49th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference
 16t
7 - 10 April 2008, Schaumburg, IL

AIAA 2008-1804

Engineering Primitives to reuse design process knowledge

E.J. Schut[1], and M.J.L. van Tooren.[2]
Delft University of Technology, Kluyverweg 1, Delft, the Netherlands

In the conceptual design process a designer has too little resources to encompass all many requirements, design options, and tools to find an initial set of feasible solutions. To find new solutions knowledge should be managed and engineered as key business asset. The knowledge based engineering technology supports the implementation of this approach. It captures human repetitive processes in knowledge primitives. In this paper the development of design process primitives is discussed, based human approaches to solve a complex design problem and by analyzing the design process. The development yielded four engineering process primitives; environment, system, process, and property primitives. These primitives match framework scoping levels and the knowledge categories; wisdom, understanding, skill, and information.

I. 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, improved knowledge is obtained through experience, also increasing the invested resources. To pass this hurdle an approach is needed that enables companies to gain product design space knowledge with a limited amount of resources.

To increase design space knowledge more design problems must be attacked and the investigation 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 an "out-of-the-box" solution. 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 them 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. Where previously the geometric model took a central position, today design knowledge should have the focus. This implies that the design process needs knowledge primitives instead of geometric model primitives; a technology that supports this shift is Knowledge Based Engineering[3] (KBE). Currently, many design processes rely on CAD primitives (e.g. lines, surfaces, volumes) to capture the product model, sometimes extended to capture design knowledge, however still presuming that the CAD primitives are the proper primitives to capture also non-geometrical design knowledge. We argue that a proper implementation should be based on the right primitives: engineering knowledge.

[1] PhD. student, Design of Aircraft and Rotorcraft, e.j.schut@tudelft.nl, AIAA member
[2] Professor, Design of Aircraft and Rotorcraft, m.j.l.vanTooren@tudelft.nl, AIAA MDO TC member

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[3,4] (DEE). Within this framework a generic modeler is developed, called the Multi Model Generator (MMG), based on KBE primitives. The primitives are called High Level Primitives[3] (HLPs), capturing not only geometry but all product modeling knowledge in a product primitive. To start the MDO process an initial set of parameter values are required, delivered by the so-called Initiator. In a previous article[5] by the authors an example structure Initiator was developed based on problem feasilization, a set of human methods to attack an engineering design problem; problem simplification, problem decomposition, and trial and error methods. The article showed that the Initiator could be modeled as multiple DEEs, different in scope (e.g. aircraft, panel) and phase (e.g. conceptual, preliminary).

To capture the Initiator process, the DEE design process must be captured. Following the KBE methodology a set of design process primitives should be obtained. This paper presents an approach to and implementation of such engineering (design process) primitives. The methodology is based on the branch of evolutionary epistemology[6]. Specific technologies used for the implementation are for searching MDO, for modeling KBE, for capturing extensible markup language[7] (XML), and for communication an agent based framework[8] (ABF).

This paper is structured as follows. First a short background on the KBE methodology and DEE concept is presented in section II. In section III the focus is on the human approach in the design process, discussing applied methods to solve an engineering design problem. In section IV the engineering design process is elaborated and related to the previously mentioned technologies. Section V introduces the engineering primitives, discussing the theory and implementation. The paper is concluded with a discussion on the performed research and the next development steps.

II. Knowledge Based Engineering

A. Knowledge based engineering

La Rocca[9] 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 modeling, and parametric CAD[3]. KBE has its roots in knowledge-based systems (KBS) applied in the field of engineering, hence the name. KBS is based on methods and techniques from artificial intelligence (AI). AI aims at creating intelligent entities[10]. KBE focuses on capturing rules of repetitive, non-creative human processes. Engineers have a product or object-oriented view of the world, which the object-oriented modeling approach supports. KBE found its first application as follow-up of CAD to enable designers to reuse models. CAD is based on geometrical primitives, KBE on knowledge primitives.

Cooper[11] defines five important features required for any KBE system: functional coding style, declarative coding style, runtime value caching and dependency tracking, dynamic data types, and automatic memory management. However to be complete the system should support code-generating macroexpansion, full-featured editing, inspecting, and debugging environment, true compiled code, tight connection to geometry. Considering existing languages two fashionable alternatives exist: augmented CAD systems, and proprietary from-the-ground-up KBE languages. Inspecting the first clarifies the distinct and complementary roles of true KBE on one hand, and CAD on the other. On the other hand the proprietary solution will fail to keep up with the cutting edge developments inherent in the field of engineering and design computing.

B. Design and Engineering Engine

A DEE is defined[3] as an advanced design environment that supports and accelerates the design process of complex products through the automation of non-creative and repetitive design activities. Figure 1 shows the DEE concept. Example architectures of the DEE can be found in [12] and [13].

The Initiator is responsible for providing feasible starting parameter values for the instantiation of the search process. The search process selects a set of parameter values and instantiates the (parametric) product model. The MMG is responsible for instantiation of the product model and extracting different *views* on the model in the form of report files, capturing discipline specific model information, e.g. aerodynamic mesh or a Finite Element model, to facilitate the related expert tools. The discipline expert tools are responsible for testing one or several properties of the product and valuation the resulting behavior (e.g. structural response or aerodynamic performance. The performance synthesis is responsible for determining the objective and constraint values based on the performance values. The search process is responsible for checking convergence of the design solution and the compliance of the product's properties with the design requirements. The multi-agent task environment is responsible for communication and facilitates data transfer between the individual tools.

