Formalizing Technology Descriptions for Selection During Conceptual Design

Roelofs, Martijn; Vos, Roelof

DOI
10.2514/6.2019-0816

Publication date
2019

Document Version
Accepted author manuscript

Published in
AIAA Scitech 2019 Forum

Former title: Formalizing Technology Descriptions for Selection During Conceptual Design

An Engineering Systems Graph Description Enabling Automated Analysis Mapping and Execution

Martijn N. Roelofs* and Roelof Vos†
Delft University of Technology, Delft, Zuid-Holland, 2600AA, The Netherlands

Evaluation and assessment of novel technologies for aerospace applications is essential for business strategy and decision making regarding development efforts. However, technology evaluation and assessment are challenging to perform objectively using a structured approach. As a first step towards a more objective and structured approach a graph-based description of engineering systems is described herein. Analyses can be applied to such a description through pattern matching, after which the quantities of interest can be computed by an automated algorithm using a dependency graph. The approach is applied to a simplified aircraft model, to perform a mission analysis and compute fuel burn. It is shown the method successfully computes the required parameter and is easily adapted to analyze an electric aircraft as well.

I. Nomenclature

A	Analysis
B	Behavior
$\mathcal{B}$	Set of behaviors
C	Component
$\mathcal{F}$	Set of flows
$G_{\mathbb{C}}$	Context graph
G, H	Graph
g	Constraint function
$\mathcal{P}$	Set of parameters
$\mathbb{P}$	Union of parameter sets
p_{ij}	Path from i to j
R	Set of relations
r	Relation
$\mathcal{S}$	Set of states
s	State
T	Technology
u	Node
$\mathcal{U}$	Set of nodes

Greek symbols

γ	Component flow interaction
μ	Parameter mapping function
ν	Input/Output mapping function

Subscripts and superscripts

$'$	Modification
k	Hierarchy level
in	Input
out	Output

Acronyms

DAG	Directed Acyclic Graph
FPG	Fundamental Problem Graph
IRL	Integration Readiness Level
MDO	Multidisciplinary Design Optimization
PSG	Problem Solution Graph
SoI	System of Interest
SRL	System Readiness Level
TCM	Technology Compatibility Matrix
TRL	Technology Readiness Level
QoI	Quantity of Interest
QPT	Qualitative Process Theory

II. Introduction

Early in the conceptual design of engineering systems it is of utmost importance to select those components, technologies and configurations that are most likely to achieve the specified requirements within a certain timeframe, while reducing risk and cost (of development, manufacturing and operations). Any part of the system that may or may not be included can be regarded as a technology. Alternatively, any technical solution for a certain functional requirement can also be viewed as a technology. Technologies therefore range from differing parameters (e.g. material strength), to components

*PhD Candidate, Faculty of Aerospace Engineering, Delft University of Technology, Kluyverweg 1 2629HS, Delft, The Netherlands
†Assistant Professor, Faculty of Aerospace Engineering, Delft University of Technology, Kluyverweg 1 2629HS, Delft, The Netherlands, Associate Fellow AIAA

(e.g. electric actuator instead of hydraulic), to behaviors (e.g. a structural element doubling as electric conductor), and to entire systems (e.g. a fixed-wing aircraft as opposed to a helicopter). Previously, a review on technology selection [1] identified the following challenges with technology evaluation and selection:

1) The application of design tools is limited to a specific vehicle type, because the sizing method is fixed [2]
2) Flexibility and scalability of disciplinary analysis tools is lacking [2]
3) Parameterization of geometry proves challenging in a generalized sense (i.e. problem specific parameterizations are possible, but geometry parameterization that holds for any problem is challenging, at the least)
4) Modeling of technologies is usually avoided and replaced by impact factors
5) Extensive use of expert judgment raises challenges due to subjectivity, conservatism, overconfidence and lack of experts

Additionally, from discussions with practitioners at Saab, the following challenges with technology selection in the conceptual design phase were identified: (i) non-performance metrics (e.g. the effect of improved human-machine interaction) need to be included, although prove difficult to define and quantify, (ii) technology descriptions are not yet meaningful, and cannot traverse from detailed to high-level descriptions, nor capture quantifiable effects and enabling fair comparison between technologies, (iii) finding the best technology portfolio without enumerating all possible combinations cannot yet be done objectively, (iv) the assessment of dependencies between technologies is too subjective, and (v) when uncertainty is quantified, decision making is impaired in case of wide uncertainty bands.

Several conclusions can be drawn from these challenges. Metrics that are not quantifiable cannot be addressed, and hence, technologies affecting those have no meaning. Alternatively, additional analysis methods should be developed. Uncertainty should be associated with technology (and system) readiness level (both on impact and development time). Additionally, the amount of uncertainty should be limited or decision-making in the event of large uncertainty should be supported. Insight is more important at this stage than arriving at a most optimal result, hence preference is given to evaluating current possibilities, rather than performing optimization. Therefore, a structured and traceable solution to technology definition, evaluation and selection is sought.

Conventionally, technology selection is performed by collecting technology TRL and IRL levels, describing their effects in an impact matrix and their compatibility in a Technology Compatibility Matrix (TCM). Such an approach was taken in the works by Amadori et al. [3, 4], who depict the process as in Figure 1. The data collection phase is essentially carried out completely using expert judgment, with limited traceability and objectivity.

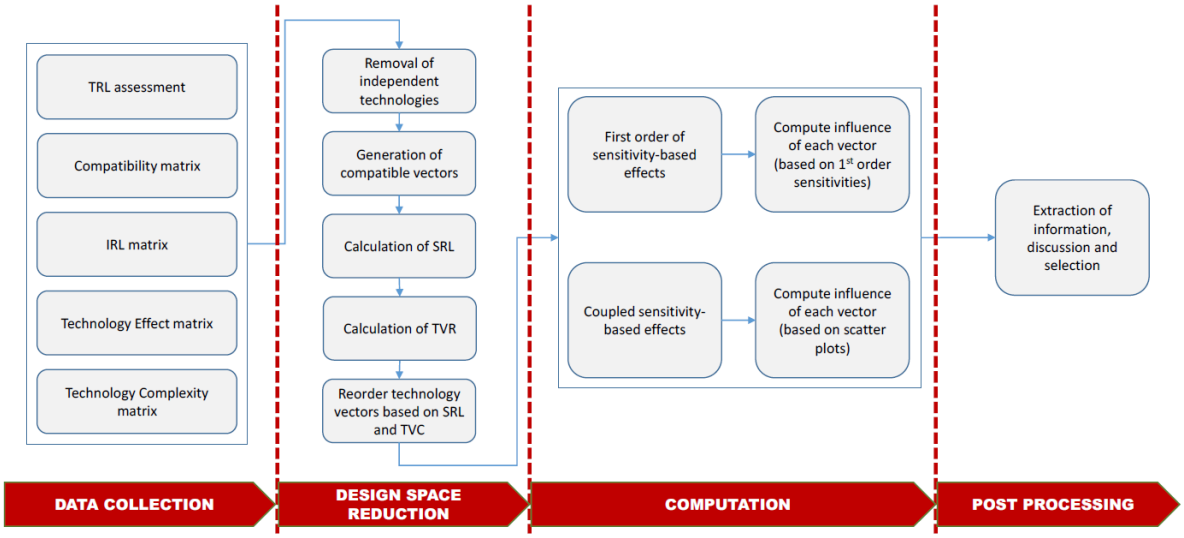

Fig. 1 Technology evaluation and selection process [4]

