection
        (projection :class 'cylinder-object
             reference-coordinate-system ^^cs
             height    (default)
             diameter  (default)
             color     (default)
             render    'boundary
             display?  (default)
             orientation (list (translate (list 0.0 0.0 (* 0.5 ^height))))
        )
   )
)
```

```
;-----------------------------------------------------------------------------
;  Class:        functional-surface-disc-2x2
;  Description:  Defines a 2 by 2 grid of functional-surface-disc.
;  Reference 1:  Functional Requirement F 1.1
;                Functional Surfaces (define)
;  Reference 2:  Functional Requirement F 2.2
;                Structure Variation (number of elements)
;-----------------------------------------------------------------------------
(define-class functional-surface-disc-2x2
   :inherit-from (functional-surface)
   :properties (
        height    (default 1.0)
        diameter  (default 1.0)
        color     (default 'blue)
        spacing   0.0
        d         (- (* 0.5 ^diameter) ^spacing)
        offset    (+ (arrange 0.5 ^d ^d) ^spacing)
        origin    (default (list 0.0 0.0 0.0))
        display?  (default t)
        reference-coordinate-system nil
   )
   :subobjects (
        (cs :class 'coordinate-system-class
            reference-coordinate-system (default)
            display? nil
            origin   (default)
        )
        (fs-1 :class 'functional-surface-disc
            diameter ^^d
            origin (list ^^offset ^^offset 0.0)
            reference-coordinate-system ^^cs
        )
        (fs-2 :class 'functional-surface-disc
            diameter ^^d
            origin (list (- ^^offset) ^^offset 0.0)
            reference-coordinate-system ^^cs
        )
        (fs-3 :class 'functional-surface-disc
            diameter ^^d
            origin (list ^^offset (- ^^offset) 0.0)
            reference-coordinate-system ^^cs
        )
        (fs-4 :class 'functional-surface-disc
            diameter ^^d
            origin (list (- ^^offset) (- ^^offset) 0.0)
            reference-coordinate-system ^^cs
        )
   )
)
```

```
;---------------------------------------------------------------------------
;   Class:         filter
;   Description:   Defines a single Filter element. Its geometry is based on
;                  the truncated-cone-object, using difference-object to
;                  provide wall thickness.
;   Reference 1:   Functional Requirement F 3.2
;                  Form Variation (dimensions)
;---------------------------------------------------------------------------
(define-class filter
   :inherit-from (difference-object)
   :properties (
        b-h            (default 1.0)
        b-d1           (default 1.0)
        b-d2           (default 2.0)
        wall-t         (default 0.25)
        object-list    (list ^outer ^inner)
        reference-coordinate-system ^cs
   )
   :subobjects (
        (outer :class 'truncated-cone-object
             height              ^b-h
             start-diameter      ^b-d1
             end-diameter        ^b-d2
             display?            nil
        )
        (inner :class 'truncated-cone-object
             height              ^b-h
             start-diameter      (inner-diameter ^b-d1 ^wall-t)
             end-diameter        (inner-diameter ^b-d2 ^wall-t)
             display?            nil
        )
   )
)
```

```
;---------------------------------------------------------------------------
;  Class:         filter-2x2
;  Description:   Defines a 2 by 2 grid of Filter elements. The individual
;                 element properties can be changed collectively from here.
;  Reference 1:   Functional Requirement F 2.1
;                 Structure Variation (relative arrangement)
;  Reference 2:   Functional Requirement F 2.2
;                 Structure Variation (number of elements)
;  Reference 3:   Functional Requirement F 3.2
;                 Form Variation (dimensions)
;---------------------------------------------------------------------------
(define-class filter-2x2
   :inherit-from (object)
   :properties (
        b-h       (default 1.0)
        b-d1      (default 1.0)
        b-d2      (default 2.0)
        wall-t    (default 0.25)
        d1        (* 0.5 ^b-d1)
        d2        (* 0.5 ^b-d2)
        offset    (arrange 0.5 ^d2 ^d2)
        reference-coordinate-system nil
   )
   :subobjects (
        (filter-1 :class 'filter
             b-d1 ^d1
             b-d2 ^d2
             reference-coordinate-system (default)
             orientation (list
                  (translate (list ^^offset ^^offset 0.0)))
        )
        (filter-2 :class 'filter
             b-d1 ^d1
             b-d2 ^d2
             reference-coordinate-system (default)
             orientation (list
                  (translate (list (- ^^offset) ^^offset 0.0)))
        )
        (filter-3 :class 'filter
             b-d1 ^d1
             b-d2 ^d2
             reference-coordinate-system (default)
             orientation (list
                  (translate (list ^^offset (- ^^offset) 0.0)))
        )
        (filter-4 :class 'filter
             b-d1 ^d1
             b-d2 ^d2
             reference-coordinate-system (default)
             orientation (list
                  (translate (list (- ^^offset) (- ^^offset) 0.0)))
        )
   )
)
```

