Models of requirements for avionics architecture synthesis: safety, capacity and security

Laurent Zimmer, Pierre-Alain Yvars, Michaël Lafaye

Abstract Embedded computing platforms must support the execution of an increasing number of system functions and must scrupulously respect multiple functional and non-functional constraints. Their architecture is more and more complex and consequently the preliminary definition of eligible architectures based on the competence and experience of a small number of expert designers is an increasingly difficult activity. In the future, they will have to use tools to assist in the generation of correct by construction architectures. In this context, we have been interested in formal requirements modeling and in the computer resolution of these formal models to find eligible architectures. We studied an industrial problem of assisted deployment of system functions on an embedded modular avionics platform with capacity requirements on hardware resources and security and safety requirements on functions. We developed a model-based approach using the DEsign Problem Specification (DEPS) language, a language dedicated to modeling and solving design problems. The results obtained show that it is possible to model complex requirements at the level required by the system architect and that taking them into account during the resolution process generates correct solutions to them.
Keywords architecture synthesis, formal specification, requirements, safety, security, capacity, DEPS.

1 Introduction

Avionics systems are becoming increasingly complex. According to [1], in the military field, we have gone from 15 subsystems and less than 40% of software-intensive systems for the F16 to 135 subsystems including 90% of software-intensive systems for the F35. The evolution in the civilian domain is less extreme but it follows the same trend. At the same time the number and complexity of technical or regulatory specifications have increased, as has the complexity of industrial organization. This overall increase in complexity has led to an explosion in development costs and lead times which has led industrialists to review their design methods and tools. According to studies financed by DARPA [1, 2, 3] the new design process to be imagined must be based on the development of four key elements:
1. Abstraction-based design tools;
2. System complexity metrics;
3. Advanced methods of architecture synthesis;
4. Robust uncertainty management.

Laurent Zimmer, Michaël Lafaye
Dassault Aviation – Direction Générale Technique
78 quai Marcel Dassault - 92552 Saint-Cloud - France
laurent.zimmer@dassault-aviation.com , michael.lafaye@dassault-aviation.com

Pierre-Alain Yvars
Institut Supérieur de Mécanique de Paris (SupMéca) - QUARTZ
3 rue Fernand Hainaut – 93407 Saint-Ouen - France
pierre-alain.yvars@supmeca.fr

The work presented concerns exclusively the first [2] and the third point [3]. For the first point, the aim is to develop or use a formal description language (FDL) capable of representing both a complex system and the specifications or requirements related to it. In particular, the FDL must allow the system to be represented at compatible levels of abstraction of the various design stages, including the preliminary design stages. The languages envisaged are AADL, UML or SysML.
The second point is to have "advanced means of architecture synthesis" available very early in the design process, which would allow automatic exploration of the design space to search for eligible architectures, i.e. those compatible with the various system requirements.
The idea being that the complexity of the systems to be designed will make the task of listing and evaluating candidate architectures beyond the reach of experts if they are not supported by Information Technology.
To develop a design tool capable of synthesizing eligible architectures of software processing chains have been proposed in seminal work [3, 4, 5].
More recent work has focused on the development of a language for modeling design problems with a view to solving them [6]. The aim is to develop a true representation language of the problem to be solved. The DEPS language allows, on the one hand, to model a sub-defined system and its associated requirements and, on the other hand, to solve to find a synthesis solution.
The purpose of this paper is to show the capabilities of DEPS by formalizing a problem of deployment of aircraft functions on an embedded computer architecture of integrated modular avionics type. We have limited ourselves in this paper both in types of architecture elements (the CPUs) and in types of requirements (safety, maximum memory capacity of the CPUs, security). These elements are extracted from a complete Integrated Modular Avionic (IMA) problem that was formalized in DEPS and solved using the DEPS Studio environment.
The paper is organized as follows: we first position our work in the context of modular avionics, operational safety and the distribution of roles between actors, then we present the DEPS language and the associated tools we use. Then we will describe the case study and the problem posed. Then we will discuss the modeling that has been done in DEPS and the results we have obtained. Finally we will present some perspectives for the evolution of this work both from the point of view of the generalization of the case study and the evolution of the DEPS formalism.