The design process connects these tools. Figure 1 is actually an abstract representation of that design process. The design process is a human developed process. First the human approaches to an engineering design problem are discussed and related to the design process. Then the design process is elaborated in section IV, discussing the process step by step.

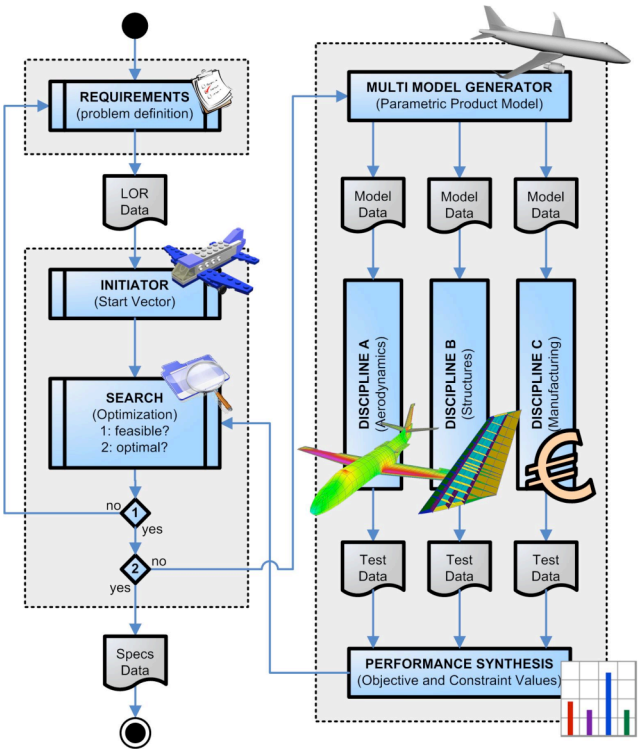

Figure 1: The Design and Engineering Engine (DEE); left the main design process flow; right the Multi-model generator and the analysis tools

III. Human approaches to an engineering design problem

This section discusses methods humans use to solve an engineering design problem. As stated before the feasilization[5] 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' solution (e.g. educated guess). These methods are solution capturing and solution inter- and extrapolation. In the next section these five methods are elaborated.

A. Problem simplification

Problem simplification, or modeling, provides the best approach to find an initial solution space that can be obtained with the information and engineering knowledge available at the start of the design problem. Via simplifying the design space the problem complexity is reduced. This enables the designer to make a (relatively) quick scan of a simplified design space, to identify feasible and 'optimal' areas that can be further investigated using a more detailed analysis. The design problem is simplified such that only the driving requirements (e.g. function), design option parameters (product configuration), and constraints (behavior) are taken into account. A simplified problem is defined as a reduced set of the requirements (*i*) and is solved by using a reduced number of design options (*ii*) and simplified behavior (*iii*).

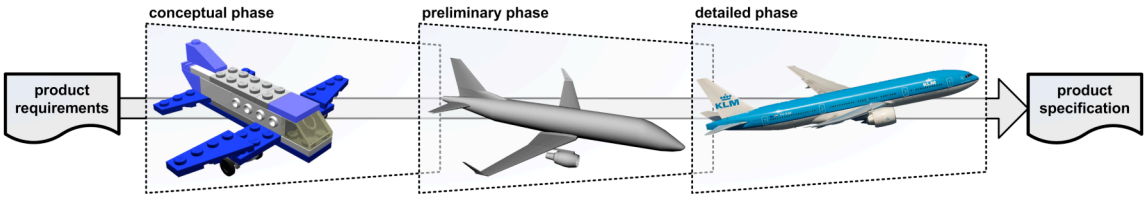

Figure 2, The phases resulting from problem simplification; commonly referred to as the conceptual, preliminary, and detailed phase

i. The requirements are divided into functional requirements, performance requirements, and constraint requirements. The functional requirements are a set of test cases. These test cases are a representation of the product functions and specify the properties of the environment, e.g. only in-plane load cases (other functions are left out, e.g. out-of-plane loads). The performance requirements in structural design are described in weight and costs, and the constraints are failure criteria like structural buckling, material strength.

ii. The design options define the parametric product model. A reduced version is obtained by including only the driving model design options. The resulting problem has a reduced design domain, which can be described by fewer parameters than the complete problem. For instance in conceptual design the panel parametric model does not encompass connection elements, like rivets.

iii. The constraint performance values are determined by analysis during the test process. The test models are based on approximations of system boundaries, system geometry, and system functions. E.g. for a panel the system boundaries are schematized via approximation of the polygonal edge with a rectangular shape and interaction with surrounding structure is left out. The system geometry of the real product and the model differ, e.g. a curved skin is approximated as a flat plate. Finally, only the main system functions (e.g. load transport, not noise insulation) are used for model definition, and functional interference with surroundings is left out, e.g. behavioral coupling between different products, in structural design the buckling interaction between panels.

Via iteratively decreasing the level of simplification the designer attempts to find the 'optimal' and feasible solution to the complete design problem. Commonly this iterative process is modeled by multiple phases (see Figure 2). Each phase has a unique level of problem simplification, e.g. a conceptual or preliminary level. The number of phases required for a specific design problem depends on the problem complexity and the available resources. During the design process these phases are executed iteratively, if the designer is unable to find a feasible solution space, the previous phase must be repeated. This process is stopped if no feasible solution space is found, or if a feasible solution is obtained for the complete problem.

