DEVELOPMENT AND IMPLEMENTATION OF KNOWLEDGE-BASED DESIGN PROCESS PRIMITIVES

E.J. Schut and M.J.L. van Tooren
Delft University of Technology

Keywords: *design process, knowledge based engineering, engineering primitives*

Abstract

In the conceptual design process a designer has too little resources to encompass all design options, and tools. A designer loses too much time in adapting old solutions to encompass for new requirements. 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 design process primitives is discussed that embodies such a method. An implementation of generic sizing problem primitives is presented and tested for a conceptual panel structural design problem. The primitive approach showed a high decrease of repetitive work associated with the conceptual design phase.

Nomenclature

b_i	:	system behaviour properties
c_i	:	system constraint properties
m_i	:	system model properties
t_i	:	system test case properties
v	:	Lagrange multiplier estimates
x_i	:	system variable properties
M	:	penalty weights

1 Introduction

New solutions in engineering design are required to meet future demands. In 2002 the Advisory Council for Aeronautics Research in Europe[1] stated that in twenty years the aeronautic systems will differ from today's systems at least as much as today's systems differ from those of the 1930s. The aeronautics community will have to take on such a challenge and be one of the pioneers of the European Union *knowledge society*[1,2].

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'. Risk is defined as probability of failure times impact, which can be simplified to invested resources (at least today). Merit of success is the product of probability of success and impact, the gained resources (returns). 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 knowledge on the product design space will increase this insight. However, generally this also increases the required resources. To pass this hurdle an approach is needed that enables companies to gain product design space knowledge with a limited amount of resources.

1.1 Approach

To increase design space knowledge more design problems must be attacked and the investigation must be more thorough, both increasing invested resources. In engineering design the driving resources are people. 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". Supporting engineering by automating repetitive non-creative processes the design process can be

improved, decreasing required resources. This relieves engineers from non-value adding activities, making more time available to exploit their creativity and engineering skills. In order to pass to this new vision of business, knowledge should be managed and engineered as a key business asset[2].

This means that the engineering design process will need to make a paradigm shift. A technology that can support this shift is Knowledge Based Engineering[3] (KBE).

1.2 Paper objective

This paper focuses on the identification, capturing, and reusing of design process knowledge such that the process is separated from product knowledge. The approach will identify the interfaces between the different disciplines (knowledge domains) associated with the design process, and provide a generic implementation of these interfaces.

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

A design problem aims at finding a physical relation between function, model, and behaviour (see section 3). However, at the start of a design problem a designer has no mathematics available to describe that relation physically. The feasilisation approach supports a designer in this phase.

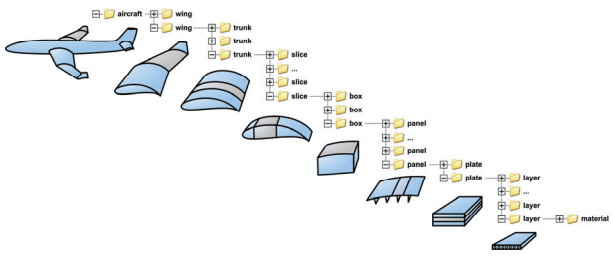

Figure 1: Products of a decomposition of an aircraft

The feasilisation[4] methodology encompasses the approaches 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 interpolate (and extrapolate) an 'in-between' or 'based-upon' solution (e.g. educated guess). 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[4].

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 until 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. Hence the decomposition process reveals a hierarchical network of products (see Figure 1). The decomposition creates a possibility to capture design knowledge at different abstraction levels, e.g. from material, to aircraft.

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[5], illustrated in Figure 2. 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.

2.2 Knowledge based engineering

La Rocca[6] defines KBE as a technology that is based on the use of dedicated software tools (i.e. KBE systems) that are able to capture and reuse product and process engineering knowledge. The main objective of KBE is reducing time and cost of product development by means of the following:

- *Automation of repetitive and non-creative design tasks*
- *Support of multidisciplinary integration from the conceptual phase of the design process*

The KBE cornerstones are rule-based design, object-oriented modelling, and parametric CAD[3]. KBE has its roots in knowledge-based systems (KBS) applied in the field of engineering, hence the name. KBS is