Before any quantitative evaluation of technologies can be performed, technologies and their effects have to be defined, which requires a formal modeling paradigm if each is to be evaluated equivalently. Evidently, other efforts have been made towards similar goals. Mostly, systems are described by functions/behaviors, leading to the topic of functional decomposition [5–10], which is mainly used in design synthesis. This allows automatic selection of components to fulfill certain requirements. Additionally, representing a system in terms of its behavior instead of a

certain fixed parameterization gains popularity in model-based systems engineering [6, 11–14], which may be attributed to the formalization of behavioral descriptions. While these allow for a structured approach to describe systems and enable automated design synthesis, they all rely on human specified functions, which effectively are only labels. As such, from another modeler's perspective, these labels may have to be different. It would model the same system, but the computer algorithm would not be able to make that deduction (since the labels are unequal).

Efforts for automated execution of analyses have also been conducted. Mainly, a M.Sc. thesis by Ramakers [15] and the work by Van Gent [16, 17] are of particular interest. Both of these refer to the study of Pate et al. [18]. That study describes the construction of a Fundamental Problem Graph (FPG) to deduce relations between analysis tools and the variables of a Multidisciplinary Design Optimization (MDO) problem. From this FPG a Problem Solution Graph (PSG) is derived that defines how a particular MDO problem should be solved.

Based on the challenges exposed above, a methodology is developed herein that aims to enable technology assessment. That is achieved through the following goals: (i) to enable a flexible, yet unambiguous description of SoI and technologies, and (ii) enable flexible analysis of a System of Interest (SoI). A computer algorithm can interpret the SoI description and perform most of the tasks in the data collection phase (shown in Figure 1) automatically. However, the focus of this paper is on the system description only, whereas the development of such an algorithm is delayed to future work. On the other hand, the current method allows analyses to be defined and automatically applied to the SoI, to compute a certain Quantity of Interest. The method is explained in section III, followed by a case study in section IV, where the method is applied to a simple aircraft mission analysis example. Consecutively, the philospophy and results of the presented method are discussed in section V and concluded in section VI, where also future work is elaborated upon.

III. Methodology

An ontology is defined that describes a system of interest (SoI) and the technologies affecting it. This ontology is based on works for functional decomposition, since it is recognized that functions/behaviors do not change and hence may form a fixed basis with which to describe any engineering system. The following constructs make up the ontology: Component, Behavior, Flow, State, Technology, Analysis and Attribute / Parameter, see Table 1. A legend showing these constructs graphically is shown in Figure 2. A notional example of a SoI is depicted in Figure 3 and the details of the ontology are elaborated upon in the remainder of this text.

Table 1 Ontological constructs. The meaning of each definition is detailed in the text.

Construct	Definition
Analysis	$(G_{\mathbb{C}}, \mathcal{P}_A, \mu, \nu)$
Attribute	$(l, q, \mathcal{V}), \mathcal{V} = (s_i, v_i)$
Behavior	$(\mathcal{F}_B, c_B, \mathcal{P}_B)$
Component	$(C_F, \mathcal{B}_C, \mathcal{F}_C, \mathcal{P}_C)$
Flow	$(c_F, \mathcal{P}_F)$
State	$\mathcal{P}_s$
Technology	$G \mapsto H$

Each component can be represented as quadruple $(C_F, \mathcal{B}_C, \mathcal{F}_C, \mathcal{P}_C)$, where C_F is the flow representation of the component (any component has some material manifestation), $\mathcal{B}_C$ are the behaviors performed by the component, $\mathcal{F}_C$ the flows associated with the behavior, which is a subset of the flows associated with the behaviors in $\mathcal{B}_C$ and $\mathcal{P}_C$ is a set of attributes that for example specify mass or geometry.

A Flow is a physical entity with a certain classification c_F, as defined in Table 2. These classifications are based on Hirtz et al. [5], although the exact classification is modified for the energy classes. Note furthermore that a Signal can be carried by an Energy, while Energy in turn can be carried by Material. As such, composite flows may be constructed. In fact, a flow could consist of multiple material flows and/or multiple energy flows. Although Hirtz specified power conjugates of these flows, such as force and pressure, here they are treated at the same level as flows. Especially in aircraft design it is useful to express forces and similar concepts that affect the (energy) flows in Table 2. Therefore, these have been explicitly included in the current ontology as Efforts, shown in Table 3. On each Flow or Effort a set of attributes $\mathcal{P}_F$ may be defined.

To uniquely define a Behavior, three elements need to come together: why, what and how. "Why" are the functions

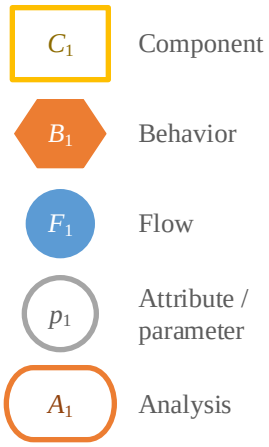

Fig. 2 Legend relating the constructs from Table 1 to a graphical notation.

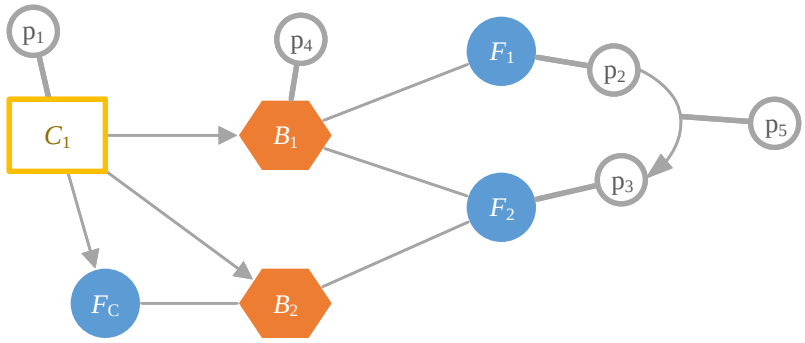

Fig. 3 System of Interest (SoI) notional example. Note that F_C is the flow representation of C_1. The relation between $p2$ and p_3 indicates p_2 is the derivative of p_3 with respect to p_5.