2 Problem

2.1 General framework

The problem consists in deploying system functions consisting of software components on IMA-type hardware [11, 12]. The IMA allows software development to be decoupled from the underlying hardware. In return, it is necessary to be able to allocate computing and communication resources that allow the system functions to be executed. In the IMA, the concept of partitioning reflects this resource allocation.
The aim is to find a scheme for allocating computers to software that meets the needs in terms of resources while respecting resource capacities, latency con-

straints for the execution of through functional chains (i.e. processed via the execution of several software programs) and operational reliability. Due to these multiple constraints and the large number of deployments, finding a satisfactory allocation manually becomes very difficult.
In order to illustrate our approach, we will limit ourselves to the case of a software deployment problem on a set of IMA CPUs that must comply with operational safety, memory capacity and security requirements. These requirements are among the most important, as all civil aircraft are designed with an approach aimed at reducing the risk of accidents and therefore potentially human losses to a minimum.

2.2 Requirements

2.2.1 Safety requirements

The purpose of safety is to establish permissible levels of reliability, availability and maintainability of essential aircraft systems to ensure the safety of the aircraft in flight and on the ground. The requirements to achieve these levels result from the analysis of cases of failures and malfunctions, which may lead, for example, to loss of functionality or data corruption.
In terms of platform architecture, these analyses lead to a set of safety patterns. This is a proposal to divide an aircraft function into software components with associated segregation or dissimilarity requirements inside. For example, the robustness of a function in the event of a failure leads to a redundancy type "pattern" of the software executing this function and its deployment on two distinct computers. We will talk about hardware segregation of the resources used. These requirements are not only applicable at the software level, but can concern for example the entire processing chain of a given aircraft function, by requiring that this function be carried out via two segregated chains. In this case, all the software making up the first chain must be deployed on separate computers from the software making up the second chain. An illustration of "patterns" and associated requirements will be given in the case study section..

2.2.2 Security requirements

Security studies produce recommendations on the need to separate the deployment of certain functions. Functions do not have the same security level [13]. Three levels are defined: High-level-trust, Medium-level-trust and low-level-trust. They therefore impose constraints of segregation between functions at the system level.

2.2.3 Capacity constraints

This is to take into account the requirements of respecting the maximum memory capacity of each of the CPUs. On each computer of the adopted deployment solution, the sum of the memory capacities required for each partition associated with it should be less than the memory size of the computer.
We are therefore confronted with a problem of "formal" representation of patterns and associated safety, security and capacity requirements (see Fig.1.) We are thus

positioning ourselves in a modeling and synthesis approach, as opposed to the current analysis approaches.

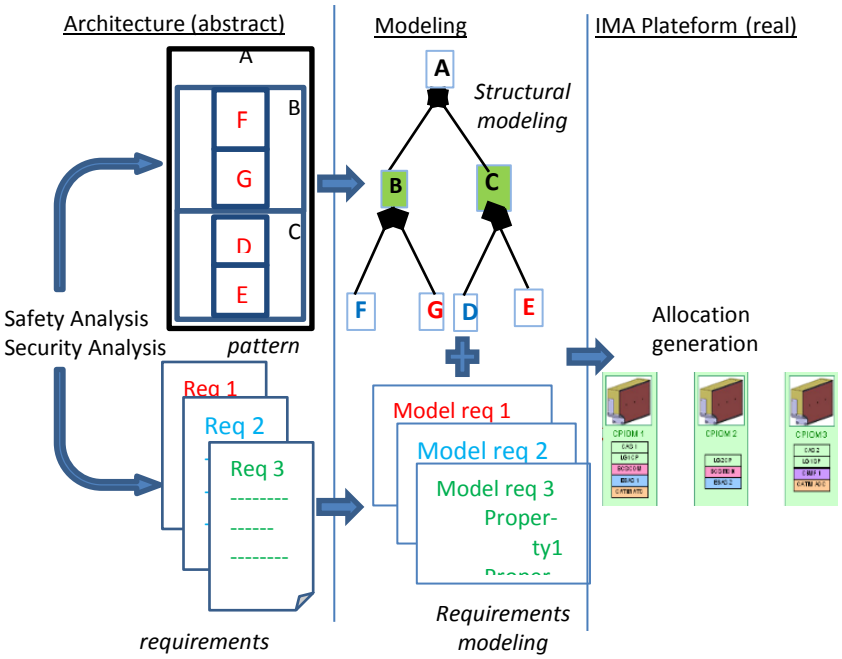

Fig.1 . Requirements building process

3 DEPS Language

3.1 Paradigm

Some authors [6] point out the difficulties of using formalisms such as SysML, initially designed to represent totally defined systems (System Modeling Language) to model problems. The current limitations of the main approaches are the low level of variability that can be taken into account and a weak coupling between the formalism and the solver, which means that the development of models is carried out in case of a problem in the solver language and not in the SysML language. These limitations have been pointed out by [7].The DEPS language and its recent integrated modeling and solving environment DEPS Studio are an attempt to address these limitations.