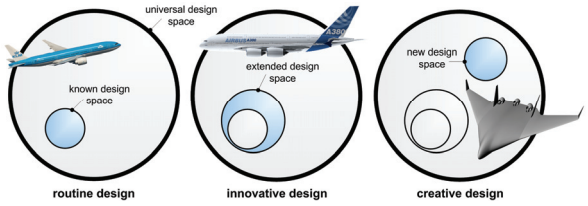

Figure 2, Product design categories; routine, innovative, and creative designs

based on methods and techniques from artificial intelligence (AI). AI aims at creating intelligent entities[7]. KBE focuses on capturing rules of repetitive, non-creative human processes. KBE found its first application as follow-up of CAD to enable designers to reuse models.

A KBE based design environment is in development to support a Multi-disciplinary Design and Optimization MDO design problems, called the Design and Engineering Engine[3,8] (DEE).

2.3 Supporting technologies

The main tools available for application of KBE are object oriented modelling and programming

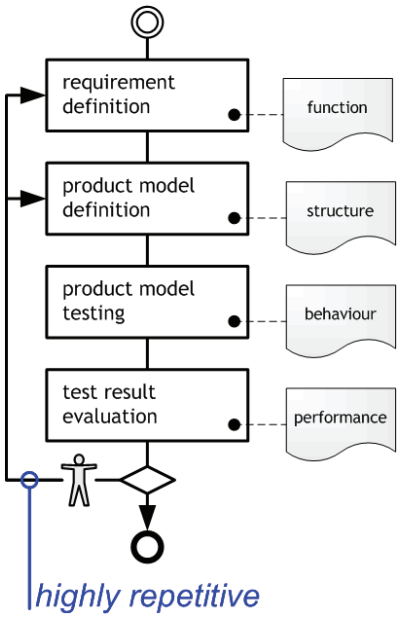

Figure 3: Traditional design process; requirement definition, product model definition, product model testing, and result evaluation.

languages. Engineers have a product or object-oriented view of the world, which the object-oriented approach supports. Useful modelling languages are UML, SysML[9] and more specific MML[10], developed specifically for KBE purposes.

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[11]).

3 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 3. Basically, it tries to find the physical relations between function, model, and behaviour[12]. 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.

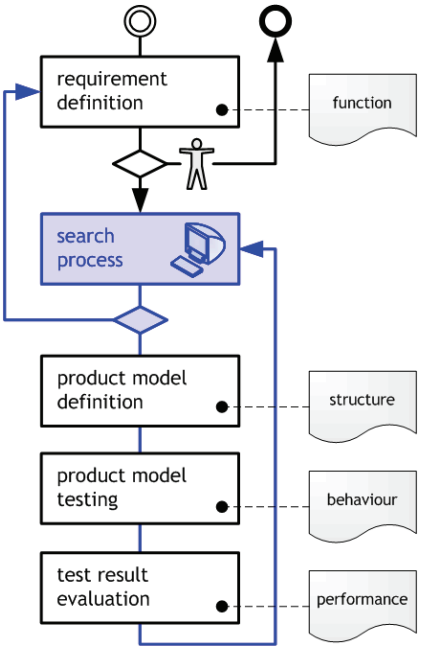

Figure 4: KBE-enabled design process. The search process takes a central position, between the requirements definition (problem) and the product definition (solution).

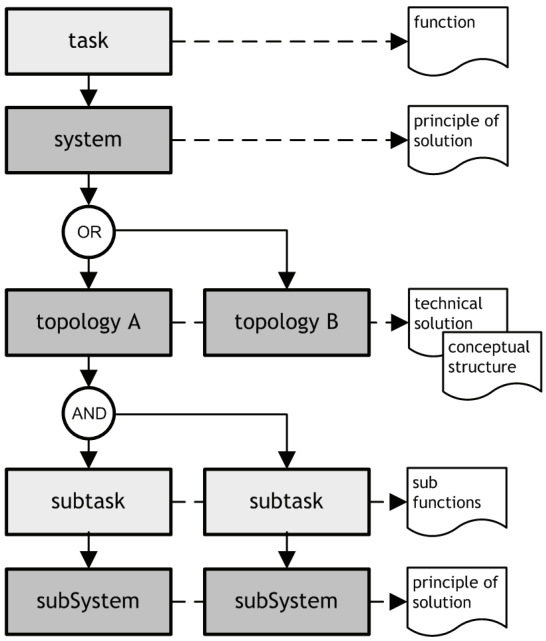

Figure 5: General decomposed design problem system layout

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 4.