/ requirements of the Component that exhibits the Behavior, "what" are the flows and form of the Component that are necessary to exhibit its Behavior (which in turn fulfills the function) and "how" is a set of physical phenomena taking place in and around that Component. Functions and requirements are not specifically included in the current implementation. Nonetheless, functions are merely a subset of the Behavior (i.e. the part of the Behavior that is intended). Requirements may be added through constraints on attributes and states associated with the Behavior. Using the form of the associated Component in the Behavior definition is expected to inhibit equality / equivalence determination of behaviors for this reason: comparing form (geometry) is not a trivial task. However, a Behavior should be characterized by the form exhibiting it. Therefore, this challenge is left for future study.

In the current implementation, a Behavior is a phenomenon that operates on a set of flows $\mathcal{F}_B$, performing a state change. This state change does not necessarily correspond to a transition (e.g of time): some behaviors are continuous and instantaneous (e.g. aerodynamics or structure dynamics). It is chosen here to classify behaviors into two types of functions on a set of flows. These are summarized in Table 4. The philosophy is that a Flow retains its identity under certain behaviors although its properties may change (Transformation). Alternatively, a Flow is converted into a different class of Flow (Transmutation). A behavior may be further characterized by a set of attributes $\mathcal{P}_B$. As such, a Behavior is represented as a triple $(\mathcal{F}_B, c_B, \mathcal{P}_B)$, where c_B is a class of basis behaviors from Table 4.

Components can be assemblies of other components, and similarly, behaviors can be composed of other behaviors (and flows). Each of these decompositions leads to a more detailed description of the system; a finer granularity. As such, a hierarchy can be formed that describes the relationship between sub-component / behavior and assembly / behavior. A trivialized example is shown in Figure 4, which depicts the essence of the remainder of this paragraph. The hierarchy forms a tree, where the root node is the highest-level component or behavior. Each node branches of in at least two other nodes, forming the next level in the hierarchy. At each level k, the nodes u_i^k are connected through flows (forming interactions by stating flow equivalence between components or behaviors). Formally, $u_i^k \mapsto (\mathcal{U}^{k+1}, \{\gamma\})$,

Table 2 Flow basis, based on Hirtz [5]. The Energy class subdivision is modified. The tertiary class has all the properties of its associated secondary class, and likewise for the secondary class w.r.t. the primary class. Furthermore, Material may contain Energy, and Energy may contain Signal.

Primary class	Secondary class	Tertiary class
Material		
	Solid	
	Liquid	
	Gas	
Energy		
	Kinetic	
	Potential	
		Gravitational, Elastic, Electric, Chemical
		Magnetic, Nuclear, Thermal, (Rest)
	Kinetic & Potential	
		Mechanical Rotation, Mechanical Translation
		Mechanical Wave, Radiant (electromagnetic)
Signal		

Table 3 Effort basis. These form conjugates to the (energy) flow basis.

Effort	Conjugate for energy class
Force	Kinetic, Potential Energy
Torque	Mechanical Rotation Energy
Pressure	Potential Energy
Stress	Elastic Energy

where $\mathcal{U}^{k+1}$ is the set of children and $\{\gamma\}$ is a set of unordered pairs that connect a flow from one child to another: $\gamma_i = \{F_v \in \mathcal{F}_{u_n^{k+1}}, F_w \in \mathcal{F}_{u_m^{k+1}}\}$, where $n \neq m$. Hence, each level in this tree forms a graph, that describes the same system or behavior in more detail. It is allowed to add flows and behaviors while descending the hierarchy, but not to remove them. Thus, each flow F_i^k in the graph of level k maps to a flow in the lower level $k+1$: $F_i^k \mapsto F_v^{k+1}$. For each path $p_{ij}^k = (F_i^k, F_j^k)$ there is a transformation taking place described by the behaviors in between: $B_{p_{ij}^k} : F_i^k \mapsto F_j^k$. This path and corresponding transformation must also be present at level $k+1$, although it may be a superset of it (i.e. there is additional behaviors and flows). Hence, $p_{ij}^k \mapsto p_{vw}^{k+1} = (F_v^{k+1}, F_w^{k+1})$, which has the behavior $B_{p_{vw}^{k+1}} \subseteq B_{p_{ij}^k} : F_v^{k+1} \mapsto F_w^{k+1}$. This is exactly the case in Figure 4.

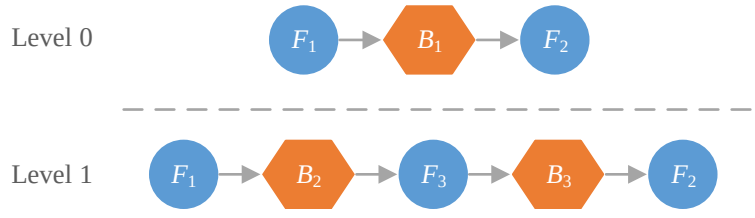

Fig. 4 Hierarchy for behavior B_1, which is broken down into two behaviors B_2 and B_3 at level 1. The total effect of the behavior remains unaffected (F_1 and F_2 are unaffected).

Technologies are pivotal to the present method, yet a detailed description of their implementation and working is delayed to a follow-up paper. Here it suffices to say they represent some function $T : G \mapsto H$, i.e. they map a certain graph G to another graph H. Both of these graphs follow the ontology described presently, and a modification to a

Table 4 Behavior basis: two basis level behaviors with an associated mathematical function. Each Behavior maps a Flow x (or set of flows) onto itself (Transformation) or another set of flows y (Transmutation).

Basis Behavior	Function
Transformation	$x \mapsto x'$
Transmutation	$x \mapsto y, x \neq y$

system description is represented by the function T. So, in short, the method described here to formalize a system description consequently formalizes the description of technologies to be applied as well.