The Design Problem Specification (DEPS) language is commonly referred to as a Domain Specific Language (DSL). The target application domain is the specification and resolution of engineering problems, particularly those encountered in product or system design: sizing, configuration, allocation and architecture generation. The industrial interest is to use a unique formalism and tooling to model and solve all these categories of synthesis problems.

DEPS is an external (as opposed to internal or embedded) dedicated language. The source code is therefore independent of any host generalist language.

The DEPS language can be seen as a combination of a software or system modeling language and a mathematical programming language. From the former have been borrowed the features of structuring and abstraction which allow to represent the elements and the system under study. From the latter were borrowed the mathematical concepts necessary to solve the engineer's problems: unknowns, equations and inequalities.
This combination makes it possible both to represent design problems and to pose and then solve or optimize the systems of equations and inequalities that govern them [8]. DEPS is supported by the DEPS Link society, a non-profit organization [9].

3.2 Main characteristics

3.2.1 The Model

The fundamental feature of the language is the Model. Any Model is defined using the keyword *Model* followed by its name and its (possibly empty) list of arguments. It contains in order: a declaration-definition area for *Constants*, a declaration area for *Variables*, a declaration-creation area for *Elements* and a definition area for *Properties*. The definition of a DEPS Model ends with the keyword *End* (see Fig.2).
The properties of a Model are the equations and inequalities that relate to the constants and variables of that Model. A Model therefore contains all the ingredients necessary to set the constraints that govern an instance of this *Model*. All the algebraic operators are available to build the properties: arithmetic, logarithmic, exponential, trigonometric, hyperbolic, power,.. Some specialized constraints can also be set as constraints on data catalogs. Any instance of a Model will necessarily contain the set of constants, variables and Elements expressed in the Model and will necessarily have to verify the set of properties of the Model.

```
Model Cpu(ram, icpu)
Constants
ram : Memory ;
icpu : CpuIndex ;
Variables
Elements
Properties
End
```

```
Model Partition()
Constants
Variables
icpu : CpuIndex;
Elements
Properties
End
```

Fig.2 . DEPS models of CPU and partition

As in an object language there is inheritance and composition: a Model can be extended and inherit constants, variables and properties from another Model and it can be composed of elements. In this case, elements are built inside the Model. They are instances of other Models. To build an element, you have to call the constructor of the reference model with values given to its arguments. Elements can be passed as arguments to a Model creating an aggregation link with it. In this case, the argument elements are named in the list of model arguments and de-

clared in the Elements zone, specifying the signature of the model to which they refer. *Constants* can also be passed as arguments to a *Model*, allowing the creation of Parametric Models. Any argument is named in the argument list of the model and is declared in its respective field. Thus, in Figure 2, the CPU model has two arguments *ram* and *icpu* whose types are declared in the *constant* field of the model. DEPS also supports polymorphism.
Modeling a problem is therefore the same as specifying DEPS Models. The problem to be solved is expressed using the keyword Problem. A Problem is a Model without arguments.

3.2.2 The Quantity

In DEPS, both constants and variables are associated with types of physical or technological magnitudes called quantities (Quantity). They are necessary in Systems Engineering (see Fig.3).
A Quantity has :
- A basic type of quantity (called QuantityKind). For example, Real, Integer, Length;
- A min (resp. max) bound that represents the minimum (resp. maximum) value that can be taken by any constant or variable having the defined quantity as its type;
- A unit of the quantity. For example the meter *m* for a length and *u* for quantities without unit.

