16th AIAA Aviation Technology, Integration, and Operations Conference,
AIAA AVIATION Forum, 13-17 June 2016, Washington, D.C.. (AIAA 2016-3143)
DOI: 10.2514/6.2016-3143

Aircraft Systems Architecting – a Functional-Logical Domain Perspective

Marin D. Guenov[*] Arturo Molina-Cristóbal[†] Vitaly Voloshin[†] Atif Riaz[†] Albert S.J. van Heerden[‡]
School of Aerospace, Transport, and Manufacturing
Cranfield University, Cranfield, Bedfordshire, MK43 0AL, United Kingdom

Sanjiv Sharma [§]
Airbus Operations Ltd, Filton, Bristol, BS34 7PA, United Kingdom

Claude Cuiller[**]
Airbus S.A.S., 1 Rond-Point Maurice Bellonte 31707 Blagnac Cedex, France

and

Tim Giese[††]
Airbus Operations GmbH, Hamburg, 43527, Germany

Presented is a novel framework for early systems architecture design. The framework defines data structures and algorithms that enable the systems architect to operate interactively and simultaneously in both the functional and logical domains. A prototype software tool, called AirCADia Architect, was implemented, which allowed the framework to be evaluated by practicing aircraft systems architects. The evaluation confirmed that, on the whole, the approach enables the architects to effectively express their creative ideas when synthesizing new architectures while still retaining control over the process.

Nomenclature and notation

TOICA	Thermal Overall Integrated Concept Aircraft (EU FP7 project)
RFLP	Requirements engineering, Functional, Logical and Physical design
AD	Axiomatic Design
FR	Functional Requirement
DP	Design Parameter
TRIZ	Theory of Inventive Problem Solving
SIT	Systematic Inventive Thinking
DSM	Design Structure Matrix
DM	Design Matrix
AVS	Avionics Ventilation System
ECS	Environment Control System
φ_i	ith function; i=1,…,n
μ_j	jth means ('means', 'component' and 'solution' are used interchangeably in the text); j= 1,...,m
p_{kj}	kth port which belongs to jth component; k=1,… ,l

[*] Professor, Head of the Centre for Aeronautics, Cranfield, MK43 0AL, United Kingdom, AIAA Senior Member.
[†] Research Fellow, Centre for Aeronautics, Cranfield, MK43 0AL, United Kingdom.
[‡] Research Student, Centre for Aeronautics, Cranfield, MK43 0AL, United Kingdom, AIAA Student Member.
[§] Expert for modelling and simulation methods and tools, Modelling and Simulation Projects, 14J3, Barnwell house, Filton, AIAA Member.
[**] Engineer, Architecture and Integration, Airbus SAS.
[††] Air Systems Simulation Expert, Airbus GmbH.

Published by AIAA. This is the Author Accepted Manuscript issued with: Creative Commons Attribution Non-Commercial License (CC:BY:NC 4.0).
The final published version (version of record) is available online at DOI:10.2514/6.2016-3143. Please refer to any applicable publisher terms of use.

$\mu_i \rightarrowtail \{ p_{1i}, p_{2i}, ..., p_{li}\}$	ports $p_{1i}, p_{2i}, ..., p_{li}$ which belong to component μ_i
$\varphi_i \rightarrowtail \{\varphi_{i1} \wedge \varphi_{i2} \wedge ... \wedge \varphi_{ik}\}$	Hierarchical breakdown of a function φ_i into k subfunctions $\varphi_{i1}, \varphi_{i2}, ..., \varphi_{ik}$
$\mu_i \rightarrowtail \{\mu_{i1} \wedge \mu_{i2} \wedge ... \wedge \mu_{ik}\}$	Hierarchical breakdown of a solution μ_i into k subcomponents $\mu_{i1}, \mu_{i2}, ..., \mu_{ik}$
$\mu_y \twoheadrightarrow \varphi_x$	Function φ_x is derived from solution μ_y.
$\varphi_x \multimap \mu_y$; $\mu_y \multimap \varphi_x$	Function φ_x is satisfied by solution μ_y.
$\mu_y \multimap \{\varphi_x \wedge \varphi_z\}$	Solution μ_y satisfies functions φ_x and φ_z
$\varphi_x \multimap \{\mu_y \wedge \mu_z\}$	Function φ_x is satisfied by solutions μ_y and μ_z
$\varphi_x \multimap p_{yz}$; $p_{yz} \multimap \varphi_x$	Function φ_x is mapped to port p_{yz}
$p_{xy} \longrightarrow p_{zq}$	Flow link from port p_{xy} to port p_{zq}

I. Introduction

THE ability to innovate is considered one of the key factors for success in the globally competitive world of today. This is particularly relevant in the design of complex products, such as modern aerospace vehicles, where the requisite systems account for up to one-third of the total empty weight.[1] Innovation in aircraft systems design can bring forth significant competitive advantages, including improved fuel consumption, reduced maintenance costs, and higher reliability.[2] The work reported in this paper is related to innovation in systems architecting and originates from a topical European research project, "Thermal Overall Integrated Conception of Aircraft" (TOICA[3,4]). Specifically, the aim of the work described herein has been to contribute to one of the research objectives of this project: the development of methods and tools enabling systems architects to discover, define, assess, and evaluate (systems) architectures. Within this context, the scope of the work is restricted to the process of conceptual systems architecting, which takes place in the functional and logical domains, as part of the RFLP (Requirements, Functional, Logical and Physical domains) process of systematic product development (see, for example, Ref. 5).

The paper is organized as follows: the next section outlines the basic concepts, state of the art tools, and methods related to the presented work. The proposed components of the framework, underpinning the functional-logical decomposition, are described in Section III. The evaluation of the proposed framework, implemented in a prototype software tool, is presented in Section IV. Finally, conclusions are drawn and future work is outlined in Section V.

II. Background

We begin this section with an outline of a case study specified within the framework of the TOICA project. It is used to define some of the terminology employed in the paper. After this, we summarize the relevant state of the art.

A. Case Study

Suppose an aircraft (ventilation) systems architect is considering how the avionics equipment should be cooled. At this level of reasoning, he/she identifies two possible solutions, liquid or air. The choice of air ventilation implies that either an existing air ventilation system will be used, or, alternatively, it has to be specified from initial requirement. Assuming the latter, the architect needs to decompose the high-level functional requirement 'cool avionics equipment' into lower level functions, such as "source air", "force airflow" and "evacuate hot air" (Figure 1). The air can be sourced either from the atmosphere or from the environmental control system. Assuming the former, the air needs to be dehumidified and any solid particles removed. It can be observed that the sourced air goes through a number of transformations, implying a (process) flow which can be inferred in one step (before even specifying equipment) or in a "zig-zagging" fashion, as shown in Figure 1.

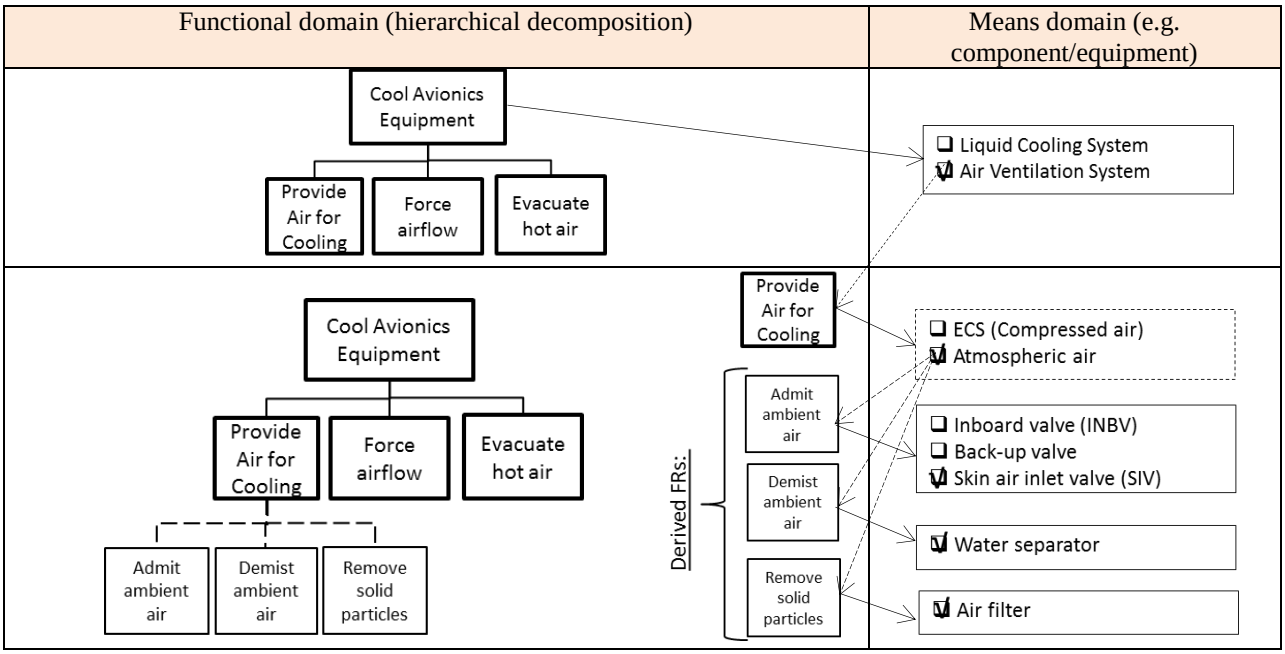


Figure 1: Functional decomposition of an air ventilation system.

