Abstract

In the conceptual design process a designer has too little resources to encompass all design options, and tools. To integrate new solutions a method should exist that reuses known solutions and is able to efficiently and effectively integrate new design options, and tools. The knowledge based engineering methodology can support an implementation of such a method. It captures human repetitive processes in software primitives. In this paper the development of a multi-level design environment is discussed that embodies such a method. The environment uses feasibilisation heuristics to structure the design process and is based on a set of modular building blocks that enable a designer to define any design process. An implementation of this environment is presented a sample panel design problem based on structural and manufacturing considerations. The approach requires extensive process formalisation, but returns flexible process definition, traceability, automated execution and information capturing.

1 Introduction

To meet future demands new feasible solutions must be available to industry. The attractiveness of a new solution depends on the associated risk and ‘merit of success’. The attractiveness of a solution depends on the balance between risk and merit of success, ideally low risk and high merit of success. Before companies embrace a new solution proper insight in the involved risk and merit of success should be available. Improving the quality of the solution and the efficiency of the process would increase the attractiveness. However, generally this also increases the amount of work. With decreasing number of engineers in the western hemisphere the focus is on deploying computer technologies to perform part of the work. To pass this hurdle an approach is needed that effectively supports engineers with computer technologies.

In the design process the quality of the best solution depends on available availability knowledge. Ideally, a designer has complete knowledge on the design space, implying that the highest quality solution is implicitly known. To approach the ideal situation in reality, all relevant solutions must be known and available before the design process starts. A solution is a depended on the specific combination of requirements, design options, and tools. A method is required that enables capturing and reusing of requirement, design options, and tool related knowledge. Considering the ever-changing requirements, design options, and tools, it is vital that such a method presumes these changes.

1.1 Approach

A design problem aims at finding a physical relation between function, structure, and behaviour. However, regarding new design problems, at the start of the design process a designer has no mathematics available to describe that relation physically. The feasibilisation approach can support a designer in describing that relation by employing problem simplification, problem decomposition, and trial and error methods.

To support engineers in the design process with computer technologies a paradigm shift is required. A human is capable to relate different worlds and to find “outside-the-box” solutions. However, a computer is never bored and can perform the same routine “endlessly”. The design process can be improved by automation of repetitive non-creative processes, decreasing required resources. This relieves engineers from non-value adding activities, making more time available to exploit their creativity and engineering skills. This would increase both the quality of the product and the effectiveness of
A technology that can support this paradigm shift is Knowledge Based Engineering (KBE).

1.2 Paper objective
The research goal is to provide the designer with an overview of the design space giving insight into possible solutions. To illustrate that goal an example implementation of a material design space is presented in Figure 1. This paper focuses on the development of a framework that provides a designer with the functionality to capture and reuse design space knowledge. The framework is based on a set of design process building blocks that can capture any design process. The building blocks are KBE software objects, able to solve the defined problem, relieving the designer from these activities. The framework is tested for a panel design space problem, addressing multiple load cases, stiffener types, stacking sequences, and dimensions based on structural and manufacturing design considerations. In the implementation discussed in this paper the Matlab programming environment is used, mainly for the integral availability of search tools (Matlab optimization toolbox) and a surrogate modelling software tool (Dace).

The paper starts with an elaboration on the methodology that gave rise to this development in section 2. The implication of the methodology on the design process is discussed in section 3. The design environment and its building blocks are elaborated in section 4. The example implementation is described in section 5. The paper is finished with a discussion in section 6.

2 Methodology
The basic approach to analyse the problem is to follow the human engineer. First a brief overview of an approach to a conceptual design problem is presented. Secondly the KBE approach is presented, followed by technologies that can be used in the implementation.

2.1 Feasilisation
The feasilisation methodology encompasses the heuristics of problem simplification, problem decomposition, and trial and error methods. This methodology focuses on how a human solves a problem from the ground up. However, a human also uses previously found solutions to shortcut this elaborate trial and error process and interpolates (and extrapolate) an ‘in-between’ solution. These methods are solution capturing and solution inter- and extrapolation. This paper will focus on the development of a technology that embodies the feasilisation approach. The focus of the implementation will be on the decomposition and trial and error methods. For completeness these are elaborated in the next sections. More details on the methodology can be found in a previous paper by the authors.