B. Problem decomposition

A human uses decomposition, or analysis and synthesis, to break up a complex problem into multiple smaller problems concerning part of the design space. One of the complications with the decomposition of a design problem is that it assumes that the sum of partial solutions forms a feasible overall solution. In reality this is often not the

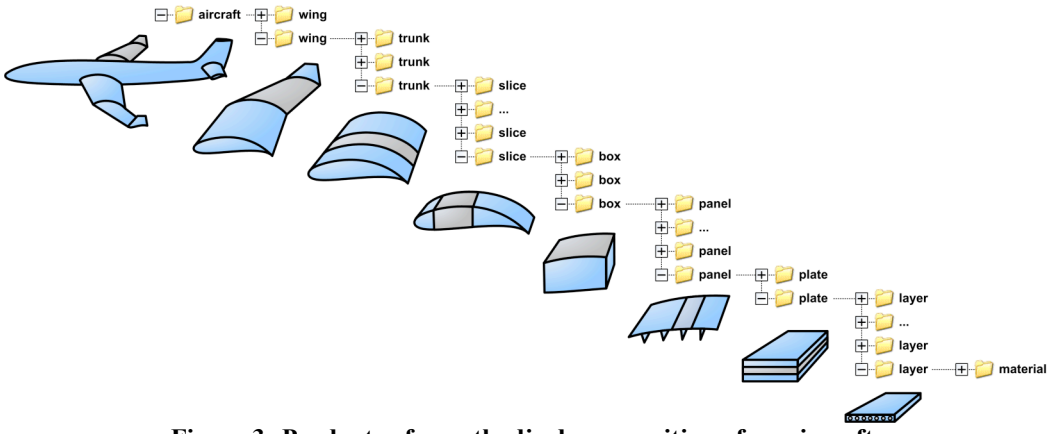

Figure 3: Products of a methodic decomposition of an aircraft

case. Interference between the partial solutions can invalidate the applicability of the sum of partial solutions. This problem is addressed. After the components, the product is designed; incorporating checks on failure modes associated with the combination of component solutions, e.g. rib crushing due to bending of the wing.

Two general types of decomposition exist, the methodic and systematic approach[14]. A systematic approach is to select a subset of the product properties (characteristics) and decompose the product based only that subset, e.g. discipline oriented (aerodynamics or structures) as used in the test phase. The advantage is that this methodology can be used for multiple products. The methodic approach is based on a decomposition based on all properties, and hence is restricted to the defined product. In principle, a product can have multiple systematic decompositions and only a single methodic decomposition. The systematic approach is commonly used, but the methodic approach is not. Here we focus at the initial implementation of the methodic approach.

The decomposition is based on function and design option. The functions product components depend on the function and design option of the product. Since a component is again a product of its components this process continues until a physical relation between function, model and behavior is known or can be chosen. Typically, in aircraft design, these products are called 'design values' (e.g. material properties), known feasible design solution specifications. Hence the decomposition process reveals a hierarchical network of products (see Figure 3). Methodic decomposition creates a possibility to capture design knowledge at different abstraction levels, e.g. from material, to aircraft. An advantage of the structured 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.

In another paper[15] by the authors an application of the methodic decomposition approach is presented for the structural design of a wing structure. The XML language[7] is used to capture the hierarchical product structure.

C. Trial and error methods

Three product design categories can be identified; routine, innovative, and creative designs[16], illustrated in Figure 4. 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.

Through experience a human gains knowledge and experience is gained by trial and error, a search process. In case of non-routine designs, the designer has not sufficient knowledge to define a design solution, since a physical relation between function, model, and behavior 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 behavior.

Search algorithms are used to mimic this human capability. In this work two technologies are used; a design of experiments[17] (DoE) and an algorithm for constrained nonlinear optimization, called 'fmincon' (part of the Matlab optimization toolbox). These were selected based on availability.

D. Solution interpolation

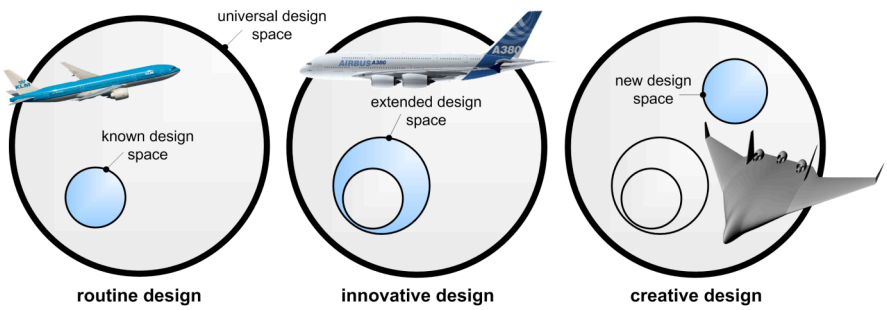

Figure 4, Product design categories; routine, innovative, and creative designs

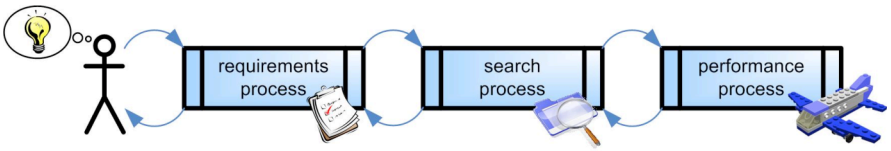

Figure 5: Generic design process sub-processes; requirements, search, and performance process

Solution interpolation, or analogy, humans are able to find 'in-between' solutions. Humans also use this capability across different domains. In case of interpolation the method is constrained within one problem domain.