An Attribute is a tuple: $(l, q, \mathcal{V})$, where l is some label, q is a quantity type (e.g. mass, force, etc.) and $\mathcal{V} = (s_i, v_i)$, i.e. a set of tuples that specify a value v_i corresponding to a certain state s_i. Finally, analyses can be defined as a quadruple $(G_{\mathbb{C}}, \mathcal{P}_A, \mu, \boldsymbol{\nu})$. Here $G_{\mathbb{C}}$ is a graph that serves as a context for the analysis, in such a way that it is a graph pattern that will be matched against the SoI graph. As a result, analyses can be applied flexibly to any part of the system of interest that matches the prerequisites for the analysis. The second term $\mathcal{P}_A$ are the analysis parameters (both input and output). These do not necessarily correspond directly to the attributes defined in the system, and multiple system attributes may correspond to only one analysis parameter (if this is a vector, for example). Hence, μ is a function that maps the set of attributes $\mathbb{P}_{G_C}$ collected from $G_{\mathbb{C}}$ to the analysis parameters: $\mu : \mathbb{P}_{G_C} \mapsto \mathcal{P}_A$. Finally, the analysis is a function that takes an input subset of its parameters $\mathcal{P}_A^{\text{in}}$ and maps them to an output subset of its parameters $\mathcal{P}_A^{\text{out}}$. However, different directions of computation may be available, so that a set of functions results:

$$\nu_i : \mathcal{P}_{A_i}^{\text{in}} \mapsto \mathcal{P}_{A_i}^{\text{out}} \qquad \forall \nu_i \in \boldsymbol{\nu} \tag{1}$$

Note that each ν_i has its own set of input and output parameters. This also means that $\mathcal{P}_{A_i}^{\text{in}} \neq \mathcal{P}_{A_j}^{\text{in}} \forall i, j : i \neq j$ and likewise for the output parameter sets. Each ν_i is called a mode of the analysis, and essentially specifies a direction that the analysis can be applied in.

An example pattern $G_{\mathbb{C}}$ is depicted in Figure 5, which is to be mapped onto the SoI shown earlier in Figure 3. From this figure it can be deduced the analysis parameters $\mathcal{P}_A$ are p_a, p_b and p_c. Suppose this analysis has only one mode ν_0:

$$\nu_0 : p_c = p_a \cdot p_b \tag{2}$$

If we furthermore state that F_a and F_1 are of the same class and F_2 is not, then the parameter map μ becomes:

$$\mu : \begin{cases} p_1 \mapsto p_a \\ p_4 \mapsto p_b \\ p_2 \mapsto p_c \end{cases} \tag{3}$$

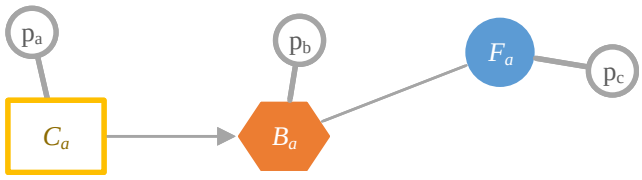

Fig. 5 Example analysis pattern $G_{\mathbb{C}}$ with an application in the SoI from Figure 3

Having a set of analyses and a system of interest (SoI), an algorithm is developed that automatically finds where these analyses can be applied. Then for each application, a map μ is created that maps the attributes in the SoI to the analysis parameters. From that information, a so-called dependency graph is built as shown in Figure 6 (this is similar to the Fundamental Problem Graph [18]). This graph has all attributes in SoI as nodes, as well as analysis nodes. Each analysis node is a triple (A, m, H), where A is the analysis it refers to, m is a mode of analysis A and H is the subgraph of the SoI that matches the analysis pattern. The nodes in the dependency graph are connected with directed edges from attributes to analyses and vice versa. The direction of the edges indicates the flow of information. Finally, an attribute may specify the derivative of another attribute. This derivative is with respect to a third attribute. For example, fuel

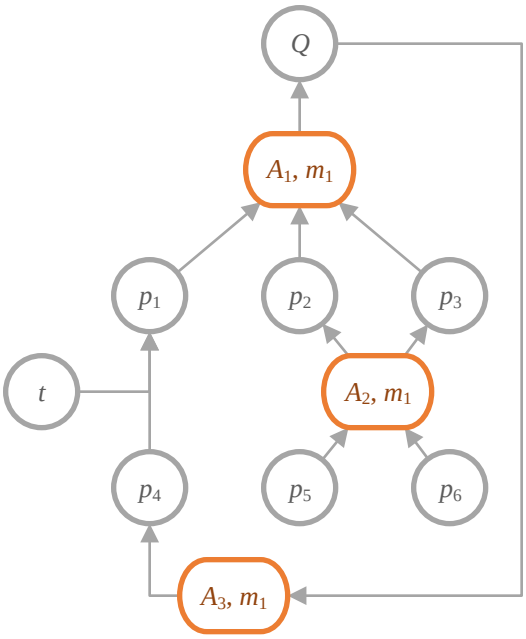

Fig. 6 Dependency graph for Quantity of Interest Q. The parameter p_4 is a derivative of p_1 with respect to t. Three different analyses a_1, a_2 and a_3 compute the various parameters using their first mode m_1. (Note that the mode belongs to that analysis, so m_1 is not the same object for each analysis.)

burn (kg/s) is the derivative of fuel mass (kg) with respect to time. Such a structure is also present in Figure 6 between the parameters p_4 and p_1 with respect to t.

In order to compute a Quantity of Interest (QoI) Q, the initial state s_0 and terminal state s_t need to be specified. The initial state contains the values for all parameters needed to compute Q (and to compute any intermediate values). Furthermore, it contains the initial value for a certain transition variable t. This transition variable is the variable that separates states from one another. Usually it is time, but it could well be any other variable. The terminal state only contains a value for t, such that the algorithm knows when to stop.

The dependency graph G_d may not be connected (there are multiple disjoint components), so any component that does not contain the QoI will be removed (since these are not required to compute Q). Consecutively, all cycles that do not contain a derivative will be removed. Such a cycle represents the following function:

$$\text{Indeterminate cycle} : f(x) \mapsto x \tag{4}$$

Such a cycle may converge, diverge or oscillate on the parameter x. Without knowing the specifics of the analyses executed within such a cycle, it is impossible to deduce which of the three will be the case. Thus it was chosen for the present implementation to omit such cycles altogether. Otherwise, a maximum iteration counter may be set, but the values obtained are still questionable. (It is interesting to note that the transition variable t may actually be such an iteration counter, generalizing the current approach to this case. Such an option has to be investigated in future work, however.) When a derivative is present in the cycle, it does not have to be removed. Basically, at any derivative edge, the computation can be split into two different states, separated by the transition variable (shown in Figure 7). Such a cycle can be written as:

$$\text{Determinate cycle} : \left.\begin{array}{l} g(x) \mapsto y \\ y = \frac{dz}{dt} \\ h(z) \mapsto x \end{array}\right\} = x^{(s_0)} \mapsto \left(\frac{dz}{dt}\right)^{(s_0)} \overset{\Delta t}{\mapsto} z^{(s_1)} \mapsto x^{(s_1)} = f(x^{(s_0)}) \mapsto x^{(s_1)} \tag{5}$$

Thus, x (technically x in state s_1: $x^{(s_1)}$) depends on itself, albeit in a previous state ($x^{(s_0)}$). Hence, such a cycle can be computed without issues.