2.1.1 Decomposition
A human uses decomposition to break up a complex problem into multiple smaller problems concerning part of the design space.
The decomposition is based on function (embodied by tasks) and design option, such that a diagonal or at least sparsely filled $N^2$ diagram is obtained. The component function is derived from the function and design option of the product, such that the sum of component functions is equal to the product function. Since a component is again a product of its sub-components this process continues down to a level where a physical relation between function, model and behaviour is known or can be chosen. For example, in aircraft structural design these products are called ‘design values’ (e.g. material properties), known feasible solutions. An example for the material product is illustrated in Figure 1. Hence the decomposition process reveals a hierarchical network of products (see Figure 2). The decomposition creates a possibility to capture design knowledge at different abstraction levels, e.g. from material, to panel or aircraft. To obtain the panel design space, all lower level relevant solutions must known.

An advantage of the approach is that new design knowledge (e.g. design options or analysis tools) or known design solutions can be captured by the structure without changing it.

2.1.2 Trial and error methods
Through experience a human gains knowledge and experience is gained by trial and error, a search process.

Three product design categories can be identified; routine, innovative, and creative designs, illustrated in Figure 3. Routine designs concern designs that fit within the space of previous solutions, e.g. redesign of a Boeing 767, innovative designs are based on the same design options, but have extended parameter values, e.g. Airbus A380, and creative designs are based on a different design option, e.g. ‘Blended Wing Body’ instead of a ‘Kansas-city aircraft’. Typically, a new product design will encompass all three design categories, spread across the network of products, e.g. from material to aircraft.

In case of non-routine designs, the designer does not have sufficient knowledge to define a design solution, since a physical relation between function, model, and behaviour is not yet established. By using a trial and error process the designer increases experience via exploring the new design space for feasible relations. If sufficient relations are established the product design problem has become a routine design problem, which can be solved based on the known relations between function, model, and behaviour.

3 Design process
To tackle a new problem defined by a set of requirements, design options, and tools a human uses the design process. The design process aims at finding a set of ‘optimal’ product specifications (model and behaviour properties) to a certain set of requirements (functions, performances, and constraints), see Figure 4.

![Figure 4: Traditional design process; requirement definition, product model definition, product model testing, and result evaluation.](image-url)
Basically, it tries to find the physical relations between function, model, and behaviour. The term model means a simplified description of a product. The difference between model and behaviour is that the term model refers to the intrinsic properties of the system, and the term behaviour refers to the extrinsic properties of the product. For example length is an intrinsic property, independent of the product environment, and drag is an extrinsic property, dependent on the environment.

3.1 KBE-enabled design process

In order to find a feasible relation between function, model, and behaviour, the designer iteratively changes the product topology and dimensions. Topology is used here to differentiate between discrete changes in product layout or configuration, e.g. a table can have 4 or 3 legs, not 3.5. Although the product is changed, the process itself does not change significantly, the designer has to perform the same steps repeatedly. This is a typical process that can be improved by the use of KBE technologies. In the KBE-enabled process the selection of topology and dimensions is outsourced to a software tool, see Figure 5.

The design process is started by defining per requirements set one or more topologies (design option sets), assumed to be able to meet the requirements. Generally, these topologies have disconnected design spaces, making it a search problem of a discontinuous design space. If the topology parameters are properly defined as continuous variables, the search process can be simplified to a set of continuous search problems, a sizing problem (a design problem with fixed topology). In this implementation the topologies are assumed to be properly defined and are investigated separately. After each topology is sized individually, the best performing topology is obtained by a trade-off. The trade-off is simplified to a selection based on objective function value, since this is identical for every topology.

3.2 Search process

In search for the best feasible product design a product is modelled as a system, a ‘dynamic’ product able to adapt. In concurrence the term system is used from this point on instead of product.

The search process involves four steps; problem definition (how are the requirements translated to a search problem), topology definition (what are possible model topologies, parameters), variable definition (what are possible parameter values), and objective and constraints evaluation. The objective and constraints are related to certain performances, which follow from the performance process, see section 3.3. Next to that constraints can be related to model parameter values.