Similar reasoning can be applied should the architect have chosen liquid cooling for parts of the avionics equipment (**Figure 2**). In this case, the architect may utilize the accumulator as storage for the liquid coolant. This appears to be a secondary function (the primary being to maintain constant pressure in the circuit). This, in turn, means that the accumulator becomes a multifunctional component. Also, it should be noted that the choice of a cross-heat exchanger brings an additional, or *derived*, requirement for the coolant, which would not necessarily have been inferred at the outset.

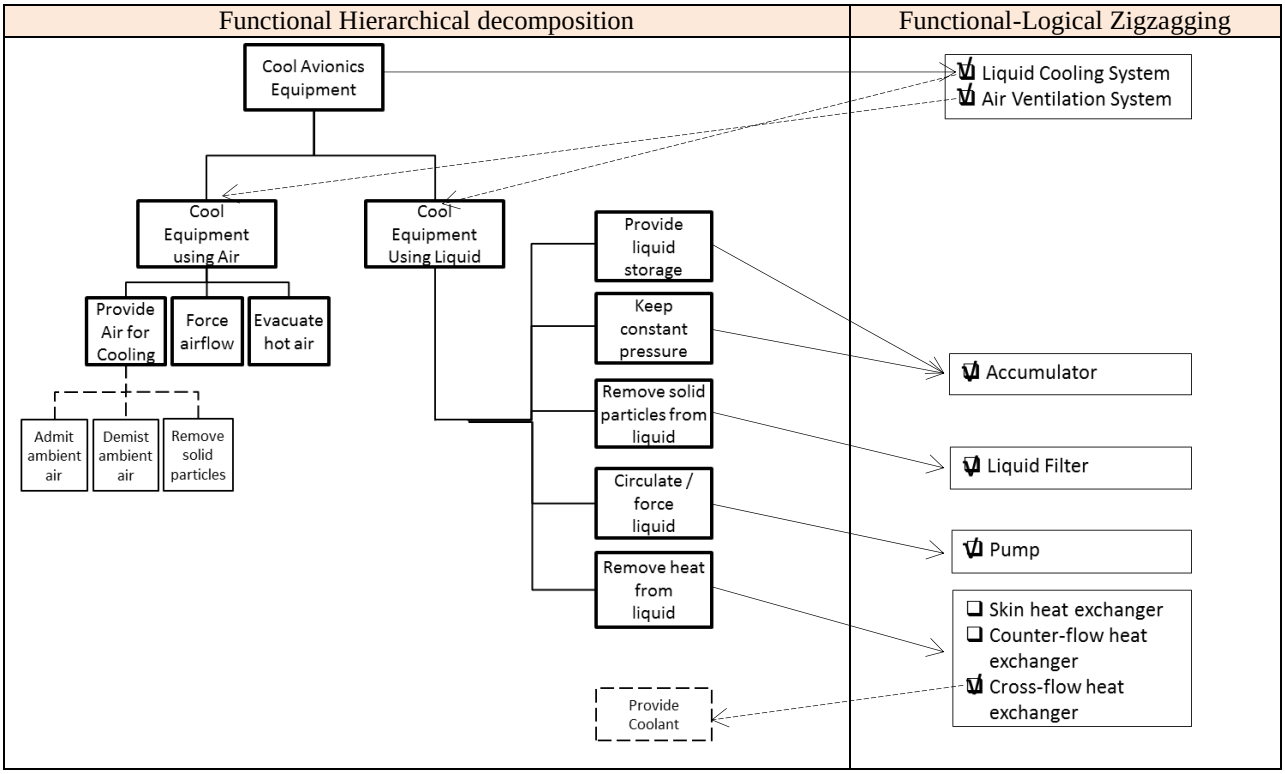


Figure 2: Functional decomposition of the liquid cooling system.

It can be observed, on the whole, that the architect has to deal with the decomposition of (functional) requirements, the identification of possible functional flows (which implies a process, or sequence) and the mapping of these functions and flows to components or subsystems.

B. State of the Art

As stated in the introduction, it is assumed that the systems architecting process takes place in four domains: Requirements, Functional, Logical and Physical – referred to as 'RFLP'. Specifically, RFLP stands for Requirements engineering, Functional design, Logical design, and Physical design and describes the process of systematic product development, from system analysis to system development, and comprises the descending branch of the well-known 'V' (vee) model. RFLP is based on the German guideline VDI 2206, "Design methodology for mechatronic systems",[6] quoted also in Ref 5. RFLP has gained popularity – not least because of its adoption by leading product lifecycle management (PLM) vendors. It appears that, in its current application, RFLP is unidirectional. However, in practice, designers/architects have to operate interactively in all domains, for example, when dealing with "derived" requirements. The latter, as described in systems engineering standards, such as ANSI/EIA-632,[7] may "result from a design decision for a logical or physical solution representation".

One design methodology, which, amongst other systems engineering concepts, relates to the definition of derived requirements, is Axiomatic Design (AD). The underlying hypothesis of AD[8,9] is that there exist fundamental principles that govern good design practice. The main distinguishable components of AD are: domains, hierarchies, and design axioms. The foundation axiom (known as the *Independence Axiom* or *Axiom 1*) is stated as: "*Maintain the independence of the functional requirements"*[9]. According to AD, the design process takes place in four domains: Customer, Functional, Physical, and Process. Through a series of iterations, the design process converts customer needs (CNs) into Functional Requirements (FRs) and constraints (Cs), which, in turn, are embodied into Design Parameters (DPs). DPs determine (but can also be affected by) the manufacturing or Process Variables (PVs). The process starts with the decomposition of the top level system requirement. Before decomposing an FR at a particular hierarchical level in the functional domain, the corresponding DP must be determined for the same hierarchical level in the physical domain. This iterative process is called 'zigzagging' (see also Ref. 10) and reflects, to an extent, the concept of 'derived requirements'. Zigzagging also involves the other domains, since manufacturing/production considerations may constrain certain design decisions.

At each level of the design hierarchy, the relations (dependencies) between the FRs and the DPs can be represented in an equation of the form:

$$\boldsymbol{FR} = [A]\boldsymbol{DP},$$

where each element of the design matrix [A] can be expressed as $A_{ij} = \partial FR_i/\partial DP_j$ (i = 1,…, m and j = 1,…, n). Eq. (1) is called the "design equation" and can be interpreted as "choosing the right set of DPs to satisfy given FRs".[7] Each element, A_{ij}, is represented as a partial derivative to indicate the dependency, or sensitivity, of a particular FR_i with regard to a particular DP_j. For simplicity, the value of an element, A_{ij}, can be expressed as '0' (i.e. the functional requirement does not depend on the particular design parameter), or, otherwise, 'X'. Depending on the type of resulting design matrix [A], three types of designs exist: uncoupled, decoupled, and coupled.

Uncoupled design occurs when each FR is satisfied by exactly one DP. The resulting matrix is diagonal and the design equation has an exact solution, i.e., Axiom 1 is satisfied. When the design matrix is lower triangular, the resulting design is decoupled, which means that a sequence exists which, by adjusting DPs in a certain order, will lead to the satisfaction of the FRs. The design matrix of a coupled design contains mostly non-zero elements and the FRs can therefore not be satisfied independently (i.e., this violates the Independence axiom). A coupled design can be decoupled, for example, by adding components to carry out specific functions. One additional factor that affects coupling is the number of FRs (m) relative to the number of DPs (n). If m>n, the design is either coupled, or the FRs cannot be satisfied. If m<n, the design is redundant. Note that in both cases the design matrix is not square. Particularly relevant to this research are Corollary 3 and Theorem 5, which originate from the first axiom:

- Corollary 3 states: "Integrate design features in a single physical part if the functional requirements (FRs) can be independently satisfied in the proposed solution."[8]
- Theorem 5 states: "When a given set of FRs is changed by the addition of a new FR, by substituting one of the FRs with a new one, or by selection of a completely different set of FRs, the design solution given by the original DPs cannot satisfy the new set of FRs. Consequently a new design solution must be sought." [8]

While AD has been one of the pioneering approaches to systems engineering and has given inspiration to the work presented here and in the recent past[11], it appears to have some limitations with regard to conceptual architectural design. For example, matching functions to means at the conceptual stage involves the selection of components, where more than one parameter (per component) could affect the function. Thus, one cannot guarantee that the design matrices are square. On the other hand, modern control systems have evolved to cope with coupled designs. Examples include the flight controls of modern aircraft, which are inherently coupled. In general, the use of multifunctional components in highly integrated mechatronic systems is justified when considering the necessary tradeoffs between performance, weight and other functionalities, and requires the application of advanced multidisciplinary (model–based) analysis and optimization tools.