Removing the derivative edges from the dependency graph we obtain a (or possibly multiple) so-called computation graph(s). These are directed acyclic graphs (DAG) and specify the order of analysis execution (the two blocks in

Figure 7). Each computation graph starts with the attributes which have no predecessors. These are the input for their analysis successors. These compute another set of attributes, which propagate into the next set of analyses, until the end of the graph is reached (those attributes which have no successors). All attribute values computed using these graphs are in the initial state the computation is started in. Finally, for each derivative edge the value of its source attribute (which is in s_0) is used along with the change in transition variable Δt to compute the value of its integral variable (the target attribute) in s_t.

One final remark has to be made regarding the above approach. From the dependency graph it can be derived whether the transition from s_0 to s_t can be performed in one step, or has to be marched with small changes in the transition variable. When a cycle with derivative (see Equation 5) exists in the dependency graph it means that some parameter used to compute Q (or even Q itself) is dependent on its own value in a previous state (as is the case for the dependency graph in Figure 6). Furthermore, if that parameter's derivative is dependent on t (conversely on the state) too. If this is the case, a marching solution is required, that divides Δt into some specified amount of intervals. The computation is started from s_0 and the above procedure is executed to compute each intermediate state s_i until s_t is reached, which is depicted in Figure 7, where *Next state* becomes *Initial state* after each computation step.

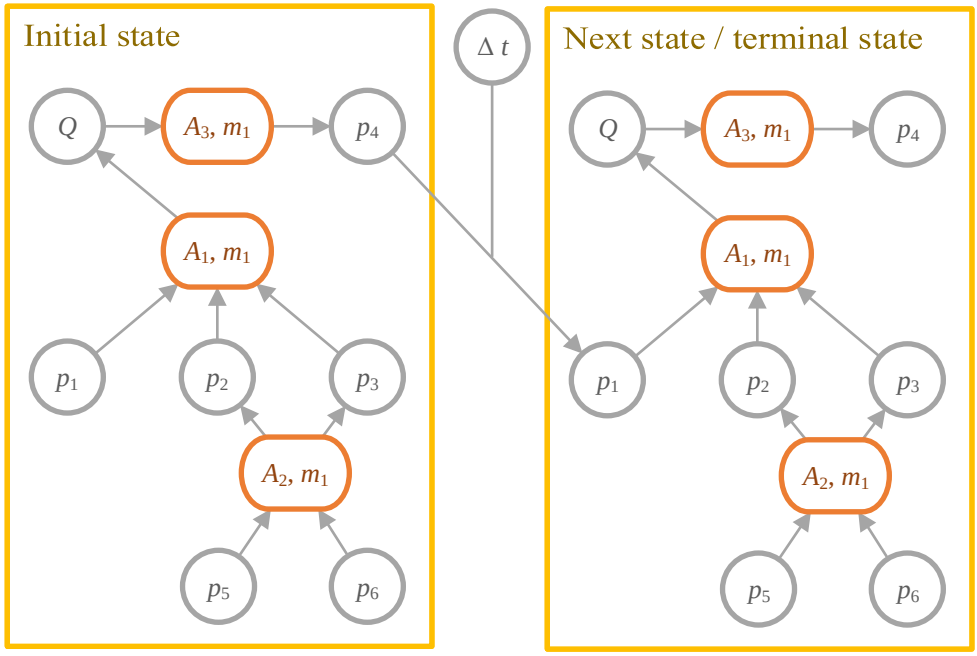

Fig. 7 A notional computation graph. The dependency graph from Figure 6 is broken down into state-independent parts, which are separated by the derivative between p_4 and p_1. The QoI Q is dependent on this transition, so a marching solution over the parameter t is required to arrive at a certain terminal state.

IV. Case study

A simplified aircraft model is shown in Figure 8, which represents the case study for the presented method. The arrows from flows to behaviors (and vice versa) have been included for clarity, although in the actual graph these are undirected, to prevent a false sense of causality. The Component *Aircraft* exhibits four behaviors: *Steady flight*, *Store*, *Combustion* and *Propulsion*. All parameters needed for computation of the fuel mass m_f at the end of cruise flight are present in Figure 8. The lift-to-drag ratio L/D, operational empty mass OEM and thrust-specific energy consumption $TSEC$ are assumed constant.

A set of four analyses are defined with the patterns shown in Figure 9. Clearly, these map easily to the system in Figure 8, although it should be emphasized the method does not use the labels shown in the figures. These are merely there for clarity. Furthermore, the *Store analysis* is very generic: it can be applied to any component storing any Flow, where both have a mass. Similarly, the *Combustion analysis* and *Propulsion analysis* are high-level descriptions of physical phenomena and can be applied to a wide variety of systems. Using the hierarchical structure described in

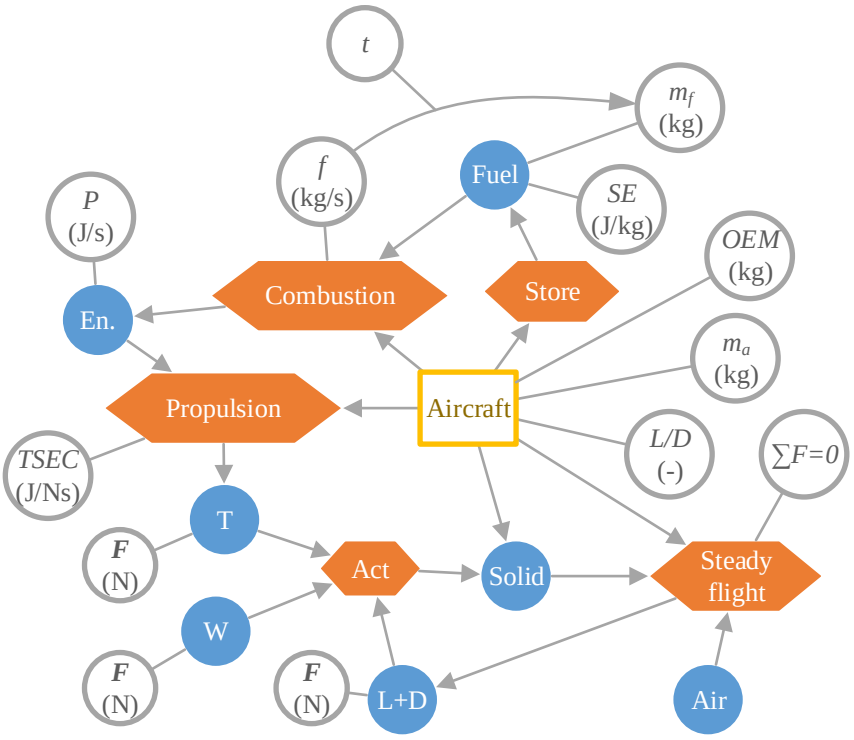

Fig. 8 A simplified aircraft model that allows computation of the fuel mass m_f throughout cruise flight.

section III these can be detailed when more specific representations are required, without losing generality as they still align with this high-level description.