```
;--------------------------------------------------------------------------
;   Class:          cylinder-heater
;   Description:    Defines a cylindrical shape for the water heater tank.
;   Reference 1:    Functional Requirement F 3.1
;                   Form Variation (geometrical shape)
;   Reference 2:    Functional Requirement F 3.2
;                   Form Variation (dimensions)
;--------------------------------------------------------------------------
(define-class cylinder-heater
   :inherit-from (cylinder-object)
   :properties (
        a-h       (default 1.0)
        a-d       (default 1.0)
        height    ^a-h
        diameter  ^a-d
   )
)


;--------------------------------------------------------------------------
;   Class:          box-heater
;   Description:    Defines a cubical shape for the water heater tank.
;                   Its X and Y dimensions are set from one property.
;   Reference 1:    Functional Requirement 3.1
;                   Form Variation (geometrical shape)
;   Reference 2:    Functional Requirement 3.2
;                   Form Variation (dimensions)
;--------------------------------------------------------------------------
(define-class box-heater
   :inherit-from (box-object)
   :properties (
        a-h       (default 1.0)
        a-d       (default 1.0)
        height    ^a-d
        width     ^a-d
        depth     ^a-h
   )
)
```

```
;--------------------------------------------------------------------------
;  Class:         coffee-maker
;  Description:   Defines the entire coffee maker model with properties and
;                 geometry. The main components are: water heater tank,
;                 coffee filter, and coffee pot.
;--------------------------------------------------------------------------
(define-class coffee-maker
   :inherit-from (object)
   :properties (

       ; General
       ;------------------------------------------------------------------
       display-cs? nil
       color 'gray
       (render :class 'option-property-class
            label          "Render Mode"
            formula        'shaded
            mode           'combo
            labels-list    '("Shaded" "Boundary")
            options-list   '(shaded boundary)
       )
       (wall-t :class 'editable-data-property-class
            label "Wall Thickness"
            formula 0.25
       )


       ; Filter Count Selection
       ; Functional Requirement F 2.2
       ; Structure Variation - Number of Elements
       ;------------------------------------------------------------------
       (b-count :class 'option-property-class
            label          "Filter Count"
            formula        'filter
            mode           'combo
            labels-list    '("Single Filter" "Four Filters (2x2)")
            options-list   '(filter filter-2x2)
       )
       b-fs-count (case !b-count
                          (filter         'functional-surface-disc)
                          (filter-2x2     'functional-surface-disc-2x2)
       )
```

```
; Heater Shape Selection
; Functional Requirement F 3.1
; Form Variation (geometrical shape)
;-------------------------------------------------------------------
(a-shape :class 'option-property-class
     label          "Heater Geometrical Shape"
     formula        'cylinder-heater
     mode           'combo
     labels-list    '("Cylinder Heater" "Box Heater")
     options-list   '(cylinder-heater box-heater)
)