3.2.1 Problem formulation

The selection of the search method depends on the type of problem structure. A generic layout of the problem structure is visualised in Figure 6. The decomposition process (introduced in section 2) can be implemented by a multi-level formulation. Here the functional view of the MML is used, relating function, principle of solution (here system primitive), technical solution (here topology), and concept structure (here problem instantiation). In this fashion the design problem addresses a kind of design
problem tree. Three different types of problems are identified:

- **Mono-level**
- **Multi-level, fixed sub-system topology**
- **Multi-level, variable sub-system topology**

In this paper the multi-level optimisation of a fixed topology is implemented, see ⁶ for an extended discussion on the other problems. The multi-level optimisation addresses a fixed-topology problem, a sizing problem. The optimisation problem is defined as:

\[
\min f(x) = \sum a_i \cdot f_i(x)
\]

subject to:

\[
\begin{align*}
g(x) & \leq 0 \\
h(x) & = 0 \\
lb & \leq x \leq ub
\end{align*}
\]  

The problem definition is obtained by integration of the system optimisation problem with the sub-system optimisation problems. Because the design vector of the sub-systems is known, the system level design vector can be simply extended to incorporate the sub-systems level design vectors. To obtain the objective and constraint functions the sub-systems are called twice; once for model properties based on the design vector, and once for performance properties based on the test case.

To get an initial set of parameter value sets that comply with the shape of the solution space, Design of Experiments (DoE) is used, based on Latin Hypercube Sampling. The experiments are analyzed separately. In all three cases a single level optimiser can be used to solve the optimisation problem, e.g. Sequential Quadratic Programming (SQP).

### 3.2.2 Integration of topologies

A difficulty occurs in addressing the multitude of topologies. The nature of the fixed topology approach necessitates that all topologies must be investigated separately. Thus per design problem all relevant topologies must be reinvestigated. For each problem the designer must assemble and integrate the systems required to solve the specific problem.

3.3 Performance process

The performance process is responsible for finding the performance values required to define the objective and constraint functions. The performance process features three steps; performance definition (what performances are required to define the objective and constraints), test process definition (which tests are required to obtain certain performances), and a model process definition (which model properties are required for certain performances, and which system descriptions are necessary to perform certain tests).

The performance process is often referred to as analysis. Analysis is a decomposition of the system properties, generating a number of sub-systems (discipline specific, e.g. structural model or aerodynamic model) contemplating a sub-set of properties. Here the subdivision is made between model and test processes to differentiate between model and behaviour properties relating to the design. The relations between the model, test, performance, objective, and constraint properties can be illustrated with a set of simple formulae, see equation 2.

![Figure 6: General decomposed design problem system layout linked to the functional view of the MML](image-url)
3.3.1 Test process

The model process defines all required behaviour (extrinsic) properties. The required performances are obtained by applying criteria to behaviour. Thus the required behaviour properties follow from the required performances. Every type of behaviour is associated with one (or multiple) test cases, relating to the system tasks (and functions). The system behaviour is obtained by determining the impact of the environment on the system in a certain condition, defined by a test case, through model-based behaviour calculation. The remaining required properties to construct the system and environment models are delivered by the model process. These properties depend on the view of specific discipline on the system.

3.3.2 Model process

The model process defines all required model (intrinsic) properties. As seems logical, before the analysis is started first a generic system model should be defined, to ensure (e.g. geometry) model consistency in the test phase. Based on this model the discipline-specific model views can be derived to ensure a consistent test process. This view is often limited to geometrical properties of the system, but surely not always (e.g. cost modelling).

4 Multi-level environment

The described process is implemented in a knowledge-based software tool. The repetitive character of the design process is used to identify multiple system primitives, used as building blocks to reconstruct the complete design problem. These primitives belong to the set of engineering primitives and focus on capturing knowledge related to search processes. Other engineering primitives are constructed for knowledge related to transformation processes and properties.