Analysis A_5 (not shown in the figure) automatically computes the weight force on the solid from its mass. Each analysis is assumed to have only one mode, which compute one of the attributes as follows:

$$\text{Store analysis} \quad A_1 : m_c = m_f + OEM \tag{6}$$

$$\text{Combustion analysis} \quad A_2 : f = P/SE \tag{7}$$

$$\text{Propulsion analysis} \quad A_3 : P = TSEC \cdot T \tag{8}$$

$$\text{Steady flight analysis} \quad A_4 : T = W \cdot (L/D)^{-1} \tag{9}$$

$$\text{Gravity analysis} \quad A_5 : W = m_a \cdot g \tag{10}$$

From these modes and the maps of the analysis patterns to the system graph, the dependency graph is derived as shown in Figure 10. It shows a cycle where the QoI m_f depends on its derivative f (w.r.t. time t), which in turn depends on m_f itself. Hence it is deduced automatically a marching solution in t is required. Analogous to how the computation graph is derived in Figure 7, it is done here, which is not shown for brevity. Finally, the initial state s_0 and terminal state s_t have to be defined. The initial state should contain values for m_f, OEM, L/D, $TSEC$, SE and t. The terminal state only needs to contain a value for t. The QoI m_f is then computed in a pre-defined amount of intermediate states, and finally in the terminal state.

For those readers wondering why in Figure 8 the *Propulsion* and *Combustion* behaviors are specified separately, consider the aircraft is electric. Instead of fuel there is an electric energy store (i.e. a battery). Its energy content E_B is analogous to the fuel mass, and the attribute P specifying the power required for propulsion is its derivative w.r.t. time t. The dependency graph then becomes as shown in Figure 11, where it can be seen there still is a derivative present, but no cycle. Therefore, knowing the same values as for the fuel-driven aircraft in the initial state, only one iteration is required to compute the energy content E_B of the battery in the terminal state.

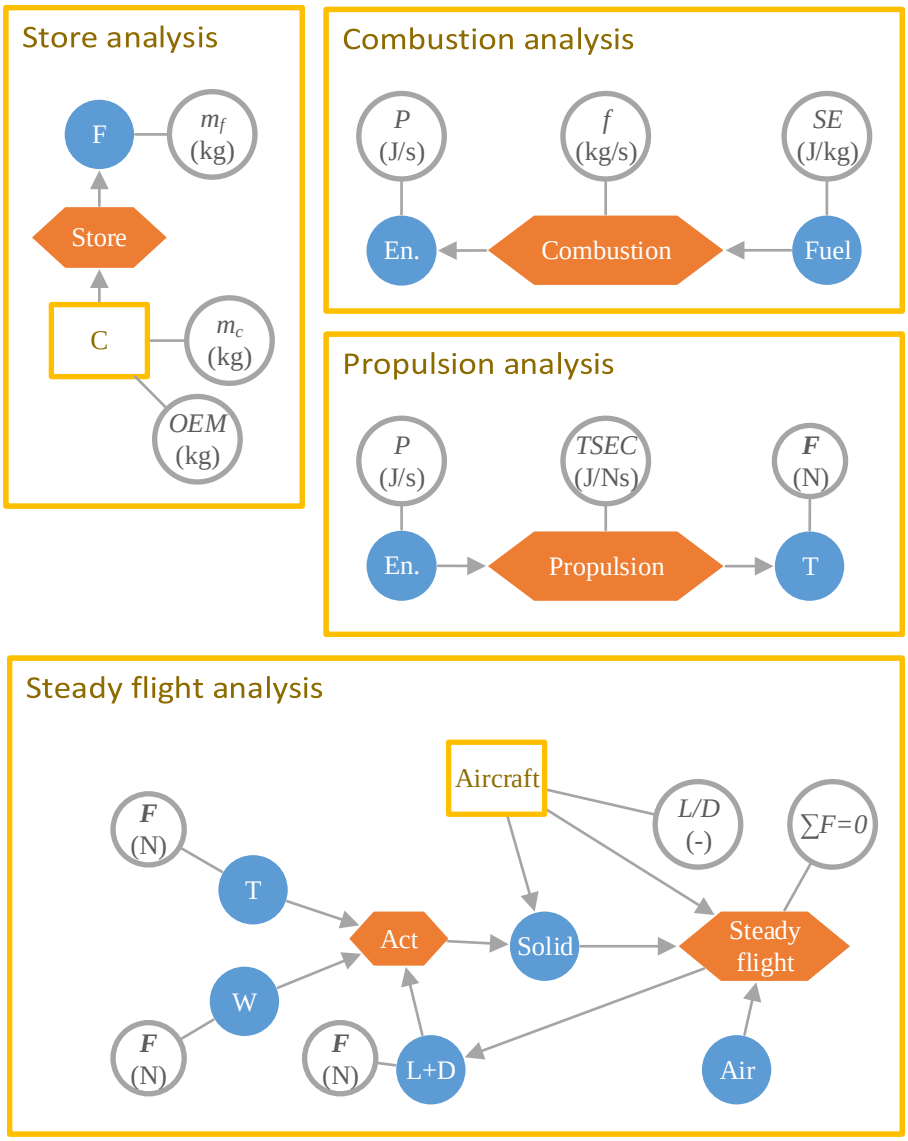

Fig. 9 Patterns for four analysis methods implemented in the case study.

V. Discussion

A formal language is constructed for describing engineering systems, and consequently technologies. The considerations put forth in the introduction motivated the decisions made herein. However, by no means does this imply the above method is the only, nor necessarily correct way to achieve the goals set out in the introduction. That is to say, the way humans think about physics and these systems is very different from how a computer likes to operate on information. To align the two requires a substantial amount of abstract thinking on the human's account. Even in the best attempt, it is likely assumptions and simplifications are made. In fact, the goal here is not to achieve a 100% accurate description of physics; such would be too complicated for conceptual design and hence technology selection. Additionally, different practitioners think differently about system descriptions. Although the current method aims to allow for that, while resulting in the same semantics, the method itself is also subject to different views, which is why the reader is encouraged to envision whether or not the claims made herein seem acceptable.