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

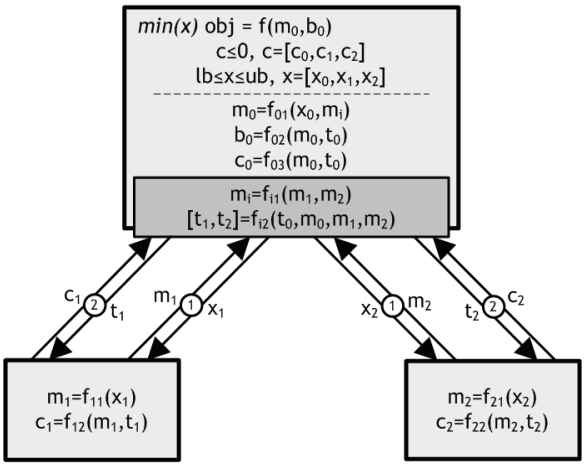

Figure 6: Multi-level search problem definition for a fixed topology design. The search algorithm is able to change input at all levels.

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.

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. The second step of deliberating between topologies is out-of-scope for this paper and performed separate from the search process.

3.2.1 Problem formulations

The selection of the search method depends on the type of problem structure. A generic layout of the problem structure is visualised in Figure 5. 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*

The mono-level problem is the problem that is to be solved at the problem tree leafs. This problem can be addressed by standard constrained optimisation algorithms. These problems are generically defined as:

$$\begin{aligned} &\min_x f(x) = \sum a_i \cdot f_i(x) \\ &subject\ \ to: \\ &\quad g(x) \le 0 \\ &\quad h(x) = 0 \\ &\quad lb \le x \le ub \end{aligned} \tag{1}$$

In case the multi-level optimisation addresses a fixed-topology problem, also referred to as a sizing problem, the system optimisation can be integrated with the sub-system optimisations. 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 design vector, and once for performance properties based on the test case, see Figure 6.

In case the sub-systems can assume multiple topologies, the system is not able to control the design vector (input) of the sub-systems anymore. The only control left is output

$$
\begin{aligned}
model &= f_1(variables) \\
criteria &= f_2(model) \\
behaviour &= f_3(model) \\
performance &= f_4(criteria, behaviour) \\
objective &= f_5(performance) \\
constraint &= f_6(performance, variables)
\end{aligned}
\tag{2}
$$

control of the sub-systems. Multiple possibilities exist to formulate such a problem. In this case a constraint relaxation[13] formulation is preferred, because it suits both optimisation problem and design methodology. The formulation is illustrated in Figure 7. Another possibility would be to use e.g. Bi-Level Integrated System Synthesis[14] (BLISS) for the formulation. An implementation of this approach is out of scope for this paper.

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).

3.3 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.

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.

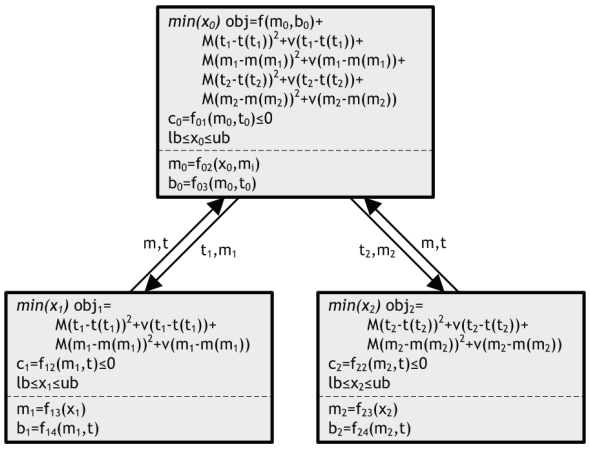

Figure 7: Multi-level search problem definition for a variable topology design. The search algorithm is not able to change input.

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[3] 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 geometry properties of the system, but surely not always (e.g. cost modelling).

4 Engineering primitives

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[15]. However an elaboration on these primitives is out of scope for this paper.

This paper focuses on the development of system primitives capturing the first two problem formulations, mono-level, and multi-level with fixed topology.