The iterative processes can be improved by reusing product design knowledge. The obtained product designs define a mapping between the problem and the solution space, which can be directly applied in the design process. However, one must realize that this mapping is only valid within the defined problem domain (interpolation). If a problem is defined outside that domain (extrapolation) the solution should not be trusted. In these cases the knowledge must be updated to include the new design space.

Known methods that mimic this capability are e.g. case-based reasoning, and surrogate modeling. In the field of aircraft design the method developed by Torenbeek[18] is a good example of the strength of analogy, using analytical formulae to find an initial aircraft weight based on previous aircraft designs. In this work a Matlab kriging toolbox is used, called DACE[19], selected based on availability.

E. Solution capturing

Solution capturing, or memorizing, is a premise of all previous methods. How the human brain actually captures knowledge is not yet clear. One group states that humans capture experiences in patterns on different abstraction levels[20], matching the human decomposition approach. The method is included for completeness, but a discussion on this topic is out of scope of this paper.

IV. Engineering design process

The engineering design process can be divided into three main process categories, the requirements, search, and performance process (see Figure 5). These processes can also be identified in the DEE process, see Figure 1.The requirements process translates the customer demands and company demands into a list of requirements. The result is a range of possible product requirements. Here, the customer requirements are defined in the same 'language' as the requirements and only the conceptual design phase is addressed. A proper discussion on this phase is out of the scope of this paper. The search process aims at finding a solution that meets the requirements by trial and error. The performance process aims at finding the performance values of the product. The values are required to evaluate the fitness of each instantiation considered, supporting the search process. The discussed design process is based on previous mentioned human approaches and is only one of the possible implementations of the design process.

A. Search process

The MDO process attacks this search problem. The search process aims at finding a set of 'optimal' system specifications (model and behavior properties) to a certain set of requirements (functions, performances, and constraints). Basically, it tries to find the physical relations between function, model, and behavior[16]. The term model means a simplified description of a system. Important to note here is that during the design process the model has certain level(s) of simplification (further elaborated in section II.A). The difference between model and behavior

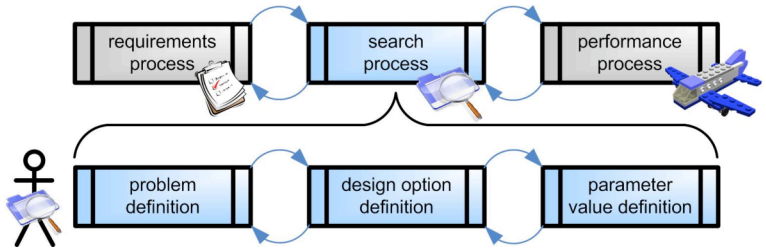

Figure 6: Basic search processes; problem definition, design option definition, and parameter definition

is that the term model refers to the *intrinsic* properties of the system, and the term behavior refers to the *extrinsic* properties of the system. For example length is an intrinsic property, independent of the product environment, and drag an extrinsic property, dependent on the environment.

The search process needs start vectors to be initiated, obtained by design feasilization[5]. The search process can be started after the problem is adapted to the level of simplification (section III.A). The search process involves three steps; problem definition (how are the requirements translated to a search problem), design option definition (what are possible model configurations), and parameter definition (what are possible parameter values), illustrated in Figure 6. The steps are applied iteratively. At the end the optimal parameter set (which parameter values perform best), and design option set (which performance is best) are selected.

To optimal reuse previous solutions in the design process, an even distribution of the solutions in the design space should be obtained. These solutions can then be used for interpolation (and extrapolation to a certain extent) and find new (estimated) solutions. Here for the requirements are discretized in one or multiple sets of requirements, turning the N-dimensional search problem into a set of M-dimensional search problems. The discretization applies to function, performance, and constraint requirements. If a function is specified as a range it is translated to a set of fixed function values, in case of multiple objectives a fixed set of Pareto points is used, and if criteria or parameter bounds are variable also these are discretized to a set of values. The number of requirement sets depends on the number of discretizations, e.g. four discretizations transforming a range into ten cases results in ten thousand separate requirement sets.

Per requirements set one or more design option sets (adapted to the level of simplification) are defined that are assumed to be able to meet the requirements. Often, these design options have disconnected design spaces, making it a search problem of a discontinuous design space. If the design option parameters are properly defined as continuous variables, the search process can be simplified to a set of continuous search problems, a size problem. In this implementation the design options are assumed to be properly defined and are investigated separately. First determining the individual performances, then making a trade-off based on the requirements. The trade-off is simplified to a selection based on objective function value, since this is identical for every design option.