The other driver behind the work presented here, is innovation/creativity in systems architecting. Innovative problem solving has been a research topic in its own right for a number of decades. Two of the most popular approaches, with particular relevance to engineering, are TRIZ[12,13] and its more recent streamlined version, Systematic Inventive Thinking ('SIT').[14] TRIZ and SIT are based on the observation that inventive solutions share common patterns. In SIT, one of these is the 'Closed World' condition, which states that, in the development of a novel solution, one must utilize only elements already present in the existing solution, or in its immediate environment. This condition forces the designer to rely on existing resources, e.g., on an existing baseline solution. Since design freedom is intentionally limited, the designer/architect ought to reconsider the relations between existing components, including their arrangement in space and time, their assigned functions, and their necessity. Thus, the expectation is that the problem solver would arrive at solutions which are both innovative and simple. The Closed World condition can be seen as creative thinking "within the box", or, in more colloquial terms, as "necessity is the mother of invention". Note however, that, in the SIT method, if a solution cannot be found in the Closed World it is permitted to search for an alternative beyond the boundaries of the Closed World – that is, "outside of the box".

Numerous sources seem to agree that the Closed World condition is one of the most important characteristics of an innovative/creative solution. For example, Chakrabarti[15] refers to the Closed World condition as 'resource-effectiveness'. According to him, resource-effectiveness is one of three underlying characteristics of a creative solution – the other two being 'novelty' (i.e. the "degree of difference from other existing ideas"[15]) and 'purposefulness' (i.e. the degree to which the solution solves the problem or satisfies the task – the higher, the better). In other words, the more resource-effective and purposeful a solution is, the more innovative it can be deemed to be.

Innovation, as a prime consideration, can be traced also in the aforementioned standards for the engineering of systems, such as EIA 632[7] and ISO/IEC 15288[16]. These support a seamless process of converting customer needs into systems/technical requirements, which are subsequently transformed into logical representations and, finally, into physical solution representations. The standards prescribe that (functional) requirements are developed in a *solution neutral environment* to allow the exploration of different solutions (physical embodiments). Indeed, the prevailing opinion in the engineering design field is still that form-follows-function. That is, the functional decomposition is followed by a logical and physical product decomposition (mapping). A review of recent cognitive and organizational sciences literature related to creativity in design (e.g. Refs.17-19) suggests that, in many cases, the opposite is true. For example, experiments, as reported in Ref. 19, indicate that solutions provided under the more structured function-follows-form condition would be judged more original and creative than those provided under the less structured form-follows-function condition and also that solutions provided by intuitive participants (in the experiment) would be judged more original and creative than those provided by systematic participants.

Additionally, Vermaas[20] has shown, from a philosophical point of view, that the relation between technical functions and their sub functions in (abstract) functional descriptions of technical products cannot be analyzed as a formal relation of parthood. That is, operating solely with (abstract) functional decompositions may lead to the paradox of a function containing an instance of itself. This appears to be an additional argument for the interactive co-evolution of the functional and logical domains in practice. Last, but not least, there is a view in the field of Psychology that "*a clear, unequivocal, and incontestable answer to the question of how creativity can be enhanced is not to be found in the psychological literature*" (Ref. 21, p. 407, cited in Ref. 22). This, in turn, suggests that, whatever mechanisms may be implemented in a computational design system that promotes innovation, a creative human should be at the center of its operation and control.

III. Framework Specification

The state of the art review and the interviews conducted by the authors with practicing aircraft systems architects, as part of the TOICA project, led to the specification of several requirements for an innovative systems

architecting framework. As stated in the introduction, the framework is aimed at the early stages of product development. It is intended to operate in all the RFLP domains, although the focus is here on the functional and logical domains. In particular, the framework has to be:

- Highly interactive, allowing work in either the functional or logical domain, while both domains are automatically updated.
- Flexible enough to encourage, rather than hinder, creativity and intuition by avoiding overly prescriptive processes.
- Accommodate both 'out of the box' thinking, as well as the (SIT) Closed World approach, including allocation and re-allocation of functions to components and vice versa.
- Able to handle derived requirements.
- Scalable.
- Able to handle conditional (time dependent) views.
- Exportable to sizing tools.

In this particular work, scalability, conditional (time dependent) views, and links to sizing tools were not covered, but are part of ongoing effort.

A. Architecture definition: elementary process mappings

As stated in the requirements, the aim here is not to prescribe a rigid requirements decomposition process, but a formalization of essential F-L mappings. The latter are intended to comprise a descriptive language, which captures the evolving architecture.

Functions and Means

A functional requirement (or 'function' for short), φ is the action that a product or system has to perform in order to meet the stakeholders' needs. The derivation of functional requirements from ill-structured customer needs is not a trivial problem, but is considered outside the scope of this work. Functions are expressed as a 'verb-noun' combination, φ(n), where 'φ' is the action and 'n' is the object of the action, e.g., dehumidify (air). For convenience, the subject 'n' will be omitted from now on, unless explicitly required.

A means, or a solution, μ is the physical entity or entities (e.g. part, component, subsystem, or even the whole system/product) that performs (fulfils) the required function. Defined below are a number of basic relationships between functions and means. The basic mappings between architectural elements employed in the proposed framework are presented below:

- **Function-to-means**. This is a mapping from a functional requirement to means. The following cases can occur:
  - A single function, φ_1, is satisfied by a single means, μ_1, which can be expressed as $\varphi_1 \circ\!\!- \mu_1$;
  - A single function is fulfilled by a number of (equivalent) means, $\varphi_1 \circ\!\!- \{\mu_{1.1} \wedge \mu_{1.2} \wedge \ldots \mu_{1.n}\}$, which is called redundancy;
  - A series of components $\mu_{1.1}, \mu_{1.2}, \ldots, \mu_{1.n}$ collectively satisfy a single function φ_1, which can be represented as $\varphi_1 \circ\!\!- \{\mu_{1.1} \wedge \mu_{1.2} \wedge \ldots \wedge \mu_{1.n}\}$. That is, function φ_1 will not be fulfilled if any of these components are missing. This mapping, together with the specification of multifunctional means (components) reflect the 'closed world' condition of TRIZ/SIT, as discussed above.

- **Means-to-function**. This is a mapping from a solution (e.g. a component) to a function. Unlike the Function to Means mapping, where only a function satisfaction relationship is possible, here two types of relationships are possible: function satisfaction and function derivation. Such relations can occur due to:
  - The emergence of a derived requirement. For example, choosing a bootstrap refrigeration system will require the air to be pressurized, which can be stated as $\mu_1 \twoheadrightarrow \varphi_{1.1}$;
  - Assigning additional function(s) to an existing means. That is, specifying multifunctional means (components), which can be expressed as $\mu_i \multimap \{\varphi_{j.1} \wedge \varphi_{j.2} \wedge \ldots, \varphi_{j.k}\}$.

- **Decomposition:**
  - Iterative function-means decomposition – this is akin to the "zig-zagging" process in axiomatic design[10], as shown in Figure 1 and Figure 2. This can be represented as $\varphi_1 \circ\!\!- \{\mu_1 \twoheadrightarrow [\varphi_{1.i} \ldots \circ\!\!- (\ldots \mu_{1.j})] \ldots\}$. This process comprises a series of function-to-means and means-to-function

mappings. It may also include partial single domain decompositions (see bullet point below and also the Case Study section above)

- (Invariant) functional decomposition – if a function, φ_1, can be decomposed into, say three independent sub-functions, $\varphi_{1.1}$, $\varphi_{1.2}$, and $\varphi_{1.3}$, which are not derived functions, it can be written as $\varphi_1 \prec \{\varphi_{1.1} \wedge \varphi_{1.2} \wedge \varphi_{1.3}\}$. Such decomposition can be representative of a 'functional flow', i.e. $\varphi_{1.1} \rightarrow \varphi_{1.2} \rightarrow \varphi_{1.3}$. There is a clear process sequence which does not need to be directly derived from the means. For example, the decision to use ambient air implies the need for functions, such as source (air), dehumidify (air), heat (air), etc., which will be part of a process, but the actual means are not yet specified.
- Leaf function node – a leaf node is a function at the lowest level in the functional hierarchy. A leaf node can be identified when further decomposition of this function would result in a change of the object of the function, n in $\varphi(n)$[10].

- **Aggregation:**

Aggregation is needed when (parts of) the functional or logical hierarchical structures are constructed from bottom-to-top. It can also be part of a zig-zagging process, or can be done independently in a single domain (functional or logical). Usually, this happens when either an alternative to a reference architecture is being developed, or when only the logical/physical description is available, and it is required to "reverse engineer" it to obtain the functional description of the architecture. The following concepts relate to the idea of aggregation:

- Component aggregation – is the process of combining components to comprise a single higher level component (i.e. an assembly of these components), i.e. $\mu_1 \rightarrowtail \{\mu_{1.1} \wedge \mu_{1.2} \wedge \ldots \wedge \mu_{1.n}\}$. This may be the case when an integrated solution (e.g. motor gear group) replaces previously individually-linked components.
- Functional aggregation – can be the consequence of means-aggregation in the logical domain.