; Pot (Element C)
; Functional Requirement F 3.2
; Form Variation (dimensions)
;-------------------------------------------------------------------
(c-d1 :class 'editable-data-property-class
     label "Pot Height"
     formula 7.0
)
(c-h1-diff :class 'editable-data-property-class
     label "Pot Bottom Cut"
     formula 1.25
)
(c-h2 :class 'editable-data-property-class
     label "Pot Rim Height"
     formula 1.25
)
(c-d2 :class 'editable-data-property-class
     label "Pot Rim Diameter"
     formula 4.0
)
c-h1 ^c-d1
c-d1-diff ^c-d1


; Filter (Element B)
; Functional Requirement F 3.2
; Form Variation (dimensions)
;-------------------------------------------------------------------
(b-h :class 'editable-data-property-class
     label "Filter Height"
     formula 5.0
)
(b-d1 :class 'editable-data-property-class
     label "Filter Diameter (bottom)"
     formula 2.0
)
(b-d2 :class 'editable-data-property-class
     label "Filter Diameter (top)"
     formula 7.5
)
```

```
; Heater (Element A)
; Functional Requirement F 3.2
; Form Variation (dimensions)
;----------------------------------------------------------------------
(a-h :class 'editable-data-property-class
     label "Heater Height"
     formula 5.0
)
(a-d :class 'editable-data-property-class
     label "Heater Diameter/Width"
     formula 7.5
)
(a-pipe-h :class 'editable-data-property-class
     label "Heater Pipe Height"
     formula 1.0
)
(a-pipe-d :class 'editable-data-property-class
     label "Heater Pipe Diameter"
     formula 2.0
)
(a-pipe-t :class 'editable-data-property-class
     label "Heater Pipe Thickness"
     formula 0.25
)


; Arrangement Multipliers (Element A)
; Functional Requirement F 2.1
; Structure Variation (arrangement)
;----------------------------------------------------------------------
(x-multiplier :class 'editable-data-property-class
     label "Heater X-Multiplier"
     formula 0.0
)
(y-multiplier :class 'editable-data-property-class
     label "Heater Y-Multiplier"
     formula 0.0
)
(z-multiplier :class 'editable-data-property-class
     label "Heater Z-Multiplier"
     formula 1.5
)


; Coordinates (Element A)
; Functional Requirement F 2.1
; Structure Variation (arrangement)
;----------------------------------------------------------------------
a-x (arrange ^x-multiplier ^a-d ^b-d2)
a-y (arrange ^y-multiplier ^a-d ^b-d2)
a-z (arrange ^z-multiplier ^a-h ^b-h)
```

```
        ; Coordinates (Element B)
        ;-----------------------------------------------------------------
        b-x 0.0
        b-y 0.0
        b-z (* 0.5 (+ ^c-h2 ^b-h))


        ; Coordinates (Element B)
        ;-----------------------------------------------------------------
        c-x 0.0
        c-y 0.0
        c-z1 (* 0.5 ^c-h1)
        c-z2 (* 0.5 (+ ^c-h1)) ; ^c-h2))
        c-h1-diff-z (* 0.5 (- ^c-h1-diff ^c-d1))


        ; Functional Surfaces
        ;-----------------------------------------------------------------
        (display-fs? :class 'editable-data-property-class
             label "Display Functional Surfaces"
             formula t
        )
        (fs-gap :class 'editable-data-property-class
             label "Functional Surface Projection"
             formula 0.6 ; Offset from element
        )
        a-fs-z (- (+ (* 0.5 ^a-h) ^fs-gap))
        b-fs-z    (+ (* 0.5 ^b-h) ^fs-gap)

    )


    :subobjects (

        ; General
        ;-----------------------------------------------------------------

        ; Main Coordinate System
        (main-cs :class 'coordinate-system-class
             origin (list 0.0 0.0 0.0)
             display? ^^display-cs?
        )


        ; Element C (Pot)
        ;-----------------------------------------------------------------

        ; Coordinate System - Pot
        (c-cs1 :class 'coordinate-system-class
             origin (list ^^c-x ^^c-y ^^c-z1)
             reference-coordinate-system ^^main-cs
             display? ^^display-cs?
        )
```