The attentive reader might have noticed the similarities of the presented approach with other modeling paradigms like Modelica, HOPSAN, SysML and PaceLab, which is expected to enable translation from either side to the other. As such, systems described in one language may be imported into the current approach and vice versa. As of now, such translations are not available. When available, these will improve integration of the current approach into existing

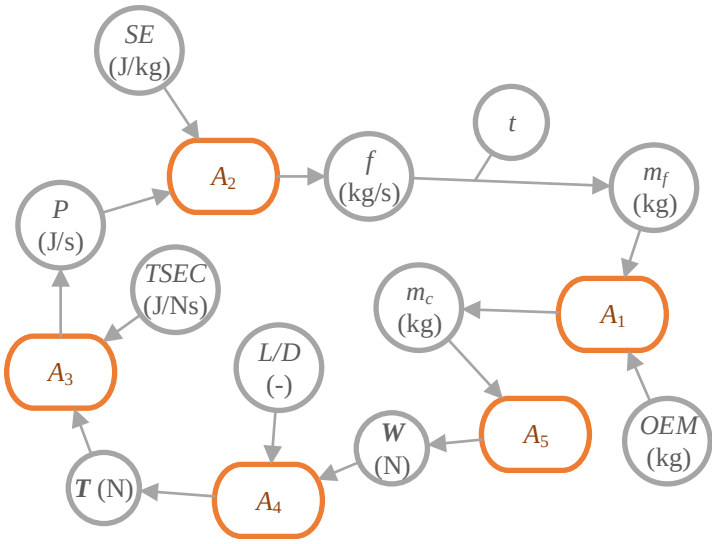

Fig. 10 Dependency graph derived for the case study. There is a cyclic dependency of m_f on itself, with a derivative allowing it to be temporally separated.

workflows. On the other hand, we believe our current approach allows for more flexibility in a few different aspects. Firstly, specifying analyses separately from a specific element or behavior allows them to be specified more generically, e.g. independent of the amount of input flows. Secondly, even if the behavior (as in the pattern of the analysis) is specified in more detail, it can be condensed into a more high-level description using the hierarchical structure defined above. Then, the analysis can be carried out as before (i.e. there is no strong coupling between granularity and analysis). Thirdly, different analysis can be mapping to the same part of the system, allowing different fidelity analyses to be applied to a certain element. Therefore, there is no strong coupling between system description and analysis fidelity. Finally, the approach enforces a graphical representation of the system and the available analyses, which may clarify to an analyst better what is computed and how.

Forbus, in his pivotal paper on Qualitative Process Theory (QPT), expresses the concept of what he dubs the Causal Directedness Hypothesis: *Changes in physical situations which are perceived as causal are due to our interpretation of them as corresponding either to direct changes caused by processes or propagation of those direct effects through functional dependencies* [19]. What he means by this is that humans tend to observe physical processes as causal, without these necessarily being so. The same problem was faced by the first author when attempting to apply the established functional bases [5] and structure-behavior-structure [20] models. These tend to work well for electrical and mechanical systems, where electricity moves from A to B, first through a resistor, then through a lamp. Or when modeling a hydraulic pump, with a certain hydraulic flow going in, and a pressurized one going out. On the other hand, if one tries to describe the Behavior of a wing moving through an airflow, this causal description tends to happen too, although it leads to a description that should not be: the wing affects the airflow, which creates a force on the wing. This force deforms the wing, which then alters the airflow again, and so on. While this is how we analyze such aero-elastic behavior, it should not be described as such, since the behavior itself is instantaneous and should be interpreted as such by a computer algorithm. The same case of describing a wing's behavior revealed a challenge with the common Flow breakdown (from [5]): it becomes very tedious, if not impossible to describe the exchange of energy between the wing and airflow. This is why the Effort basis in Table 3 is introduced. Even though these bases appear to be a good starting point, a more mathematical formulation may perhaps prove to be more useful. The work of Tonti [21] might provide a means on how to achieve this.

Forbus also states the following: *QPT concerns the structure of qualitative dynamics. We can view it as specifying a language in which certain commonsense physical models can be written. Can this language be extended to form a full language of behavior for physical systems? Although I have not yet done so, I will argue that the answer is yes, and that several advantages would result from the extension.* [19]. The same belief is held by the first author, and it is even deemed a necessity for a true unambiguous language to describe engineering systems and technologies. For example, think of how one should describe the Behavior *flying*. Some might say it is equivalent to moving through air, which can be more formally described as one Flow (a solid) being submerged in another Flow (air) and their velocity vectors

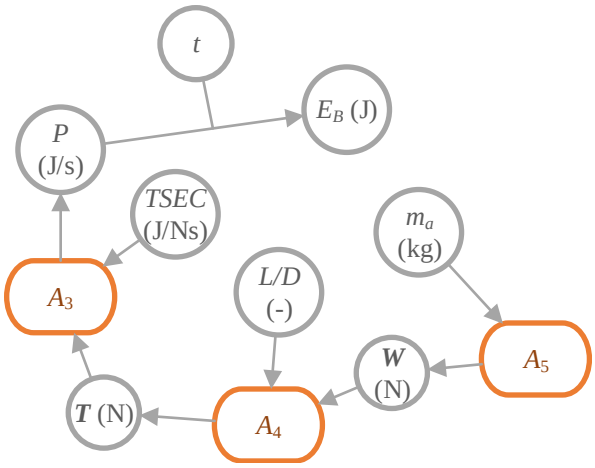

Fig. 11 Dependency graph in case the aircraft from Figure 8 is electric. Since the *Combustion* and *Store* behaviors have been omitted, analyses A_1 and A_2 are not required anymore and the dependency graph is acyclic.

being unequal. However, considering the flight of birds and aircraft, more is going on; the former definition also holds for falling through air, which is clearly not what a flying aircraft or bird are considered to be doing. Therefore, the ideal is to construct a language consisting of a finite set of atoms (analogous to an alphabet), which can be understood unambiguously, while allowing to express more high-level concepts according to one's own view. However, these higher-level concepts are a collection of these atoms, related in some way, and as such, even though two behaviors may have been called *flying*, a clear disambiguation can be made between the two.

Connecting analyses by means of variables required and computed has been done before, as mentioned in the introduction [16–18]. However, those approaches focus on multidisciplinary design optimization (MDO). It should be emphasized the current approach is not necessarily aimed at MDO, although it may in the future be used towards that end. On the contrary, the current approach does not allow analyses to form a cycle (e.g. analysis 1 computes y from x, while analysis 2 computes x from y), since it is presumed it cannot be guaranteed such a cycle will converge. Logically, this is a restriction that can easily be removed (it would actually simplify the algorithm, since cycle detection and removal is computationally expensive). The only case where a cycle is allowed is when there is a derivative relation in it, allowing it to be broken and becomes a linear computation in the states. The ability of the current method to "understand" derivatives is unique to the knowledge of the authors.

VI. Conclusion

In this work, a method is presented that aims to provide a more structured, formal and unambiguous way to model engineering systems and their behavior. Concretely, an ontology consisting of components, behaviors, flows, attributes, states, technologies and the various relationships between these constructs is developed. The entire ontology enables systems to be described in graphs, where nodes are the constructs and the edges the relationships between them. An analysis can be mapped to this graph using pattern matching. A dependency graph is deduced which relates the attributes with the analyses, from which a computation graph is constructed, which shows the order of computation. It is automatically derived when a marching solution is required, or a direct solution is possible. Finally, the computation for a specified Quantity of Interest is carried out automatically given enough information in the initial state.