All the constructs (mapping, decomposition, etc.) described above constitute a basic language, which should enable an architectural description to be synthesized in the functional and logical domains. By employing such a language to capture the actions of constructing the architecture enables the designer to document the final architecture, as well as the process of its creation. The language can therefore be used for the purposes of configuration management and rationale capture. Furthermore, it can enable the synthesized architectures to be parsed by other software tools.

B. Object Model

The functional-logical part of the architectural design framework, as outlined in the previous subsection, is implemented using an object-oriented approach. The proposed object model consists of several primary classes, including: 'Architecture', 'Function', 'Component', and 'Port'. A high level UML class diagram, describing the object model, is shown in Figure 3.

The 'Architecture' class contains all the information that describes the architecture as a whole, including the list of all functions and components.

The 'Function' class represents a function (functional requirement). It has attributes and methods enabling the (functional) decomposition process. A function has a reference to a component to implement the concept of function-means mapping, including derived requirement

The 'Component' class stands for physical solutions (means) satisfying the functional requirements. A component may also have attributes that represent internal component parameters (e.g. an electrical transformer can have its transformation ratio and efficiency coefficient as parameters). All external parameters (e.g. in case of the transformer, input/output voltage and current) will be managed by port variables, which are described below. Similar to the function class, 'Component' has attributes and methods enabling the function-means mapping and the decomposition process. Components always have a reference to a 'template'. A template (not shown in Figure 3) is a reference component in the component library (a pre-existing collection of model components, which can be edited by the architect). Reference components are represented by objects of "Component" class. The main advantage of maintaining the template relationships is that, should the architect decide to change something in a reference component (e.g. add an extra parameter), this will also be added to all the relevant components in the actual architecture. A Component has references to all functions derived from it.

The 'Port' class describes the interfaces of a component with other components and the environment. It describes the type of the flow allowable through the interface (e.g. air, heat, torque, etc.) and the directions of the flow, which

can be either 'input' or 'output'. The Port class may also contain a description of flow parameters (e.g. in the case of an electrical interface, 'voltage' and 'current'). Such parameters provide an additional level of compatibility control between components or solutions. Another important reason to introduce the port class is that it is intended to facilitate the export of the synthesized architectures to sizing and analysis tools (e.g. Modelica). This forms a part of our planned future work.

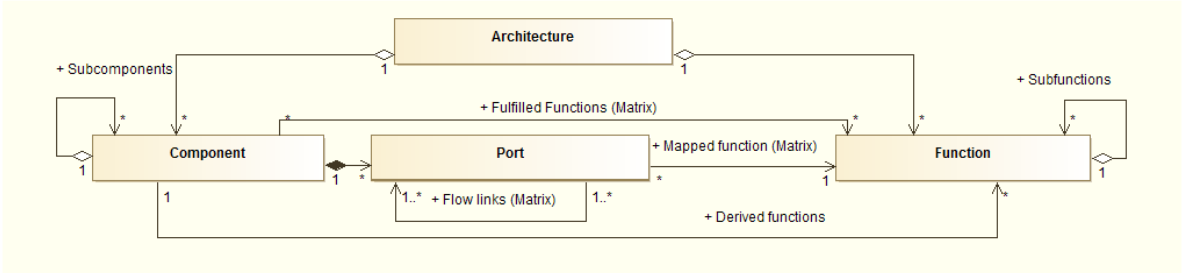


Figure 3: UML class diagram of the prosed framework (attributes and methods are not shown).

The classes outlined above allow the construction of the following data structures:

- In the Functional domain:
  - A 'Functional hierarchy' – a graph (tree) that contains the evolving functional decomposition of the architecture, including derived functions.
  - 'Functional flow graph' and associated 'functional flow matrix' – this is a graph and a square matrix, which contains the functional flow for the entire architecture, i.e. the flow connections between the functions
- In the Logical domain:
  - 'Component hierarchy' (tree) – this (optional) graph is aimed at composing the emerging product decomposition trees. It is similar, but not identical to the better-known 'bill of materials'.
  - 'Connectivity graph' and the associated 'connectivity matrix' – the connectivity graph shows the components/subsystems/systems of the solution and their logical/physical relations (flows of energy and matter). The connectivity matrix is another way of representing the information in the connectivity graph. It is a square matrix, and maintains the logical and physical connections between the components that embody the functions. It is also known as a DSM or 'Design Structure Matrix' (see, for example, Ref. 23).
- Cross Domain:
  - Design Matrix (DM) – this is a matrix that represents which components fulfil which functions. In particular, this matrix allows functions coupled through multifunctional components to be identified.

C. Functionality

The object model shown in Figure 3 is augmented with algorithms of which a few examples are presented in Table 1.

Table 1: Example algorithms for architecture synthesis in the functional and logical domains.

Functionality	Algorithm outline	Notes
Add function	$FL[\varphi_1,...,\varphi_n] \rightarrow FL[\varphi_1,...,\varphi_{n+1}]$; $F2PM[n,m] \rightarrow F2PM[n+1,m]$ $DM[n,m] \rightarrow DM[n+1,m]$	
Remove function	If $\varphi_i \rightarrowtail \{\varphi_{i1} \wedge \varphi_{i2} \wedge ... \wedge \varphi_{ik}\}$, then: For l=1 to k Remove_Function(φ_{il}) End if If $\mu_j \multimap \{\varphi_i\}$ OR ($p_{jp} \multimap \{\varphi_i\}$ AND ($\mu_j \twoheadrightarrow \{p_{jp}\}$ OR ($\mu_j \twoheadrightarrow \{p_{j1},...,p_{jp},...,p_{jq}\}$ AND $\forall l \neq p\ p_{jl} \multimap \{\}$))), then: Propose to remove solution μ_j	The removal algorithm works in a similar fashion (but in the reverse order) to the one for adding a function. However, the mapping to components and ports needs to be checked.

	End if F2PM[n,s]→F2PM[n-1,s] DM[n,m] →DM[n-1,m] FL[φ_1,...,φ_i,...,φ_n]→FL[φ_1,...,φ_{i-1}, φ_{i+1},...,φ_n];	
Add a child to a function	If $\varphi_i \multimap \{p_{kl}\}$, then: $\varphi_i \multimap \{\}$ $\varphi_j \multimap \{p_{kl}\}$ End if $\{ \varphi_i , \varphi_j \} \rightarrow \varphi_i \rightarrowtail \{\varphi_j\}$	Function φ_j becomes a child of the function φ_i. The parent reference of function φ_j is set to the object, representing function φ_i. The reference to the object, which represents function φ_j, is added to the list of children of function φ_i.
Link component to component	If $\sigma_i \twoheadrightarrow \{p_{ik}\}$ AND $\mu_j \twoheadrightarrow \{p_{jl}\}$, then: If $\mu_i = \mu_j$, then: Break End if If (p_{ik}.Type != p_{jl}.Type), then: Break End if If (p_{ik}.Direction = "input" AND p_{jl}.Direction = "input") OR (p_{ik}.Direction = "output" AND p_{jl}.Direction = "output"), then: Break End if If p_{ik}.Direction = "output", then: P2PL[p_{ik},p_{jl}] = p_{ik}.Type Else: P2PL[p_{jl},p_{ik}] = p_{ik}.Type End if If ($\varphi_m \multimap p_{ik}$ AND $\varphi_m \multimap p_{jl}$), OR ($\varphi_m \multimap p_{ik}$ AND $\varphi_m \multimap \mu_j$) OR ($\varphi_m \multimap \mu_i$ AND $\varphi_m \multimap p_{jl}$), then: F2PM[$\varphi_m$, p_{ik}] = false; F2PM[φ_m, p_{jl}] = false; End if End if	Two components, μ_i and μ_j, can be linked by linking their respective compatible ports. If there is no pair of compatible ports, they can be created using the "create a port" function (not shown in this paper).

IV. Evaluation

The evaluation of the proposed framework was conducted as follows: first, a prototype software tool, 'AirCADia Architect' was developed from the requirement specification and the object model outlined in Section III.B. Practicing architects and engineers from the industrial partners in the TOICA project were then trained on how to use AirCADia Architect. Following the training, the architects were asked to synthesize their own Avionics Ventilation System (AVS) architecture, by using the tool unsupervised. After this 'hands-on' session, the participants were asked to complete a semi-structured questionnaire, the aim being to obtain a qualitative assessment of the approach. The rest of this section provides more details about each part of the process.

A. AVS Architecture Synthesis in AirCADia Architect

AirCADia Architect was developed during the course of the TOICA project. Its development followed the wider requirements elicitation effort, which covered not only the functional and logical domains, but also the requirements and physical domains, including standardization. The tool was developed by making use of the object model described in Section III.B and a number of algorithms, examples of which were presented in Section III.C.

During the evaluation of the proposed approach and the prototype tool, the architects utilized both decomposition (top-down) and aggregation (bottom-up) reasoning approaches for architecture synthesis (described in Section III.A). Note that the actual architectural synthesis with the tool involves only the clicking, dragging, and dropping actions performed by the user. While following the top-down reasoning approach, the top-level invariant

function φ_1(cool avionics equipment) was decomposed into three invariant child functions $\varphi_{1.1}$ (source air), $\varphi_{1.2}$ (force airflow), and $\varphi_{1.3}$ (evacuate air), as shown in Eq. (1):

$$\varphi_1 \prec \{\varphi_{1.1} \wedge \varphi_{1.2} \wedge \varphi_{1.3}\} \tag{1}$$

Equation (1) and the rest of the equations in this section are listed for illustration of the basic architecture description language presented in Section III.A. In the prototype tool, the functions and means are represented with their actual names, as illustrated in Figure 4-Figure 8.