```
; Coordinate System - Pot Rim, Handle
(c-cs2 :class 'coordinate-system-class
     origin (list ^^c-x ^^c-y ^^c-z2)
     reference-coordinate-system ^^c-cs1
     display? ^^display-cs?
)


; Pot
(obj-c1 :class 'difference-object
     object-list (list ^sphere ^cylinder)
     reference-coordinate-system ^^c-cs1
     (sphere :class 'sphere-object
          height    ^^c-h1
          diameter  ^^c-d1
     )
     (cylinder :class 'cylinder-object
          height    ^^c-h1-diff
          diameter  ^^c-d1-diff
          orientation (list
               (translate (list 0.0 0.0 ^^c-h1-diff-z)))
     )
)


; Pot Rim
(obj-c2 :class 'pipe-object
     height         ^^c-h2
     outer-diameter ^^c-d2
     thickness      ^^wall-t
     reference-coordinate-system ^^c-cs2
)


; Pot Handle (part 1)
(obj-c-handle1 :class 'box-object
     ; NOTE: Properties are currently hard-coded
     height 1.0
     width  2.75
     depth  0.5
     reference-coordinate-system ^^c-cs2
     orientation (list (translate (list (+ 2.0 1.25) 0.0 0.0)))
)


; Pot Handle (part 2)
(obj-c-handle2 :class 'box-object
     ; NOTE: Properties are currently hard-coded
     height 1.0
     width  0.5
     depth  4.0
     reference-coordinate-system ^^c-cs2
     orientation (list (translate (list (+ 2.0 2.5) 0.0 -1.75)))
)
```

```
; Element B (Filter)
;-------------------------------------------------------------------

; Coordinate System - Filter
(b-cs :class 'coordinate-system-class
     origin (list ^^b-x ^^b-y ^^b-z)
     reference-coordinate-system ^^c-cs2
     display? ^^display-cs?
)


; Functional Surface B
(b-fs :class !b-fs-count
     height    (- ^^fs-gap)
     diameter  (inner-diameter ^^b-d2 ^^wall-t)
     color     'green
     origin    (list 0.0 0.0 ^^b-fs-z)
     display?  ^^display-fs?
     reference-coordinate-system ^^b-cs
     spacing   (if (equal
          ^^b-fs-count 'functional-surface-disc-2x2) ^^wall-t)
)


; Filter
(obj-b :class !b-count
     reference-coordinate-system ^^b-cs
)


; Element A (Heater)
;-------------------------------------------------------------------

; Coordinate System - Heater
(a-cs :class 'coordinate-system-class
     origin (list ^^a-x ^^a-y (+ ^^a-z ^^a-pipe-h))
     reference-coordinate-system ^^b-cs
     display? ^^display-cs?
)


; Functional Surface A
(a-fs :class 'functional-surface-disc
     reference-coordinate-system ^^a-cs
     height    ^^fs-gap
     diameter  (inner-diameter ^^a-pipe-d ^^a-pipe-t)
     color     'red
     origin    (list 0.0 0.0 (- ^^a-fs-z ^^a-pipe-h))
     display?  ^^display-fs?
)
```

```
        ; Heater
        (obj-a :class 'difference-object
             reference-coordinate-system ^^a-cs
             object-list (list ^outer ^inner)

             (outer :class !a-shape

             )
             (inner :class !a-shape
                  a-d (inner-diameter ^^a-d ^^wall-t)
                  orientation (list
                       (translate (list 0.0 0.0 ^^wall-t)))
             )
        )


        ; Heater Pipe
        (obj-a-pipe :class 'pipe-object
             reference-coordinate-system ^^a-cs
             outer-diameter ^^a-pipe-d
             thickness      ^^a-pipe-t
             height         ^^a-pipe-h
             orientation (list
                  (translate (list 0.0 0.0 (* -0.5 (+ ^^a-h !height)))))
        )