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 used to instantiate the software objects, which in turn are able to translate its structure back into the same generic data format.

Which types of knowledge are used depends on the analysis. In this work knowledge is divided into information, skill, and understanding, relating to properties, processes, and systems. The last one is elaborated in this paper. For more information see a previous paper[15] by the authors.

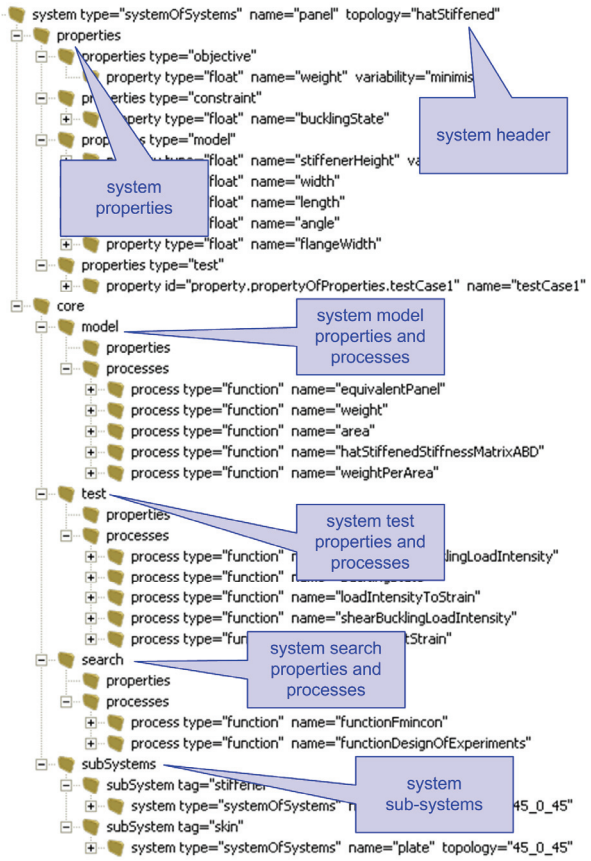

Figure 8: Example the system primitive, specifically for the multi-level fixed topology

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

4.2 System primitives

The system primitives capture knowledge related to search processes. Each primitive represents a different type of search process. The system primitive data format of the multi-level system with fixed-topology is presented in Figure 8.

A system is composed of two main bodies; properties, and core. The properties are generically defined and identical (in format) for all systems. Generically a search process is defined by an objective, constraints, and

variable/fixed properties. To support the nature of the design process the variable/fixed properties are split into model and test properties.

The core holds search specific information and is tailored to encompass the search process. Since both examples address the design process, the core features model, test, and search processes. The multi-system also encompasses the sub-systems and is equipped with model and test processes to enable the multi-level formulation.

The system processes are executed based on the formulation defined in section 3. These processes are defined by the user and depend on the problem to be solved.

5 Implementation

These system primitives are used as building blocks for specific design problems. Following the feasilization methodology, multiple product levels can be identified. All these levels can be captured by a single system primitive, encapsulating design knowledge associated with that specific level. The actual decomposition depends on the user.

In this paper the primitives are used to capture the multiple levels associated with panel design; panel, plate, layer, and material. See section 6 for more details.

5.1 Application

The technology is tested for a panel structural design problem. The panel design problem example is described in detail in a previous paper[16] by the authors. For completeness a short elaboration is presented here.

The panel design problem is designed for an in-plane load case for minimum weight and constrained by strength and stability. The topology is defined at each level. Each panel can have a certain fixed length and width, and can have multiple topologies; for example a panel can be hat-stiffened, I-stiffened, a plate can have multiple stacking sequences, and a layer can be a composite of multiple materials.