A QuantityKind has a basic type (integer or real), a min and max limit and the dimension in the sense of the dimensional analysis of the quantity. For example L for a length or U for a dimensionless quantity;

QuantityKind Integer **Type** : integer ; **Min** : -maxint; **Max** : +maxint; **Dim** : U ; **End**	**Quantity** CpuIndex **Kind** : Integer ; **Min** : 1 ; **Max** : 7 ; **Unit** : u ; **End**

Fig.3 . QuantityKind and Quantity

3.2.3 Implementation

The DEPS Studio integrated modeling and solving environment associated with the DEPS language includes [10] model editing functions, project management functions based on a package mechanism, a compiler and a solver.
An integration approach rather than a model transformation approach has been chosen. Indeed, in the case of solving a sub-defined system synthesis problem, it will be necessary in case of unsatisfactory computation results to perform a model tuning in the DEPS language. By deliberately opting for an integration approach, we choose this fine-tuning process.
The computational methods we use are taken from the work on the resolution of CSP [14]. The solver implements a revised HC4 propagation method [15] on equations and inequalities for four types of domains: open real intervals, integer intervals, enumerated sets of floating values and enumerated sets of signed integer

values. The object-oriented architecture of the solver has been designed so that it can be extended to other propagation and/or resolution methods.

4 DEPS modeling of system function requirements

4.1 Description of the LGS function

The LGS Landing Gear System management function is initially deployed on a single software component.
The dependability analysis imposes two requirements (see Fig.4):
- Requirement 1: a second software component (called *SC* for "safety" component) must be provided, in order to take over in case of loss of the main component (*MC*). These software components, called applications, must be hardware independent (*HI*) so that a hardware failure affecting *MC* does not affect *SC*;
- Requirement 2: this *MC* + *SC* processing chain, called channel, must be duplicated into two hardware dissimilar (*HD*) channels in order to be robust to the loss of the first chain.

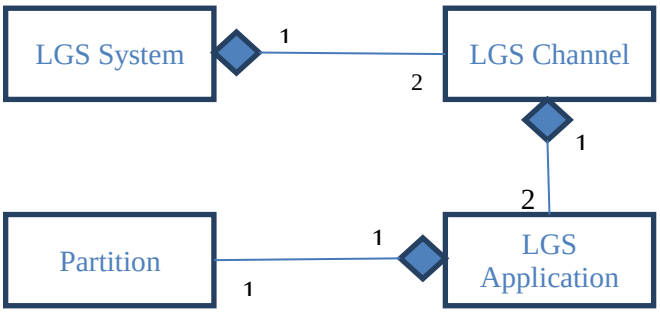

Fig.4. Macroscopic view of the LGS function

From a processing point of view, the provider of the function tells us that each component can be projected on a single partition. We therefore end up with a decomposition scheme resulting on four partitions, with the following constraints:
- Constraint 1: the {SC1, MC1} partitions must be hardware-segregated from the {SC2, MC2} partitions i.e. the {SC1, MC1} partitions do not share the same CPUs than the {SC2, MC2) partitions (see Requirement 1);
- Constraint 2: SC1 and MC1 partitions must be physically segregated i.e. partitions do not share the same CPU. They are not co-located (*NonColoc*)(see Requirement 2);
- Constraint 2bis: SC2 and MC2 partitions must be physically segregated (see Requirement 2).

4.2 Formalization in DEPS

4.2.2 Structural modeling

We have modeled the components of the aircraft landing gear system management (LGS) function at the different levels of abstraction necessary to express the re-

quirements using the 2018 version of the DEPS language. These are the models of: aircraft function, channel, application and partition.

- A partition must be projected to a computing resource to run. Each CPU resource is represented by a maximum memory capacity (*ram*) and a resource index (*CpuIndex*). The CPU index (*icpu*) that will be assigned to a partition when the problem is solved is unknown (see Fig. 2).
- An application is split into one or more partitions. Any application is characterized by the memory capacity required for its execution as well as the partition(s) of which it is composed (see Fig. 5). For the landing gear function, the applications are processed into one partition.

Model OnePartApplication () **extends** Application **Constants** ram1 : Memory = 100; default; **Variables** **Elements** P : Partition(ram1); **Properties** **End**	**Model** ALGSsp () **extends** OnePartApplication **Constants** ram1 : Memory = 2100 ; redefine; **Variables** **Elements** **Properties** **End**

Fig.5 . DEPS model of the CS application of LGS

- A processing chain is here limited to one or more software components. The channels for the landing gear function are composed of two applications (Fig. 6).