Next, solutions $\mu_{1.i}$ were mapped to all the child functions $\varphi_{1.i}, \forall\, i = 1, 2, 3$. Two solutions were identified for $\varphi_{1.1}$: outside atmospheric air during ground operation $\mu_{1.1}^{gnd}$, and the ECS pack air during cruise condition $\mu_{1.1}^{crz}$. Similarly, $\mu_{1.2}$ (electric fan) and $\mu_{1.3}$ (outboard flow valve) were selected to fulfil functions $\varphi_{1.2}$ and $\varphi_{1.3}$ respectively. The mapping of the child functions, $\varphi_{1.i}, \forall\, i = 1, 2, 3$, is listed in Eq. (2).

$$\begin{gathered} \varphi_{1.1} \multimap \{\mu_{1.1}^{gnd} \wedge \mu_{1.1}^{crz}\} \\ \varphi_{1.2} \multimap \mu_{1.2} \\ \varphi_{1.3} \multimap \mu_{1.3} \end{gathered} \tag{2}$$

After mapping solutions, $\mu_{1.i}$, to the child functions, $\varphi_{1.i}$, the architects identified and created the derived functions emerging from the chosen solutions $\mu_{1.i}$. For instance, the architects discovered that, if the solution $\mu_{1.1}^{\mathrm{gnd}}$(outside atmospheric air) is utilized for cooling, then three new derived functions $\varphi_{1.1.1}$ (admit ambient air) $\varphi_{1.1.2}$ (filter air) and $\varphi_{1.1.3}$ (demist air) are required, whereas the solution $\mu_{1.1}^{\mathrm{crz}}$ (ECS pack air) does not require these three derived functions. This functional-logical zig-zagging (described in Section III.A) for solution $\mu_{1.1}^{\mathrm{gnd}}$ is expressed in Eq. (3).

$$\varphi_{1.1} \multimap \mu_{1.1}^{gnd} \twoheadrightarrow \{\varphi_{1.1.1}, \varphi_{1.1.2}, \varphi_{1.1.3}\} \tag{3}$$

Since, the functions $\varphi_{1.1.1}$, $\varphi_{1.1.2}$ and $\varphi_{1.1.3}$ were derived from solution $\mu_{1.1}^{\mathrm{gnd}}$, solutions $\mu_{1.1.1}^{gnd}$ (skin inlet valve), $\mu_{1.1.2}^{gnd}$ (water separator) and $\mu_{1.1.3}^{gnd}$ (air filter) are relevant, as long as solution $\mu_{1.1}^{\mathrm{gnd}}$ is present. That is, if the architect decides to delete $\mu_{1.1}^{\mathrm{gnd}}$, then the derived functions $\varphi_{1.1.1}$, $\varphi_{1.1.2}$ and $\varphi_{1.1.3}$ (and their mapped solutions $\mu_{1.1.1}^{gnd}$, $\mu_{1.1.2}^{gnd}$, and $\mu_{1.1.3}^{gnd}$) will automatically be deleted with the help of the links maintained in the object model. The functional-logical zig-zagging is represented by Eq. (4).

$$\varphi_{1.1} \multimap \mu_{1.1}^{gnd} \twoheadrightarrow \{\varphi_{1.1.1}, \varphi_{1.1.2}, \varphi_{1.1.3}\} \multimap \{\mu_{1.1.1}^{gnd}, \mu_{1.1.2}^{gnd}, \mu_{1.1.3}^{gnd}\} \tag{4}$$

Figure 4 shows the functional hierarchical decomposition on the left, whereas the functional-logical zig-zagging for function $\varphi_{1.1}$ (when satisfied by the solution $\mu_{1.1}^{\mathrm{gnd}}$, i.e. $\varphi_{1.1} \multimap \mu_{1.1}^{\mathrm{gnd}}$) is shown on the right. The green color shows that the functions have been mapped to solutions in the logical domain, whereas the red color represents unfulfilled functions. It should be noted that the derived functions, $\varphi_{1.1.1}$, $\varphi_{1.1.2}$, and $\varphi_{1.1.3}$, are listed one level down of the function $\varphi_{1.1}$ in the functional hierarchical decomposition (Figure 4).

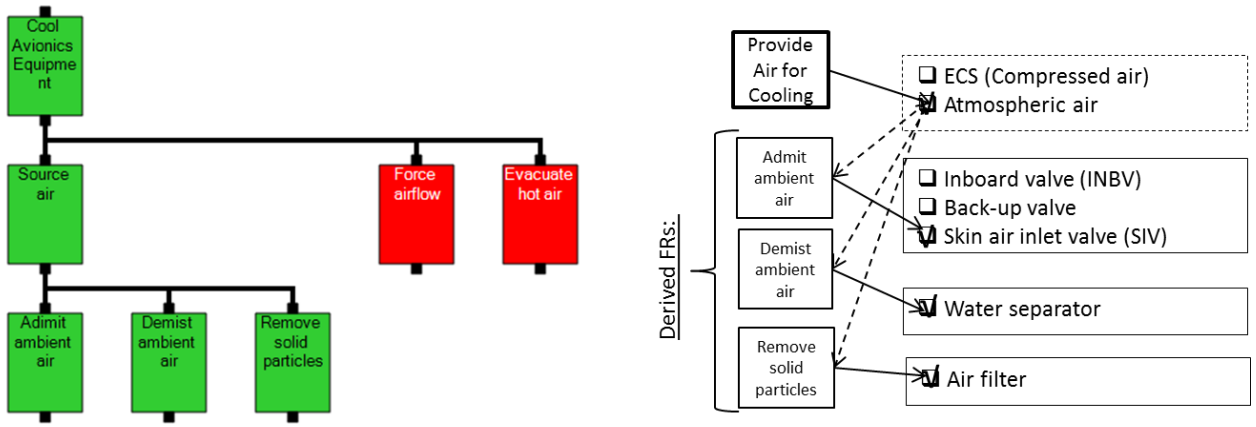


Figure 4: Functional decomposition (left) and functional-logical zig-zagging (right).

After assigning the solutions to $\varphi_{1.1}$, $\varphi_{1.1.1}$ $\varphi_{1.1.2}$ and $\varphi_{1.1.3}$, respectively, the architects continued to assign solutions to the other leaf functions ($\varphi_{1.2}$ and $\varphi_{1.3}$) in the functional hierarchy. However, in order to increase the reliability of the architecture, the architects decided to opt for two redundant solutions $\mu_{1.2}$ (two electric fans) for the function $\varphi_{1.2}$ (force airflow). The mapping of the function $\varphi_{1.2}$ with two redundant solutions ($\{\mu_{1.2} \wedge \mu_{1.2}\}$) is represented by Eq. (5):

$$\varphi_{1.2} \multimap \{\mu_{1.2} \wedge \mu_{1.2}\} \tag{5}$$

Alternatively, while synthesizing the architecture, the architect may choose to utilize a single solution (component) to fulfil more than one function, with the intent to save weight and increase performance efficiency. However, as pointed out in the introduction, multi-functional solutions (components) could increase the complexity (coupling) of the architecture.

Figure 5 shows the "redundant component" constructs (relations) and also the design matrix (DM) that manages these constructs in AirCADia Architect.