The primitives represent the elementary building blocks of a design problem. Following the feasilization methodology, multiple product levels are identified. All these levels can be captured by a primitive, which encapsulates design knowledge associated with that specific level. The resulting multi-level environment structure is abstractly illustrated in Figure 7. See Figure 8 for a more explicit illustration.

4.1 Primitive concept

The basic idea behind the primitive development is the reuse of system-related knowledge. The term reuse implies that after storage something can be used again. If knowledge is to be reused it must be both stored (static) and used (dynamic). Consistently, for every type of knowledge, both a generic static data format, and a set of dynamic software objects are defined. The generic data format is

\[
\begin{align*}
\text{model} & = f_1(\text{variables}) \\
\text{criteria} & = f_2(\text{model}) \\
\text{behaviour} & = f_3(\text{model}) \\
\text{performance} & = f_4(\text{criteria, behaviour}) \\
\text{objective} & = f_5(\text{performance}) \\
\text{constraint} & = f_6(\text{performance, variables})
\end{align*}
\]
used to instantiate the software objects, which in turn are able to translate its structure back into the same generic data format.

In this specific implementation XML is used as generic data format and specially developed Matlab classes are used to generate dynamic objects.

The types of knowledge primitives depend on the decomposition. In this work knowledge is divided into information, skill, and understanding, relating to property, process, and system primitives. These are discussed hereafter. For more information on the decomposition the reader is referred to [7].

4.2 System primitives
The system primitives capture knowledge related to search processes, e.g. find the optimal solution defined by an objective function and a set of constraints. A system state is defined by objective, constraint, model, and test properties. The system is able to ‘modify’ itself by using inference, defined by model, test, and search processes.

Two system primitives are developed to test the concept. The first integrates a multi-level environment, e.g. system, sub-systems, etc. into a single optimisation problem. The system state is obtained by solving the optimisation problem based on the system level objective as objective statement, and the individual system variables and constraints are collected to obtain respectively the design variable vector and the constraint vector.

The second integrates a range of design problems, by taking a predefined design space. The system state is obtained by using a surrogate model to predict the system solutions. The surrogate model is defined based on a set of discrete design problem solutions obtained by instantiations of the first implementation. An example problem using these systems is presented in section 5.

4.3 Process primitives
Process primitives capture knowledge related to model or test processes, e.g. if length and width are 10, the area is 100, or if constraint smaller then 0, then satisfied. A process state is defined by input and output properties.

Two basic types of process primitives are developed to test the concept. The first connects input and output properties by calling an underlying function, executed by string evaluation. First the input values are set, secondly the function call, captured in a string, is evaluated. This sets the output values which are collected and stored. As test functions a Matlab function and a remote function call are developed, addressing respectively a standard Matlab function, and a remote service which returns the requested values based on a HTTP call.

The second connects the input and output properties by execution of multiple process primitives via backward chaining. On instantiation the process flow that connects the output to the input properties is defined, setting the overall required input properties. If the input properties are changed the state is updated by updating the underlying process primitives.

4.4 Property primitives
Property primitives capture knowledge related to data or information, e.g. length is equal to 10, and has a maximum value of 15 or the topology of a table top is ‘rectangular’. A property state is defined by its value.

Implementations are performed for multiple data types available in the Matlab environment to test the concept. The data types; number (float or integer), text (char), array, structure, and cellArray are captured in specially developed classes. Depending on the type of property, the state definition is extended to encompass bounds, variability (e.g. is it mutable, or does it want to minimise itself), objective value (what value does it contribute to the system objective value), and feasibility (does it meet all internal constraints) among others. The latter is also used in the extended state definitions of the system and process primitives.

5 Example design environment
The primitive definition allows the designer to define the problem structure, although consistency and completeness of the definition are required. The technology is tested for a
panel design problem. The panel is designed for minimum weight and cost, subject to an in-plane load case, and constraint by structural and manufacturing design considerations.

Hereafter the design problem is discussed shortly, for a more elaborate discussion on the decomposition of a comparable panel design problem see a previous paper $^9$ by the authors. That paper addresses a panel design problem based solely on structural design considerations. To encompass for manufacturing knowledge the structure of this example had to be adapted. This strengthens the suggestion that the decomposition depends heavily on the designer and the specific design problem.