Depending on the search method, per design option one or multiple parameter value sets (start vectors) are defined. For each set the performance is evaluated and the 'optimal' set is selected. To determine the performance multiple tests are performed. Via these tests the behavior (extrinsic) properties are obtained, commonly analyzed per discipline. Each discipline has a specific view (e.g. structural model, aerodynamic model) on the product, taking only a subset of the model (intrinsic) properties into account. This process is referred to as the performance process, discussed in the next section. The search problem can be defined as:

$$\min_x f(x) = \sum a_i \cdot f_i(x)$$

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

The design option definition, performance evaluation, and design option selection can be implemented integrally by using a topology optimization method. Another possibility would be to use the multi-level, multi-dimensional characteristic of the problem (see section II.0), e.g. Bi-Level Integrated System Synthesis[21] (BLISS). In this implementation the search is discretized; each design option is evaluated individually and the best performing design option is selected based on the objective function. To get an initial set of parameter value sets that comply with the shape of the solution space, a Design of Experiments (DoE) is used. The experiments are analyzed separately, the sizing is performed by a single level optimizer, e.g. Sequential Quadratic Programming (SQP).

B. 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 for the objective and constraints), (model) analysis process, and multiple test processes. Analysis is a

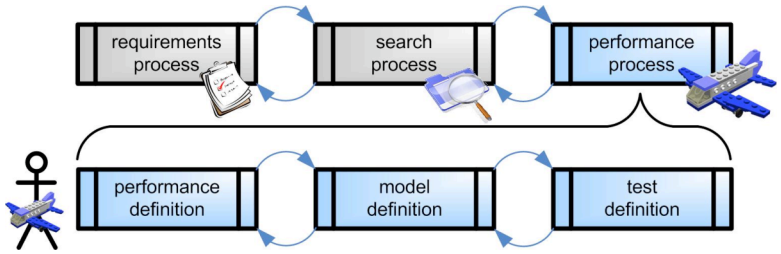

Figure 7: Performance analysis process steps; model analysis, behavior test, performance definition, and performance synthesis

systematic decomposition (see section III.0) of the product properties, generating a number of sub-products (discipline specific, e.g. structural model or aerodynamic model) contemplating a sub-set of properties. The sum of the separate evaluations is assumed to be valid for the complete problem.

The performance process translates the performance values to constraint and objective values. In case of a multi-objective statement the individual objective values should be changed to relative values to be able to compare objectives with different ranges.

The model process defines all required model (intrinsic) properties. As seems logical, before the analysis is started first a generic product 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 restricted to geometry properties of the product, but surely not always (e.g. cost modeling, equivalent structural modeling[15]). Technologies to capture this process generally use model primitives, example technologies are: CAD (using geometrical primitives like surfaces), KBE (using knowledge primitives like high level primitives[3]), and symbiotic methods (see section II.A).

Each test process receives its view on the product from the modeler. Before the test can be performed the relevant functions are translated to a test case (specifying an operating condition), and the environment model (again an abstraction of reality). The product behavior is obtained by determining the impact of the environment model on the product model in a certain condition, specified by a test case. The performance of the product is obtained by applying criteria to the behavior. A problem that can occur during a test process is that the models yield unrealistic results (and must be adapted). To overcome this problem often a convergence step is included to check whether the obtained behavior is within the range of expected values. Implementation depends on the specific discipline. The relations between the model, test, and synthesis processes can be illustrated with a set of simple formulae:

$$\begin{aligned}
model_{product} &= f_1(properties_{product}) \\
model_{environment} &= f_2(properties_{product}, properties_{environment}) \\
criteria &= f_3(model_{product}, model_{environment}) \\
behavior &= f_4(model_{product}, model_{environment}) \\
performance &= f_5(criteria, behavior) \\
objective &= f_6(performance, properties_{product}) \\
constraint &= f_7(performance, properties_{product})
\end{aligned} \tag{2}$$

The previously described process can only exist if all the tools are able to communicate. Adding to the complexity the tools are often on different computers, with different operating systems, and on different networks using different pseudo languages or requiring direct control by humans. MDO needs a support framework for their implementation into industry. The framework can be pull based, push based, or a combination of the both. Important to note is that Berends[8] uses four scoping levels to be able to support a MDO problem with the organization, framework, tool, and data level.

V. Engineering primitives

This section discusses first the background of the engineering primitives by a short discussion on an existing framework used to implement the MDO problem and by an analysis of the word 'knowledge', respectively in section A and B. This is followed by the introduction of the individual primitives in sections C, D, E, and F, respectively called the property, process, system, environment primitive.

A. MDO frameworks

The engineering process primitives are focused at an implementation in a MDO problem. Multiple MDO frameworks already exist and have coped with the different requirements. Berends[8] uses four scoping levels in the implementation a multi-agent task environment framework. The scoping levels are respectively the organization, framework, tool, and data level.

On the organization level the MDO problem is executed and managed, responsible for resource management. The aim at organization level is to find a solution that meets all the problem requirements. Ensured is that the necessary resources are available and guards schedule and resource constraints. The framework goal is to produce a MDO problem-solving environment, responsible for resource interfacing. The framework does not zoom in to tool level but only connects and integrates individual tools, such that the problem can be solved. Also a quality control function is addressed, the tool interfaces are validated. At the tool level the objective is a correct functioning of discipline analysis tools, responsible for delivering process execution support. The discipline specialists belong to this level. The data level is handles product information, and is responsible for information flow control.