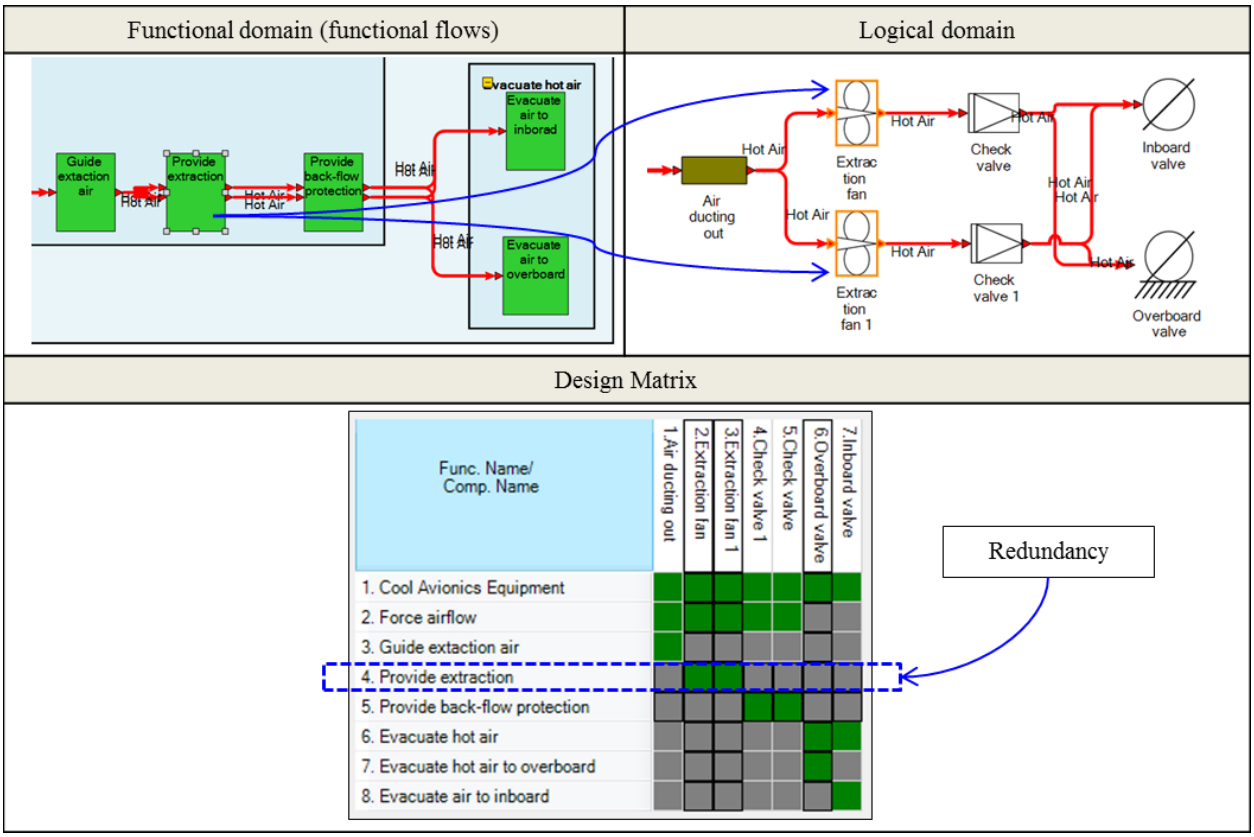


Figure 5: Component redundancy example.

Figure 6 shows the synthesized Avionics Ventilation System (AVS) baseline architecture as represented in AirCADia Architect.

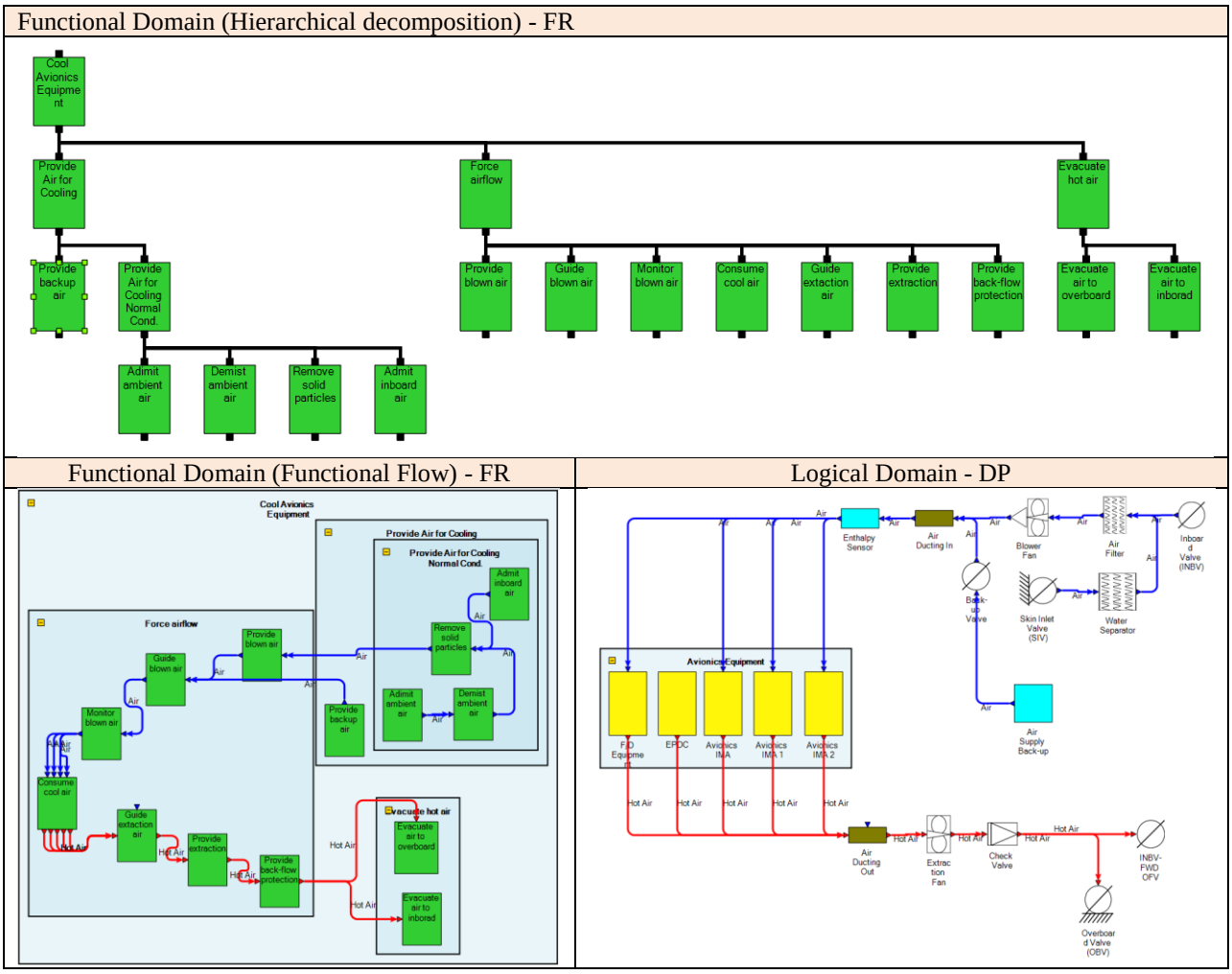


Figure 6: Functional and logical view of the baseline AVS architecture.

Discussed in the remaining part of this subsection is the capability of modifying existing architectures with the proposed framework. In order to meet the higher power dissipation requirements of the avionics IMA, the architects decided to investigate the use of a liquid-cooling technology. They modified and/or replaced the functions and components of the baseline architecture to synthesize a new (hybrid AVS) architecture. First, four new functions, $\varphi_{1.4}$ (provide liquid storage), $\varphi_{1.5}$ (remove solid particles), $\varphi_{1.6}$ (pressurize liquid), and $\varphi_{1.7}$ (guide liquid), were added to the functional hierarchy. Next, the solutions $\mu_{1.4}$ (accumulator), $\mu_{1.5}$ (liquid filter), $\mu_{1.6}$ (pump), and $\mu_{1.7}$ (ducting) were created in the logical domain and allocated to the newly created functions, $\varphi_{1.4}$, $\varphi_{1.5}$, $\varphi_{1.6}$ and $\varphi_{1.7}$, respectively, as shown in Eq. (6).

$$\begin{aligned} \varphi_{1.4} &\multimap \mu_{1.4} \\ \varphi_{1.5} &\multimap \mu_{1.5} \\ \varphi_{1.6} &\multimap \mu_{1.6} \\ \varphi_{1.7} &\multimap \mu_{1.7} \end{aligned} \tag{6}$$

Unlike the air-cooling system, no function was created to evacuate the liquid, since the liquid cooling is a closed-loop system. However, after creating new functions, it was identified that another function, $\varphi_{1.8}$, is required, namely "transfer heat to coolant". Hence, a new function was inserted in the functional hierarchy. Following this, the solution, $\mu_{1.8}$ (heat exchanger), was created in the logical domain and allocated to the function $\varphi_{1.8}$, as shown in Eq. (7).

$$\varphi_{1.8} \multimap \mu_{1.8} \tag{7}$$

After allocating $\mu_{1.8}$, a new derived function $\varphi_{1.8.1}$ (emerging from $\mu_{1.8}$), i.e. "provide cool air" was created in the functional domain, as shown below in Eq. (8).

$$\varphi_{1.8} \multimap \mu_{1.8} \twoheadrightarrow \varphi_{1.8.1} \tag{8}$$

In order to allocate a solution to the derived function "provide cool air" ($\varphi_{1.8.1}$), cool air from the air-cooling system was chosen initially. However, it was considered that the derived function $\varphi_{1.8.1}$ can also be fulfilled by the warm air extracted from the other avionics equipments, which are cooled by the air cooling system. This was seen as potentially innovative architecture with a higher expected efficiency. The functional-logical zig-zagging with regard to function $\varphi_{1.8}$ is expressed in Eq. (9) and shown in Figure 7(right).

$$\varphi_{1.8} \multimap \mu_{1.8} \twoheadrightarrow \varphi_{1.8.1} \multimap \mu_{1.8.1} \tag{9}$$