5.1 Design problem
The generic panel design problem has a varying set of ‘environment conditions’ which consist of a certain load case, length, and width. For a problem instantiation these are fixed as is the topology.

The topology is defined at multiple levels, see Figure 8. In short, a panel can have a stiffened or unstiffened topology, a plate can be a skin or have a stiffener topology (e.g. hat, L, blade, among others), a lay-up can have multiple stacking sequences, a layer can have multiple orientations and thicknesses, and a material can have multiple properties. In this paper the topology is fixed, only the stiffener type is varied between a hat- and L-stiffener topology. The resulting two problems are used as the guiding example.

The skin plate lay-up is $[45,0,45]$, and the stiffener plate lay-up is $[45,0,0,45]$. The panel is based on one material, a carbon fibre composite quasi-orthotropic material. The design variables are stiffener spacing, and dimensions.

5.2 Structural analysis
The panel is structurally designed to sustain the load case and is constrained by strength and stability. The material strength of every layer is evaluated with Tsai-Hill, the skin and stiffener plate stabilities are evaluated by initial buckling determined with conceptual analytical formulas $^3, 9$. The panel overall stability is evaluated by initial buckling determined with similar analytical formulas.

5.3 Manufacturing analysis
The skin and stiffener plates are manufactured by hand lay-up. The panel is assembled by bonding the plates.

The cost estimation is based on a time estimation$^{10}$ performed for all steps taken in the production process. The steps are analysed based on cost estimation relations, which depend on the design and in part on the machines and people performing the process. The machine and people data is provided by a manufacturing database, which stores data about the company or factory that will produce the part. The assembly joint cost analysis is based on the shape of the joint and the parts that are involved in the assembly joint. The positioning times of the parts into the mould are estimated based on surface area of the parts involved.

5.4 Design space
The example design space illustrated in Figure 1 addresses two parameters, the relative cost per volume and Young's modulus, and for multiple concepts, e.g. polymers and alloys. Currently, the panel design space includes two concepts, the hat-stiffened and the L-stiffened panel concept, and addresses 5 parameters, varying in
cost versus weight objective, normal load, length, cost, and weight. The first three dimensions are input parameters and the last two are output parameters. The two concepts are addressed for every combination of parameters. Using full factorial design the design space is spanned. Per parameter four design locations are used, adding up to a total of 64 discrete design problems per panel concept. The results are presented in Figure 9. Every design problem is printed in every graph by point. To illustrate the dependencies between the separate graphs an example design problem point is circled in every graph. On the whole, the resulting figure gives a general overview of the addressed design problem space of the two topologies.

The problems are sized based on one start vector, assuming that the optimisation algorithm will find the optimum solution. The figure demonstrates that this is almost always the case, except for a few L-stiffened panel design cases which appear to deviate too strongly with the expected trend.

Interpreting the design results of cost versus weight, the hat-panel concept is cheaper in case of heavy designs. This is expected since the extra costs involved in bonding of the stiffeners to the skin, since the L-stiffened panel requires more stiffeners to obtain the same performance. On the other hand in case of relative cheap designs the L-stiffened concept yields lighter designs.

Figure 9: Full factorial design results for a hat-stiffened and L-stiffened panel concept
6 Discussion

The implementation showed that it is possible to generically describe a fixed-topology design process using the primitives as building blocks.

The performance of the complete process depends highly on the development environment. The performance of the current implementation is low, since the Matlab environment is not designed to support such an implementation.

The flexibility in definition showed when the first implementation integrating only structural design considerations was extended to take into account also manufacturing considerations. After theoretically formalising the new environment structure the practical implementation was reduced to merely dragging and dropping functions in their new location. An important lesson learned is the dependency of problem structure on both the problem and the designer.

Although the lack of performance initial results are obtained. These results show the benefit of using the approach since with a single definition the different topologies are all designed automatically.

6.1 Future work

The next step is to use this technology in a generic design problem, incorporating more topologies. Thereafter development of a user-interface is to be addressed to enable the designer to perform topology trade-offs in the design process.