Model TwoAppChannel () **extends** Channel **Constants** **Variables** **Elements** App1: Application() ; App2: Application() ; **Properties** **End**	**Model** CLGS () **extends** TwoAppChannel **Constants** **Variables** **Elements** App1 : ALGScp() ; App2 : ALGSsp() ; IM1 : IM (App1, App2) ; **Properties** **End**

Fig. 6. DEPS model of a LGS channel

- An aircraft function is composed of one or more channels. The channels are elements of the aircraft function model and are therefore elements of the processing chain. In the case of the landing gear, the function comprises two channels (see Fig. 7).

Model TwoChannelFunction () **extends** System **Constants** **Variables** **Elements** Ch1 : Channel(); Ch2 : Channel(); **Properties** **End**	**Model** FLGS () **extends** TwoChannelFunction **Constants** **Variables** **Elements** Ch1 : CLGS(); Ch2 : CLGS(); HD1 : HD(Ch1, Ch2); **Properties** **End**

Fig. 7. DEPS model of the landing gear function

4.2.3 Requirements modeling

We have created DEPS models for each requirement depending on whether it is for partitions, applications, channels or systems.

- Non Co-location of two partitions: To say that two partitions are not co-located is to establish a difference relationship between the two partitions and their two *icpu* (see Fig. 8).
- Hardware independence of two applications (*HI*): Two applications are hardware independent if and only if their respective partitions are not colocated two by two (see Fig. 8).

Model NonColoc (P1, P2) **Constants** **Variables** **Elements** P1 : Partition ; P2 : Partition ; **Properties** P1.icpu <> P2.icpu; **End**	**Model** HI(App1, App2) **Constants** **Variables** **Elements** App1 : OnePartApplication ; App2 : OnePartApplication ; NC1 : NonColoc(App1.P, App2.P) **Properties** **End**

Fig.8 . Requirements modeling on partitions and applications

- Hardware dissimilarity of two channels (*HD*): Two channels are hardware dissimilar if and only if their respective applications are hardware independent two by two (see Fig.9).
- Security requirements mean that some functions must be physically segregated (*HS*). This means that their channels must be hardware dissimilar (see Fig.9).

Model HD (Ch1, Ch2) **Constants** **Variables** **Elements** Ch1 : TwoAppChannel; Ch2 : TwoAppChannel; IM1 : HI (Ch1.App1, Ch2.App1); IM2 : HI (Ch1.App1, Ch2.App2); IM3 : HI (Ch1.App2, Ch2.App1); IM4 : HI (Ch1.App2, Ch2.App2); **Properties** **End**	**Model** HS(F1, F2) **Constants** **Variables** **Elements** F1 : TwoChannelFunction; F2 : TwoChannelFunction; HD1 : HD(F1.Ch1, F2.Ch1); HD2 : HD(F1.Ch1, F2.Ch2); HD3 : HD(F1.Ch2, F2.Ch1); HD4 : HD(F1.Ch2, F2.Ch2); **Properties** **End**

Fig.9 . Requirements modeling on channels and functions

The capacity of each CPU resource must not be exceeded. A capacity model has been created for this purpose (see Fig.10). It expresses that the load of a given computer (*Cpiom*) must not exceed the maximum memory capacity of the cpu (*Cpiom.ram*). To do this, the memory capacity necessary for the execution of the

Partition $Part_i$ is weighted by a factor of 1 or 0 ($capa_i$ boolean expression) whether or not the Partition $Part_i$ is projected on the *Cpiom* computer.

```
Model Capacity(Part1, Part2, …, Part31,Cpiom)
Constants
Variables
expr Load :
expr capa1 : Bool ;
…
expr capa31 : Bool;
Elements
Cpiom : Cpu[Memory, CpuIndex];
Part1 : Partition[Memory];
…
Part31 : Partition[Memory];
Properties
capa1 := max(1-abs(Part1.icpu-Cpiom.icpu),0) ;
…
capa31 := max(1-abs(Part31.icpu-Cpiom.icpu),0) ;
Load := capa1 * Part1.ram  + … +  capa31*Part31.ram ;
Load <= Cpiom.ram;
End
```