These levels fulfill MDO functions, part of the product engineering field. Translating the words organization, framework, tool, and data to the product engineering language one could obtain words like environment, system, process, and property. This becomes really interesting when it is connected to the definition of knowledge.

B. Knowledge categories

The engineering primitives are based on design process knowledge. The design process has been elaborated in section II, but what do we mean by knowledge? Knowledge is defined[22] as (1) the information, understanding and skills that you gain through education or experience, (2) the state of knowing about a particular fact or situation.

The first definition identifies three knowledge categories, understanding, skill and information. These categories can be linked to the before mentioned scoping levels. Information connects to the lowest level of data and property. Skill is the knowledge of specialist know-how and could be linked to tool or process level. Understanding concerns knowledge about certain situations and the ability to act to that specific situation, and could therefore be linked to the second scoping level of framework and system. This leaves the first scoping level, which apparently supersedes knowledge. The first scoping level of organization and environment can be linked to wisdom.

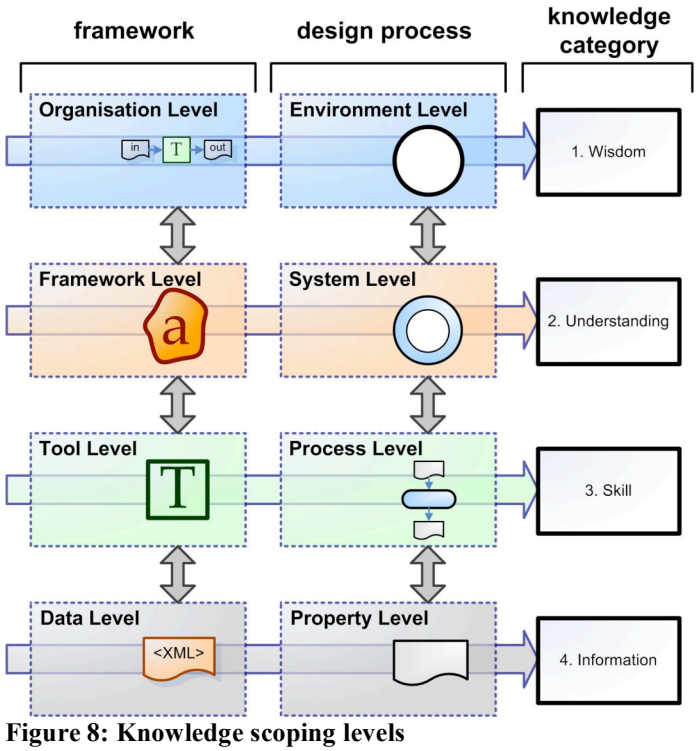

Figure 8: Knowledge scoping levels

Wisdom is defined as (1) the ability to make sensible decisions and give good advice because of the experience and knowledge that you have, (2) how sensible something is, (3) the knowledge that a society or culture has gained over a long period of time.

This decomposition fits the approach humans used to model the environment, at least in the western hemisphere. The first scoping level would be to consider everything, hence the

environment. The next step is to address the world as a collection of separate systems, interacting with each other. This gives the advantage that the systems can be addressed individually, however still influenced by other systems. The next simplification disregards the other systems and considers the system as a closed system, yielding actually a process only changing when the input is changed. The final step is to neglect time, transforming dynamic into static. This static closed system is nothing more then a collection of properties. The four scoping levels are illustrated in Figure 8.

To include the different human knowledge types, these four scoping levels are used for the development of four 'knowledge' primitives, one for every scoping level. An initial implementation is presented in the following sections. The development of the property and process primitives is completed. The system primitive can at this moment cope with single level implementations (the decomposition capability is not yet captured). The environment primitive is still only a theoretical object.

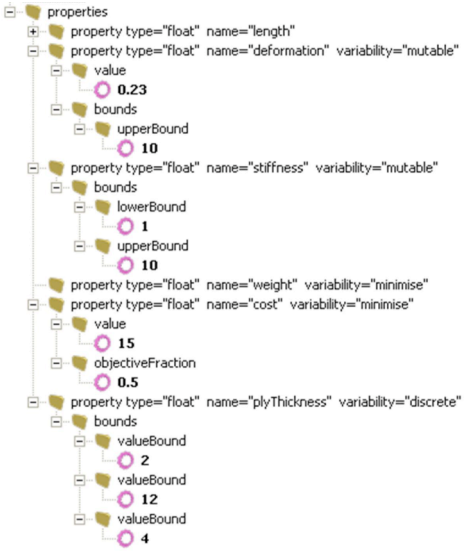

Figure 10: Example XML input for the instantiation of a Property primitive

C. Property

The highest simplified knowledge primitives are property primitives. A property defines one or more characteristics of something. Information like name, id, value, value type, and specific design space knowledge, e.g. bounds, design space, mutability, feasibility, is captured by the primitives. The property primitive is unaware of its primitive parent and is defined by its parent, which can replace it at any time, supporting dynamic data typing[11]. All primitives are designed such that the user only has to specify the relevant information at that level, obtaining a declarative programming style, a high-level programming language that can be used to solve problems without requiring the programmer to specify an exact procedure to be followed.

An UML (Unified Modeling Language) diagram of an example implementation is presented in Figure 9. A *Primitive* superclass is defined, capturing generic information and methods required for all objects. The *Property* primitive obtains the knowledge via inheritance from the *Primitive*, and has as basic attribute a value. Since different property types exist, multiple classes are defined that use the *Property* primitive as superclass; the *Number*, *Array*, *Text*, and *Structure* primitives.