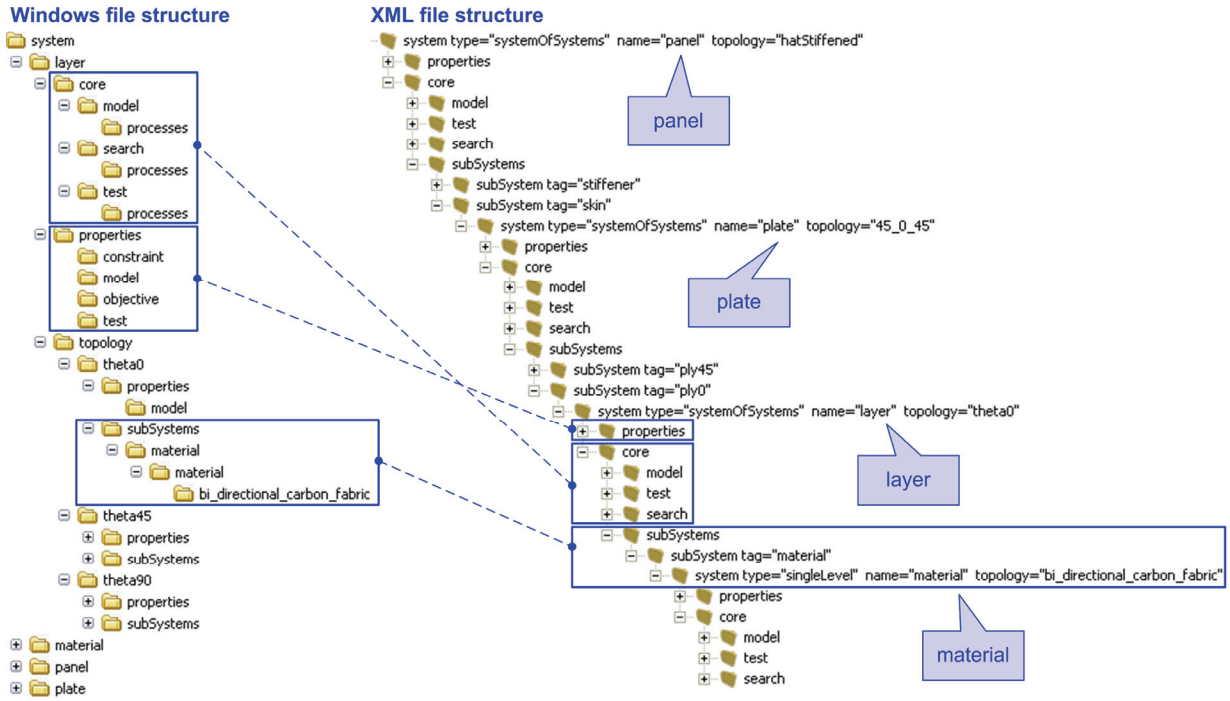

Figure 9: On the left, the windows file structure used to assemble all relevant design information. On the right, an example xml file (automatically defined based on the file structure) for the application of the system primitive definitions to model a multi-level panel sizing problem

As an initial test case a single topology is used, a hat-stiffened panel with a global plate stacking of [45,0,45] of a carbon fibre composite material.

5.2 Topology user interface

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.

This process is highly iterative and not creative. In the current application a file structure lay-out is used that is comparable with the xml structure, see Figure 9. In this file structure the user has to assemble all relevant systems. Per system the generic operations relating to modelling, testing, and search are added. Specific information on the (to be assessed) topologies are all included, thus multiple topologies can be defined for a system. A program is developed based on the primitives described in section 4 that automatically encapsulates all the information, e.g. a Matlab function is translated to a XML structure by using in-file documentation statements.

These objects are the basis for the generation of all required fixed topology design problems. Recursively all the system topologies are combined to a fixed topology problem, and added to a list. The end result is a list of XML structures, each representing a fixed topology design problem.

5.3 Engineering primitives

One of the fixed topology design problem is used to assess the performance of the automated design process.

As mentioned in section 2.3 the implementation is performed in the Matlab programming environment, using specially developed Matlab classes. The mono-level and multi-level sizing problems are implemented in two different classes, but inherit generic functionalities from a system super class.

The XML file is translated into the Matlab environment using the Docoment Object Model (DOM) structure. This structure is the input for the primitive classes, which are instantiated. The instantiation process first the information is assembled then the process is validated. If a design problem is stated the objects update to match the problem, which means that a search is performed based on the stated variable (model), test, objective, and constraint properties.

On user demand the object is able to translate itself to a DOM structure, which can be translated to a XML file using standard functions. The amount of information captured during this process can be illustrated by comparing the input and output XML file size. The input size of a single panel problem is 100 KB, after sizing the output XML has increased in size by a factor of eight to about 800 KB.