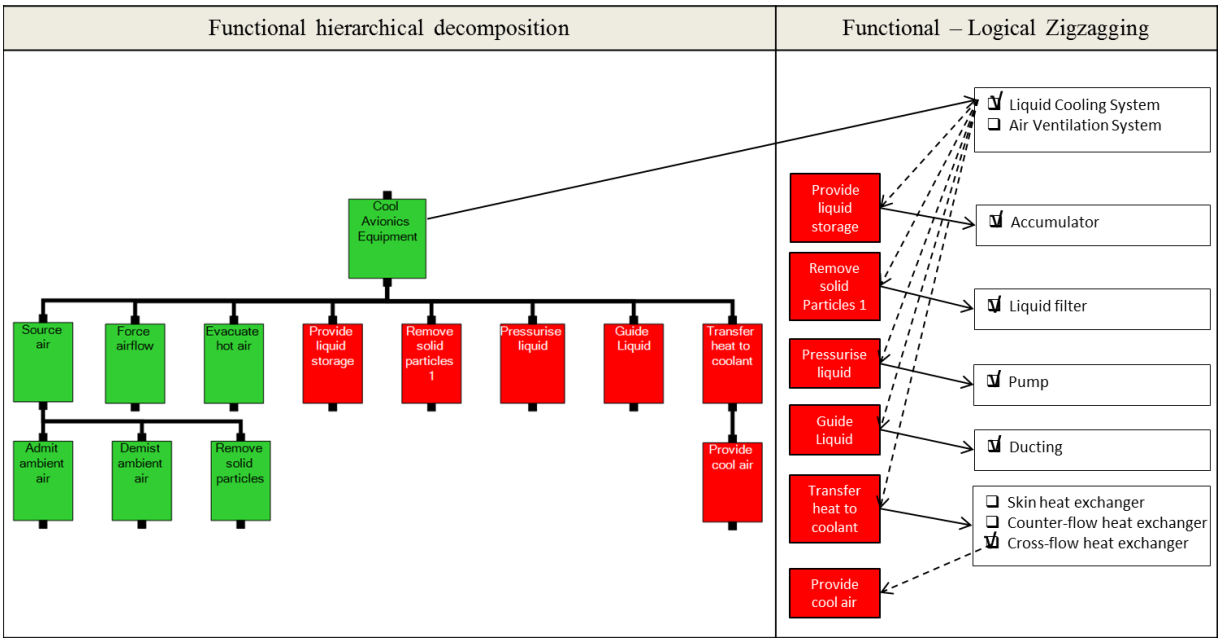


Figure 7: Zig-zagging for function "remove heat from liquid".

Figure 8 shows the completed functional and logical flow views of the hybrid AVS architecture. The blue dashed rectangles in the functional and logical flow view enclose the additional and modified functions and components, compared with the baseline (air-cooled) AVS architecture.

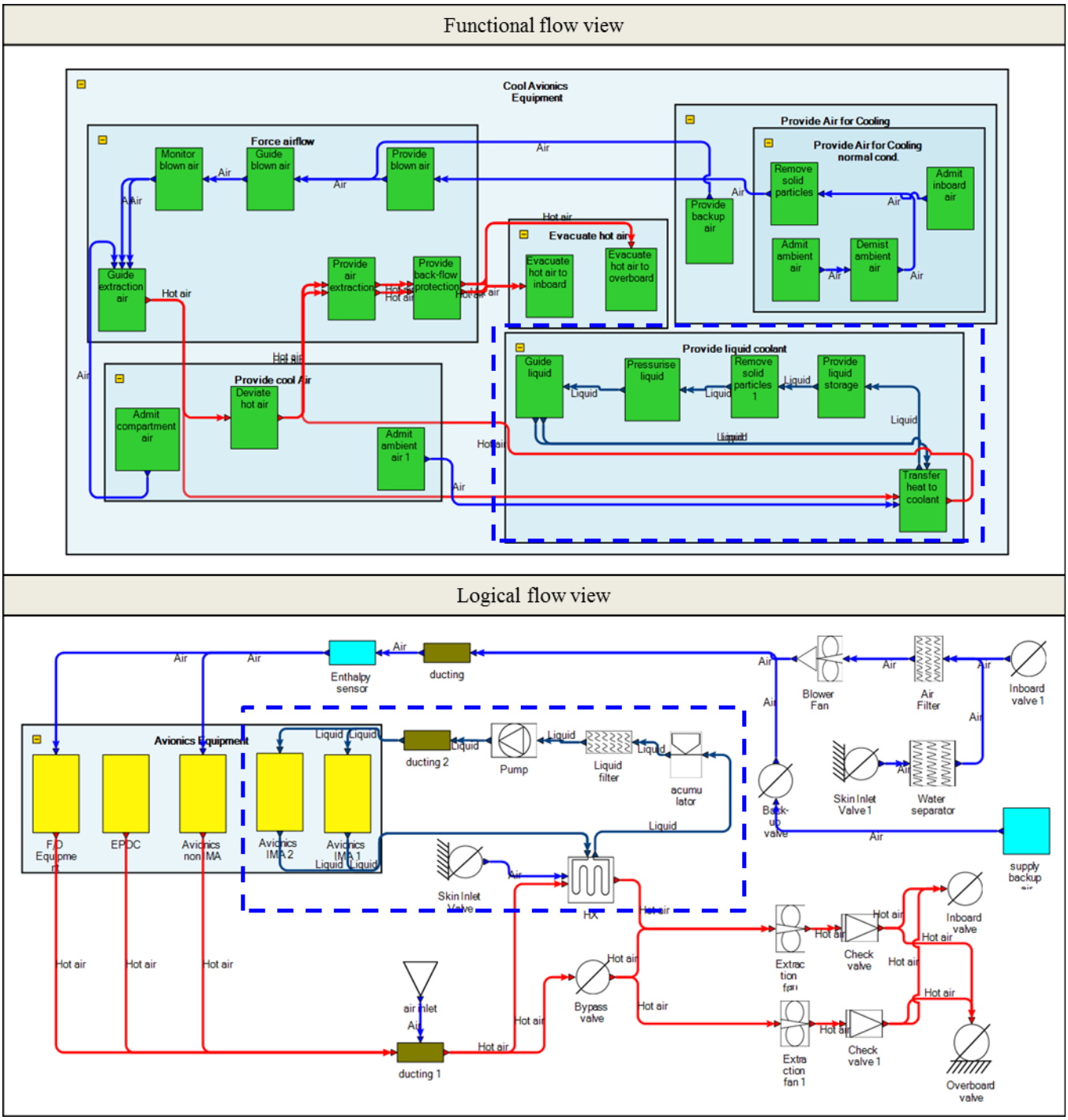


Figure 8: Functional and logical view of the novel hybrid (air- and liquid- cooled) AVS architecture.

B. Architects Feedback

In order to obtain feedback from the architects, a semi-structured questionnaire was developed. The questions consisted of a combination of open-format and closed-format Likert questions (i.e. questions to determine the extent to which the architect agrees with a particular statement). The questions were formulated in such a way that the extent to which the framework met the criteria for a creativity-enhancing engineering design tool could be determined. These criteria were obtained from Chakrabarti[15] and adapted to suit the purposes of the current work.

The questionnaire was completed by a number of system architects ('internal customers' in the TOICA project) and modeling and simulation (M&S) experts – also participants in the TOICA project. This activity took place in a session immediately following the training and evaluation exercise, described in Section IV.A above. The participant architects had between five and twenty years of experience. The overall responses to the Likert questions are shown in Figure 9. On questions 2.4 and 2.6, 40% of the respondents chose "disagree". This was attributed to software "glitches" that were experienced, rather than deficiencies with the approach, as later discussions with the participants indicated that they indeed agreed with the statements in questions 2.4 and 2.6.

These results, along with the answers to the open-format questions, indicated that, on the whole, the framework does enable innovation during architectural design. Specifically, it enables the architect to:

- Create accurate and consistent system architectures.
- Visualise and navigate architectures.
- Make variations on existing architectures.
- Easily rearrange architectural elements and connections, combine them in novel ways, and to reuse them for other architectures.
- Easily introduce new architectural elements and connections to create or explore novel architectures.
- Create or explore novel architectures more efficiently.
- Spot integration problems at early stage of architectural design of [aircraft] systems.

Constructive criticism was also expressed with regard to functionality. Some critical remarks were related to standard functionalities expected from commercial tools (e.g. the 'undo' button). These had to be factored out of the analysis of results, since the aim was to demonstrate the principles of the proposed approach.

The free discussion following the questionnaire revealed that the function-to-means mapping, as demonstrated by the tool, may not be immediately accepted by the wider aircraft systems architect community, who prefer to work solely in the logical domain. However, the architects who took part of the evaluation agreed unanimously that, at a minimum, the proposed approach offers a means for rationale capture, which is seen as crucial when exploring past solutions and for explaining a proposed architectural solutions to customers who are not experts in the field.