An interface is defined such that the primitives can be instantiated by XML files. An example XML file is presented in Figure 10. A XML interface is used because it offers an editing and inspection environment and the implementation is relatively easy.

D. Process

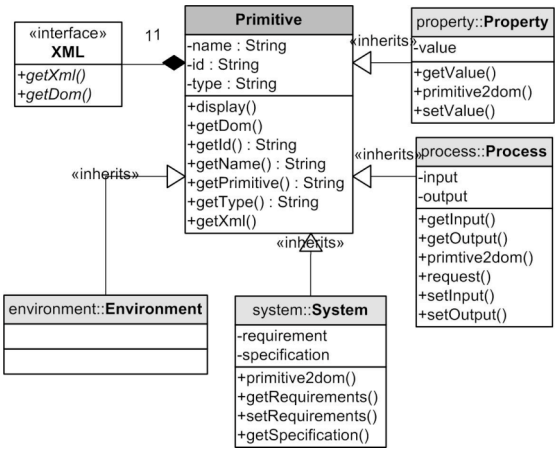

Figure 9: An overview of the four primitives used and the xml interface in the current implementation

A process is defined as (a series of) things that are done in order to achieve a particular result. Design knowledge on how to relate design option parameter values to test cases values and requested performances values is captured on this level. The processes described in section IV.B belong to these primitives.

The *Process* primitive inherits (like *Property*) from *Primitive*, and has input, output and source attributes. The *Process* is a superclass of specific processes, *Function*, *ProcessOfProcesses*, and *AnalysisProcess*. For the implementation the last two a lazy[9] evaluation is used instead of an eager one, mimicking the request methodology described by Berends[8]. In that paper Berends defines that this level is the field of discipline specialist actors and their functional coding style. In a more complete implementation the capabilities can be extended to include dependency tracking[11].

An example interface of the *Function* and *AnalysisProcess* are presented in Figure 11.

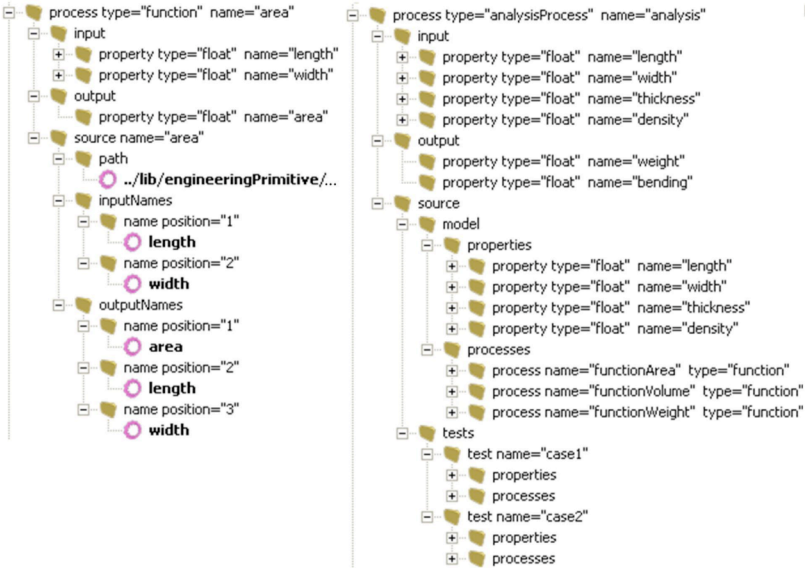

Figure 11, Example XML interfaces for *Process* primitives; *Function*, and *AnalysisProcess*

E. System

A system is an organized set of theories, based on sub-systems and processes to make it function. Design knowledge on how to adapt something to better suit the environment. This can be supported by a trial and error process (e.g. search methods) or by reusing previous solutions (e.g. regression methods). The system changes its identity if the environment requirements change, supported by runtime value caching. The processes described in section IV.A belong to these primitives.

The *System* inherits from *Primitive*, and has requirements, specification and core as attributes. Specific implementations are used to implement the trial and error process and the interpolation process. The first is implemented in a so-called *Performance* primitive, the other in a so-called *RequirementRange* primitive. The first uses the earlier mentioned DoE and SQP, and the second DACE.

An example of the *Performance* and *RequirementRange* interface is depicted in Figure 12. There one can see that all other primitive types, *Property*, *Process*, and *System* are used in the implementations. This is a good example of the amount of code reuse that can be obtained by using this approach.

F. Environment

Environment is defined as the conditions that affect behavior and development of something. Design knowledge on how to assemble a set of systems to meet a certain set of conditions is to be captured by the *Environment* primitive. However, this primitive is not yet implemented. An example implementation will follow in a future paper.