6 Discussion

The implementation showed that it is possible to generically describe a fixed-topology design process. The search process primitives are implemented for a structural design test problem using the Matlab programming environment. Although the environment is selected based on available technologies concessions are made regarding the capabilities of the programming language. The performance of the application would be better in a more suitable object-oriented software environment, such as Java or Python.

The performance of the complete process depends highly on the used search routines, functions, and surrogate modelling tools. The performance of the current implementation is low, since these technologies require specialist attention. However this also addresses the main benefit of this approach since specialists are enabled to work on their specific problem, without (in principle) addressing other specialist fields. Secondly the performance depends on the complexity of the design problem. In general, for a single run, a generic implementation will always require more resources then specific implementations. The generic implementation trade-off will outperform when repetitiveness becomes an issue.

The other main benefit of the approach is the decomposition of product and process. A

designer only has to compose the design problem based on building blocks created by specialists, highly decreasing the effort associated with assembly and integration of a design problem. This modular approach will support the capturing and reuse of design knowledge.

6.1 Future work

The next step is to use this technology in a more extensive design problem, incorporating also aerodynamics discipline. Thereafter the implementation of the variable topology system primitives will be addressed, to enable the designer to really integrate topology trade-offs in the design process. Instead of performing fixed-topology designs separately, the topology trade-off is integrated in the process itself.

References

[1] Advisory Council for Aeronautics Research in Europe (ACARE), "Strategic research agenda", Vol. 1+2 and executive summary, 2002, URL: http://www.acare4europe.com

[2] Drucker, P., "Management challenges for the 21st century", Harper Business, 2001

[3] La Rocca, G., and van Tooren, M.J.L., "Enabling distributed multi-disciplinary design of complex products: a knowledge based engineering approach", J. Design Research, Vol. 5, No. 3, pp.333-352

[4] Schut, E.J., and M.J.L. van Tooren, "Design 'feasilization' using Knowledge Based Engineering and Optimization techniques", Journal of Aircraft, Vol. 44, No 6, 2007, pp 1776-1786

[5] Gero, J.S., Maher, M.L., "Modelling creativity and Knowledge-Based Creative design", Lawrence Erlbaum Associates, 1993

[6] La Rocca, G., PhD. thesis, Delft University of Technology, Delft, Netherlands (to be published)

[7] S. Russel, P. Norvig, "Artificial intelligence: a modern approach", second edition, Prentice Hall, 2003

[8] La Rocca, G., and van Tooren, M.J.L., "Development of Design and Engineering Engines to Support Multidisciplinary Design and Analysis of Aircraft," *Delft Science in Design - A congress on Interdisciplinary Design, Faculty of Architecture*, ISBN 90-5269-327-7, Delft, NL, 2005

[9] OMG Systems Modeling Language (OMG SysML™), URL: http://www.sysml.org

[10] MML Working Group, "MOKA User Guide", URL: http://www.kbe.coventry.ac.uk/moka/Documents/consortium/mig_def.pdf

[11] Nielsen, H.B., S.N.Lophaven, and J.Søndergaard 'DACE: Design and Analysis of Computer Experiments'. URL: http://www2.imm.dtu.dk/~hbn/dace/

[12] Gero, J.S., Maher, M.L., "Modelling creativity and Knowledge-Based Creative design", Lawrence Erlbaum Associates, 1993

[13] S. Tosserams, L. Etman and J. Rooda, "Performance Evaluation of Augmented Lagrangian Coordination for Distributed Multidisciplinary Design Optimization", 49th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, AIAA-2008-1805, Schaumburg, IL, USA, 2008

[14] Sobieszczanski-Sobieski, J., J.S. Agte, and R. Sandusky Jr., "Bi-Level Integrated System Synthesis (BLISS)", NASA, TM-1998-208715, 1998

[15] Schut E.J., M.J.L. van Tooren, "Engineering Primitives to Reuse Design Process Knowledge", 49th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, 4th AIAA Multidisciplinary Design Optimization Specialist Conference, AIAA-2008-1804, Schaumburg, IL, USA, 2008

[16] Schut, E.J., M.J.L. van Tooren and J.P.T.J. Berends, "Feasilization of a Structural Wing Design Problem", 49th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, AIAA-2008-2263, Schaumburg, IL, USA, 2008

Copyright Statement