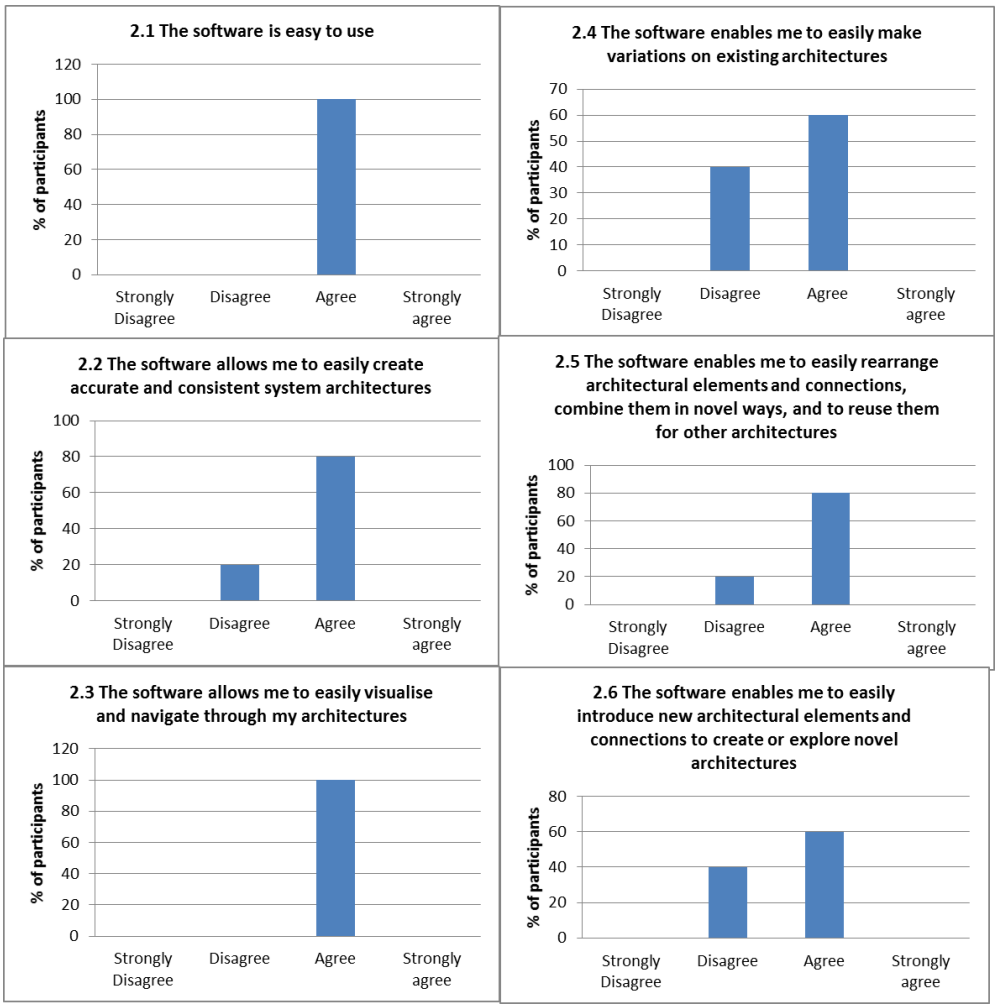


Figure 9: Evaluation questionnaire: Likert questions results.

V. Conclusions

Presented is a novel framework for (early) architectural design with a particular emphasis on enabling innovation. The framework defines data structures that allow the architect to operate simultaneously in the functional and logical domains. The aim is to enable both the "in the box" and "out of the box" approaches to innovative problem solving. A prototype tool, Arcadia Architect, was developed within the TOICA project, which was used for the evaluation of the framework by practicing architects. The evaluation confirmed that, on the whole, the approach enables the architects to effectively express their creative ideas when synthesizing architectures. The proposed approach was especially acknowledged as the way forward for rationale capture.

The framework can be extended to include the requirements and physical domains. This forms part of our future work, which will involve the development of fast methods for geometry representation and spatial layout synthesis. These are intended to enable an interactive physical domain, operating in synchrony with its functional and logical counterparts. Work is also already underway to incorporate conditional (time dependent) views of the evolving architectures and to enable the synthesized architectures to be exported to computational analysis tools and to widely accepted product definition standards (of which SysML is a main candidate).

VI. Acknowledgments

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013, TOICA project), under grant agreement n° 604981.

VII. References

[1]Scholz, D., "Aircraft Systems Reliability, Mass, Power and Costs," *European Workshop on Aircraft Design Education*, Linköping University, Sweden, June 2 - June 4, 2002.

[2]Sinnett, M., "787 No-bleed systems: saving fuel and enhancing operational efficiencies". *Boeing Aeromagazine*, Vol. 28, No.4, 2007, pp 6-11.

[3]Rouvreau, S., Mangeant, F., and Arbez, P. 2015. "TOICA Innovations in Aircraft Architecture Selection, Uncertainty Management in Collaborative Trade-offs." Paper presented at the 6th IC-EpsMsO Conference, Athens, Greece, July 8–11.

[4]Arbez, P. 2015. "Innovations in Aircraft Architecture Trade-off." *Proceedings from the Seventh European Aeronautics Days: Aviation in Europe – Innovation for Growth*, London, October 20-22.

[5]Kleiner, S. and Kramer, C., "Model Based Design with Systems Engineering Based on RFLP Using V6," *Smart Product Engineering*, edited by M. Abramovici, and R. Stark, LNPE, Springer-Verlag, Berlin Heidelberg, 2013. pp. 93–102. DOI: 10.1007/978-3-642-30817-8_10.

[6]VDI 2206, "Design methodology for mechatronic systems," Verein Deutscher Ingenieure (VDI), Düsseldorf 2004, ICS 03.100.40; 31.220, http://www.vdi.eu/uploads/tx_vdirili/pdf/9567281.pdf [cited 15 October, 2015].

[7]ANSI/EIA-632, "Processes for engineering a system," Electronic Industries Alliance, Government Electronics and Information Technology Association, Engineering Department, Arlington, VA (approved January 7, 1999).

[8]Suh, N.P., *The Principles of Design*, Oxford University Press Inc., USA, 1990

[9]Suh, N.P., *Axiomatic Design: Advances and Applications*, Oxford University Press Inc., USA, 2001.

[10]Tate, D., "A Roadmap for Decomposition: Activities, Theories, and Tools for System Design," Ph.D. Thesis, Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA, USA, Feb. 1999.

[11]Guenov, M. D. and Baker, S., "Application of Axiomatic Design and Design Structure Matrix to the Decomposition of Engineering Systems," *Journal of Systems Engineering*, Vol. 8, No. 1, 2005, pp. 29-40.

[12]Altshuller, G. S., *Creativity as an Exact Science: The theory of the solution of inventive problems*, Gordon and Breach Science Publishers Inc., 1984. (Origin: Altshuller, G. S., *Creativity as an exact science.* Sovietskoe radio, Moscow, 1979.)

[13]Altshuller, G. S., Shulyak, L., and Rodman, S., *40 Principles: TRIZ keys to technical innovation*, Technical Innovation Center, Inc., 1998.

[14]Horowitz, R., "Creative problem solving in engineering design," Ph.D. Thesis, Faculty of Engineering, Tel-Aviv University, May 1999.

[15]Chakrabarti, A. "Defining and supporting design creativity." *DS 36: Proceedings DESIGN 2006, the 9th International Design Conference, Dubrovnik, Croatia*. 2006.

[16]ISO/IEC 15288, "Systems and software engineering - System life cycle processes," 2008.

[17]Finke, R. A., Ward, T. B. and Smith, S. M., *Creative Cognition*, MIT Press, Cambridge, Massachusetts, 1992.

[18]Howard, T. J., Culley, S. J., and Dekoninck, E., "Describing the creative design process by the integration of engineering design and cognitive psychology literature," *Design Studies*, Vol 29 No. 2, March 2008, pp.160-180

[19]Sagiv, L., Arieli, S., Goldenberg J., and Goldschmidt, A., "Structure and freedom in creativity: The interplay between externally imposed structure and personal cognitive style," *Journal of Organizational Behavior* (2009), [online journal], Wiley InterScience, DOI: 10.1002/job.664, URL: http//www.interscience.wiley.com, [cited 28 October, 2015].

[20]Vermaas, P.E., "On the Formal Impossibility of Analyzing Subfunctions as Parts of functions in Design Methodology," *Research in Engineering Design*, Vol. 24, No. 1, pp 19-32, 2013.

[21]Nickerson, R.S., "Enhancing Creativity", In R.J. Sternberg (Ed.), *The handbook of creativity*, pp. 392-430. New York: Cambridge, 1999.

[22]Sawyer, R. K., *Explaining Creativity: the Science of Human Innovation*. Oxford University Press, Inc., 2012.

[23]Browning, T. R., "Applying the Design Structure Matrix to System Decomposition and Integration Problems: A Review and New Directions," *IEEE Transactions on Engineering Management*, Vol. 48, No.3, pp. 292-306, 2001.



https://dspace.org/

Aircraft systems architecting: a functional-logical domain perspective

Guenov, Marin D.; Molina-Cristobal, Arturo; Voloshin, V.; Riaz, Atif; van Heerden, Albert S. J. ...

2016-06-17
AIAA
https://doi.org/10.2514/6.2016-3143
Guenov MD, Molina-Cristobal A, Voloshin V., et al. (2016) Aircraft systems architecting: a functional-logical domain perspective. 16th AIAA Aviation Technology, Integration, and Operations Conference, AIAA AVIATION Forum, 13-17 June 2016, Washington, USA. Paper number AIAA 2016-3143

Downloaded from a DSpace repository. https://dspace.org/