In future work, algorithms will be developed that automatically generate the technology compatibility matrix, find relationships between technologies, and evaluate technologies, search for the most promising technology portfolios and estimate uncertainty of the impact metrics and input variables. The final aim is to arrive at a structured workflow, that gives a designer / engineer insight into the technologies and their effects and uncertainty, supporting the technology selection decision-making process.

Acknowledgements

This research is sponsored by the European Commission under the CleanSky II research program as part of project MANTA with grant agreement number 724558. The authors would furthermore like to thank Kristian Amadori, Christopher Jounannet and Erik Bäckström from Saab for valuable discussions and their feedback. Likewise, the insights from Ingo Staack at Linköping University were much appreciated. Finally, thanks to Jelle Boersma for critical reflections and meaningful discussions.

References

[1] Roelofs, M., and Vos, R., "Technology Evaluation and Uncertainty-Based Design Optimization: A Review," *2018 AIAA Aerospace Sciences Meeting*, AIAA, Kissimmee, Florida, 2018. doi:10.2514/6.2018-2029.

[2] Lu, Z., Yang, E.-S., DeLaurentis, D., and Mavris, D., "Formulation and test of an object-oriented approach to aircraft sizing," *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, AIAA, Albany, 2004, pp. 1–14. doi: 10.2514/6.2004-4302.

[3] Amadori, K., Bäckström, E., and Jouannet, C., "Selection of Future Technologies during Aircraft Conceptual Design," *55th AIAA Aerospace Sciences Meeting*, AIAA, 2017, pp. 1–11. doi:10.2514/6.2017-0233.

[4] Amadori, K., Bäckström, E., and Jouannet, C., "Future Technologies Prioritization for Aircraft Conceptual Design," *2018 AIAA Aerospace Sciences Meeting*, AIAA, Kissimmee, Florida, 2018. doi:10.2514/6.2018-1746.

[5] Hirtz, J., Stone, R. B., Mcadams, D. A., Szykman, S., and Wood, K. L., "A functional basis for engineering design : Reconciling and evolving previous efforts," *Research in Engineering Design*, Vol. 13, 2002, pp. 65–82. doi:10.1007/s00163-001-0008-3.

[6] Judt, D. M., and Lawson, C., "Development of an Automated Aircraft Subsystem Architecture Generation and Analysis Tool," *Engineering Computations*, Vol. 33, No. 5, 2016, pp. 1327–1352. doi:10.1108/EC-02-2014-0033.

[7] Yuan, L., Liu, Y., Sun, Z., Cao, Y., and Qamar, A., "A hybrid approach for the automation of functional decomposition in conceptual design," *Journal of Engineering Design*, Vol. 27, No. 4-6, 2016, pp. 333–360. doi:10.1080/09544828.2016.1146237.

[8] Sen, C., Summers, J. D., and Mocko, G. M., "A protocol to formalise function verbs to support conservation-based model checking," *Journal of Engineering Design*, Vol. 22, No. 11-12, 2011, pp. 765–788. doi:10.1080/09544828.2011.603295.

[9] Sen, C., Summers, J. D., and Mocko, G. M., "Physics-Based Reasoning in Conceptual Design Using a Formal Representation of Function Structure Graphs," *Journal of Computing and Information Science in Engineering*, Vol. 13, 2013, pp. 1–12. doi:10.1115/1.4023488.

[10] Wilschut, T., Etman, L., Rooda, J., and A. Vogel, J., "Generation of a function-component-parameter multi-domain matrix from structured textual function specifications," *Research in Engineering Design*, February 2018, pp. 1–16. doi: 10.1007/s00163-018-0284-9.

[11] Judt, D. M., and Lawson, C. P., "Application of an automated aircraft architecture generation and analysis tool to unmanned aerial vehicle subsystem design," *Journal of Aerospace Engineering*, Vol. 229, No. 9, 2015, pp. 1690–1708. doi:10.1177/ 0954410014558691.

[12] Guenov, M. D., Molina-Cristobal, A., Voloshin, V., Riaz, A., and Van Heerden, A. S. J., "Aircraft Systems Architecting – a Functional - Logical Domain Perspective," *16th AIAA Aviation Technology, Integration, and Operations Conference*, AIAA, Washington, 2016. doi:10.2514/6.2016-3143.

[13] Castet, J., Rozek, M., Ingham, M., Rouquette, N., and Chung, S., "Ontology and Modeling Patterns for State-Based Behavior Representation," *2018 AIAA Aerospace Sciences Meeting*, AIAA, Kissimmee, Florida, 2015. doi:10.2514/6.2015-1115.

[14] Kaderka, J., Rozek, M., Arballo, J., Wagner, D., and Ingham, M., "The Behavior, Constraint and Scenario (BeCoS) Tool: A Web-Based Software Application for Modeling Behaviors and Scenarios," *2018 AIAA Aerospace Sciences Meeting*, AIAA, Kissimmee, Florida, 2018. doi:10.2514/6.2018-1216.

[15] Ramakers, M. A. Y., "Accelerating Aircraft Design Using Automated Process Generation," M.sc. thesis, Delft University of Technology, October 2015.

[16] Van Gent, I., Ciampa, P. D., Aigner, B., Jepsen, J., La Rocca, G., and Schut, E. J., "Knowledge Architecture Supporting Collaborative MDO in the AGILE Paradigm," *18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Denver, Colorado, 2017. doi:10.2514/6.2017-4139.

[17] Van Gent, I., Lombardi, R., La Rocca, G., and d'Ippolito, R., "A Fully Automated Chain from MDAO Problem Formulation to Workflow Execution," *EUROGEN 2017*, Madrid, Spain, 2017.

[18] Pate, D. J., Gray, J., and German, B. J., "A Graph Theoretic Approach to Problem Formulation for Multidisciplinary Design Analysis and Optimization," *Structural Multidisciplinary Optimization*, Vol. 49, No. 5, 2014, pp. 743–760. doi: 10.1007/s00158-013-1006-6.

[19] Forbus, K. D., "Qualitative Process Theory," *Artificial Intelligence*, Vol. 24, 1984, pp. 85–168.

[20] Umeda, Y., Tomiyama, T., and Yoshikawa, H., "FBS modeling: modeling scheme of function for conceptual design," *Proceedings of the 9th international workshop on qualitative reasoning*, 1995, pp. 271–8.

[21] Tonti, E., *The Mathematical Structure of Classical and Relativistic Physics*, 1st ed., Birkhäuser Basel, 2013.