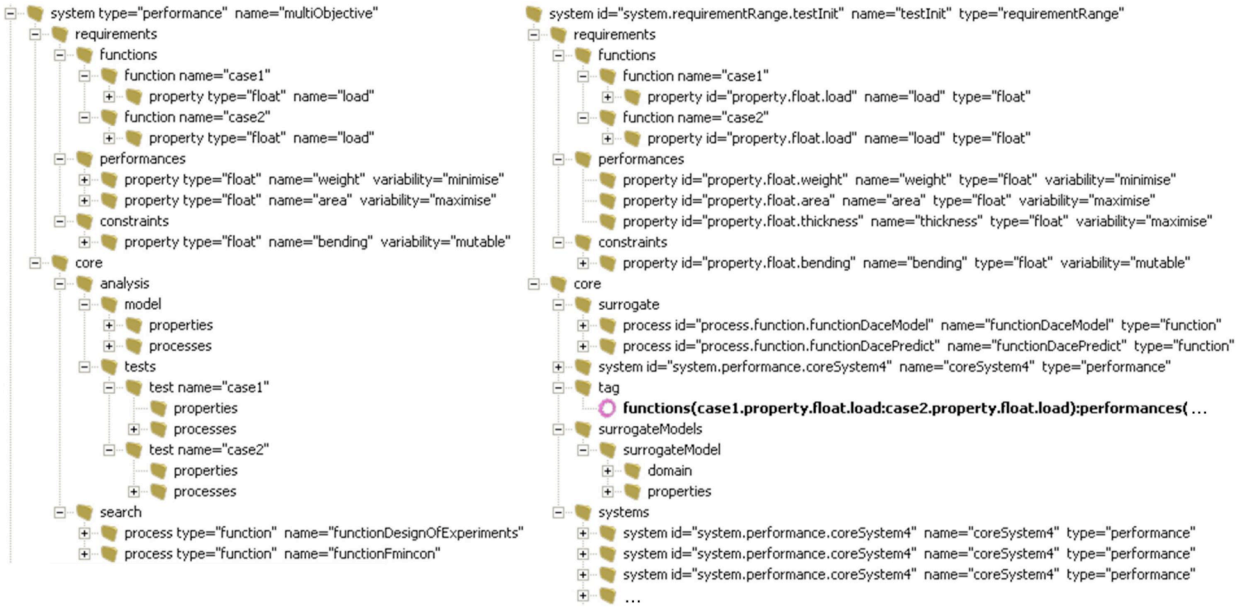

Figure 12, Example XML interfaces for *System* primitives; *Performance*, and *RequirementRange*

VI. Discussion

This paper discussed the methods human use to attach an engineering design problem. The methods are based on human knowledge. In an analysis of this knowledge four scoping levels were identified, each one considering a different type of knowledge. The four scoping levels are called environment, system, process, and property, and address respectively the knowledge types; wisdom, understanding, skill, and information. To capture these different types of knowledge four engineering primitives are being developed.

The implementation featured the power of reusing code, which besides supporting uniqueness of knowledge also saves resources. This advantage shows best in new obtained solutions, featuring an identical structure as the initial proposed problem. Hence, the solutions are stored in the same grammar as was used to describe the problem.

Another benefit is the hierarchical structure of the problem. Complexity hiding is used to support inspecting of the problem and/or solution. At each level relevant information is presented, while by 'zooming in' the information can be traced back to its origin(s).

However the presented solution still has to prove its power in more extensive applications.

Next development steps focus on an implementation of the environment primitive, enabling a multi-level system approach, and of a conceptual design test case to evaluate the performance of the methodology.

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]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

[5]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

[6]Buskes, C., "The Genealogy of knowledge", Tilburg University Press, Tilburg, the Netherlands, 1998

[7]W3C, "Extensible Markup Language (XML) 1.1", second edition, URL: http://www.w3.org/TR/xml11 [cited 17 March 2008]

8Berends, J.P.T.J. M.J.L. van Tooren, "Design of a Multi-Agent Task Environment Framework to support Multidisciplinary Design and Optimisation", 45th AIAA Aerospace Sciences Meeting and Exhibit, AIAA-2007-0969, Reno, NV, USA, 2007

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

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

[11]Cooper, D., La Rocca, G., "Knowledge-based Techniques for Developing Engineering Applications in 21st Century", 7th AIAA Aviation Technology, Integration and Operations Conference (ATIO), Belfast, Northern Ireland, AIAA 2007-7711

[12]Cerulli, C, E.J. Schut, J.P.T.J. Berends, M.J.L. van Tooren,"Tail Optimization and Redesign in a Multi Agent Task Environment", 47th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Newport, RI, USA, 2006

[13]Van der Laan, T., van Tooren, M.J.L., "Parametric Modeling of Movables for Structural Analysis", Journal of Aircraft, Vol. 42, No.6, 2005, pp 1605-1613

[14]Foucault, M., "The order of things", Vintage Books edition, Random House, 1970

[15]Schut, E.J., and M.J.L. van Tooren, "Feasilization of a structural wing design problem", 49th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, Schaumburg, Illinois, USA, 2008, ...

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

[17]Lanzi, L., L_Latincube: design of experiments, Software Package, Department. of Aerospace Engineering, Politecnico di Milano, Italy, 2004

[18]Torenbeek, E., "Synthesis of subsonic airplane design", Kluwer Academic, 1996

[19]Lophaven, S.N., Nielsen, H.B., Sondergaard, J., "DACE: A Matlab Kriging Toolbox", version 2.0, Technical University of Denmark, technical report IMM-TR-2002-12, 2002

[20]Hawkins, J., "On intelligence", Owl books, New York, 2005

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

[22]Hornby, A.S., "Oxford Advanced Learner's Dictionary", 6th edition, Oxford University Press, 2000