    )
)
```

```
;-----------------------------------------------------------------------------
;   Class:         coffee-maker-gui
;   Description:   Provides a GUI for the user to alter the
;                  coffee-maker through various inputs.
;   References:    Functional Requirements F 2.1, F 2.2, F 3.1, F 3.2
;-----------------------------------------------------------------------------
(define-class coffee-maker-gui
   :inherit-from (coffee-maker data-model-node-mixin)
   :properties (
        label "Coffee Maker Pilot"

        ; Default Properties
        property-objects-list (list
             "General"
             (the superior render self)
             (the superior wall-t self)
             ""
             "Functional Surfaces"
             (the superior display-fs? self)
             (the superior fs-gap self)
             ""
             "Structure Variation - Heater Arrangement"
             (the superior x-multiplier self)
             (the superior y-multiplier self)
             (the superior z-multiplier self)
             ""
             "Structure Variation - Filter Count"
             (the superior b-count self)
             ""
             "Form Variation - Heater Shape"
             (the superior a-shape self)
             ""
             "Form Variation - Heater Dimensions"
             (the superior a-h self)
             (the superior a-d self)
             (the superior a-pipe-h self)
             (the superior a-pipe-d self)
             (the superior a-pipe-t self)
             ""
             "Form Variation - Filter Dimensions"
             (the superior b-h self)
             (the superior b-d1 self)
             (the superior b-d2 self)
             ""
             "Form Variation - Pot Dimensions"
             (the superior c-h2 self)
             (the superior c-d2 self)
             (the superior c-d1 self)
             (the superior c-h1-diff self)
        )
   )
)
```

A.2 Coffee Maker Pilot – Supplementary Files

```
;==========================================================================
;
;   File:          Coffee Maker Pilot Supplements
;   Authors:       Eivind A. Taftø
;   Description:   Contains run commands and AML configuration files.
;
;==========================================================================


;--------------------------------------------------------------------------
;   A simple set of commands for to launch the
;   coffee maker pilot from the XEmacs command line.
;--------------------------------------------------------------------------
(load "C:\\AML\\Pilot\\sources\\coffee-maker.aml") ; Your source file path
(delete-all-models t)
(create-model 'coffee-maker-test :class 'coffee-maker-gui)
(select-model 'coffee-maker-test)
(draw (the) :clear-display? t)


;--------------------------------------------------------------------------
;   AML Lights configuration file.
;   The default lighting used in most screenshots.
;--------------------------------------------------------------------------
((1.0 1.0 1.0) (0.5 0.45 -0.2))
((0.02 0.02 0.02) (-1 0 0))


;--------------------------------------------------------------------------
;   AML View configuration file.
;   The default head-on view used in most screenshots.
;--------------------------------------------------------------------------
Coffee Maker Default View
0 -23.632500 3.733979
0 0.156066 0.987747
0.027129 0 11.187500
26.299973 23.925671


;--------------------------------------------------------------------------
;   AML View configuration file.
;   The default head-on view used when a 2x2 filter option is selected.
;--------------------------------------------------------------------------
Coffee Maker 2x2 View
12.057594 -18.332682 4.872201
-0.119114 0.181104 0.976224
0.027129 0 11.187500
26.299974 23.925671
```

A.3 Coffee Maker Pilot – Discontinued Prototype GUI

```
;==========================================================================
;
;   Module:        Connect Functional Surfaces GUI      (DISCONTINUED)
;   Version:       0.9.1.4
;   Date:          2015.12.18
;   Authors:       Eivind A. Taftø
;   Description:   This is a preliminary GUI prototype for connecting
;                  functional-surfaces. The user would interactively select
;                  two predefined functional surfaces from the model canvas.
;                  Next, the user selects connection type and directoin
;                  from dropdown menus. This information would be saved in
;                  a functional-surface-link class.
;   See also:      functional-surface,
;                  functional-surface-link,
;                  functional-surface-disc,
;                  functional-surface-disc-2x2
;
;==========================================================================

(in-package :aml)


;--------------------------------------------------------------------------
;   Class:         connect-functional-surfaces-gui
;   Description:   Defines the main GUI for connecting functional surfaces.
;   Reference 1:   Functional Requirement F 1.2
;                  Functional Surfaces (connect)
;--------------------------------------------------------------------------
(define-class connect-functional-surfaces-gui
   :inherit-from (ui-form-class)
   :properties(

        ; Common Properties
        title "Connect Functional Surfaces"
        x-margin  3
        y-margin  5
        x-spacing 5
        input-frame-w 40 ; Entire left side frame
        input-label-w 45 ; Left side input labels
        input-frame-w-half (/ ^input-frame-w 2.0)
        input-field-h 8
        input-y-spacing 4
        input-field-total-h (+ ^input-field-h ^input-y-spacing)
        canvas-frame-h (- 100 (* ^y-margin 2.0))
        canvas-frame-w (- 100
             (+ ^input-frame-w ^x-spacing (* ^x-margin 2.0)))
        ; FORM - Main properties
        label ^title
        x-offset  50
        y-offset  50
        width     800
        height    400


        ; FORM - Close
        close-action '(when
             (string= (pop-up-message
                  "Are you sure you want to close the form?"
                  :done-label "Yes"
                  :cancel-label "No")
             "Yes")
             (hide !superior)
        )
   )

   :subobjects (

        ; TITLE
        (lbl-title :class 'ui-label-class
             x-offset      ^^x-margin
             y-offset      ^^y-margin
             width         (- 100 (* 2 ^^x-margin))
             label         ^^title
             label-align   'left
        )


        ; INPUT - Select Surface A
        (txt-obj-A :class 'ui-labeled-field-class
             x-offset      ^^x-margin
             y-offset      ^^y-margin
             width         ^^input-frame-w
             height        ^^input-field-h
             label-width   ^^input-label-w
             label         "Surface A"
        )


        ; INPUT - Select Surface B
        (txt-obj-B :class 'ui-labeled-field-class
             x-offset      ^^x-margin
             y-offset      (+ ^^input-field-total-h
                             (the superior superior txt-obj-A y-offset))
             width         ^^input-frame-w
             height        ^^input-field-h
             label-width   ^^input-label-w
             label         "Surface B"
        )
```

```
; INPUT - Connection Type
(cbo-conn-type :class 'ui-labeled-option-menu-class
     x-offset       ^^x-margin
     y-offset       (+ ^^input-field-total-h
                     (the superior superior txt-obj-B y-offset))
     width          ^^input-frame-w
     height         ^^input-field-h
     label          "Connection Type"
     label-width    ^^input-label-w
     labels-list    '("Touch" "Gravitational" "None")
     options-list   (list 'touch 'gravity 'none)
     selected-option 'touch
)


; INPUT - Connection Direction
(cbo-conn-dir :class 'ui-labeled-option-menu-class
     x-offset       ^^x-margin
     y-offset       (+ ^^input-field-total-h
                     (the superior superior cbo-conn-type y-offset))
     width          ^^input-frame-w
     height         ^^input-field-h
     label          "Connection Direction"
     label-width    ^^input-label-w
     labels-list    '("A to B" "B to A" "Two-way" "None")
     options-list   (list 'ab 'ba 'two 'none)
     selected-option 'ab
)


; BUTTON - Apply
(btn-apply :class 'ui-apply-button-class
     x-offset  ^^x-margin
     y-offset  (+ ^^input-field-total-h
                (the superior superior cbo-conn-dir y-offset))
     width     ^^input-frame-w-half
     height    ^^input-field-h
     label     "Apply"
)


; BUTTON - Cancel
(btn-cancel :class 'ui-cancel-button-class
     x-offset (+ ^^x-margin ^^input-frame-w-half)
     y-offset (+ ^^input-field-total-h
                (the superior superior cbo-conn-dir y-offset))
     width     ^^input-frame-w-half
     height    ^^input-field-h
     label     "Cancel"
)
```

```
        ; CANVAS - Model
        (can-main :class 'ui-canvas-class
             x-offset  (+ ^^input-frame-w ^^x-margin ^^x-spacing)
             y-offset  ^^y-margin
             width     ^^canvas-frame-w
             height    ^^canvas-frame-h
        )
    )
)
```