Fig. 10. DEPS capacity model

Finally, the general problem is posed by creating all the CPUs, the systems to be un-deployed and the associated safety and security requirements (see Fig.11).

```
Problem IMA
Constants
Variables
Elements
cpu1 : Cpu(  30000,1);  … cpu6 : Cpu(  30000, 6);
(* fonctions creation with safety requirements inside*)
FCATIM1: FCATIM();          (* CATIM Function*)
FLGS1    : FLGS()    ;          (* LGS Function    *)
..
FAFCS1 :  FAFCS() ;          (* AFCS Function  *)
(* security requirements *)
 HS1 : HS(FCATIM1, FLGS1);  ..   HS6 : HS(FCATIM1, FAFCS1);
(* capacity requirements *)
capacity1 : Capacity(FLGS1.Ch1.App1.P, FLGS1.Ch1.App2.P, …  , cpu1);
…
capacity 6 : Capacity(FLGS1.Ch1.App1.P, FLGS1.Ch1.App2.P,  …,  cpu6);
Properties
End
```

Fig. 11. DEPS model of IMA problem

4.3 Results

The full industrial case study consists of a problem of deploying seven aircraft functions (including braking, fault escalation, communication, landing gear management, etc.) on a platform composed of computers, to be determined according to the associated operational safety, security and capacity requirements.

For each function, we therefore have as input a document describing the breakdown into software components and then the projection in terms of partitions, with requirements for hardware segregation (projection of segregated components on separate computers) The number of software components, partitions and channels differs according to the functions.

After modeling the seven aircraft functions and their respective requirements, we obtained the solutions for deploying the partitions of these functions on the CPUs. The resolution shows that six calculators are necessary and sufficient to deploy the seven aircraft functions (see Fig.12). The results are obtained after 2 seconds of calculation on an Intel Core i5 4GB RAM Personal Computer.

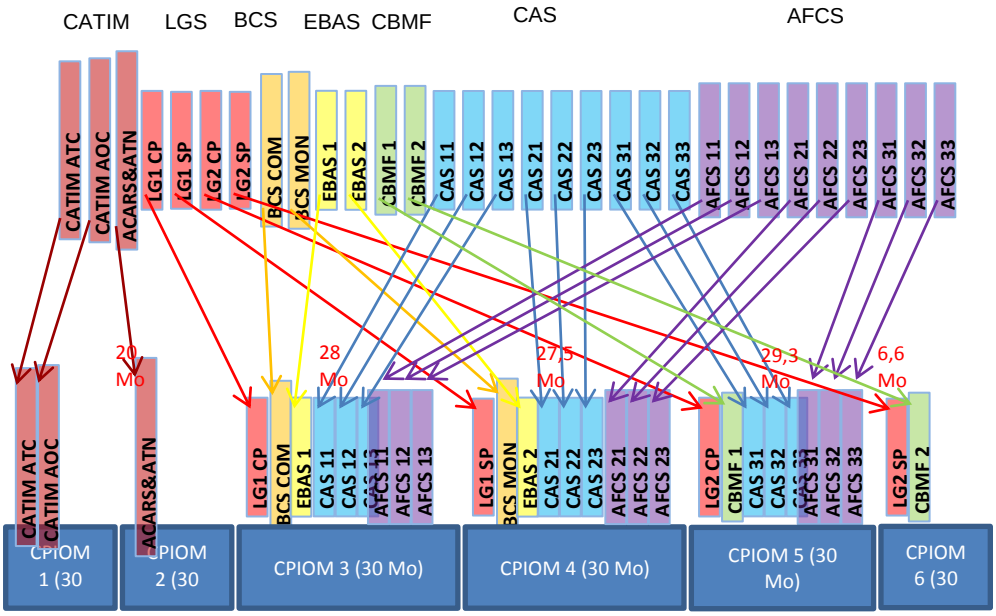

Fig.12. DEPS Studio solution for the deployment of the seven aircraft functions

5 Conclusions and perspectives

In this article we have shown that it is possible to capture capacity, safety and security requirements that are difficult to formalize directly without a suitable formal language. We used the DEPS language which allowed us to model the right abstractions at the right level to describe a sub-defined model of IMA architecture on the one hand, and requirement models on the other hand. The deployment problem was solved by applying constraint satisfaction methods to the properties

of the problem. The structuring features offered by the language facilitate the reuse of the models developed. Future work will focus on taking into account additional elements in the processing chains (sensors, power supplies, actuators ...) as well as on additional recommendations for DEPS language specifications.

References

[1] Becz, S., Pinto, A., Zeidner, L. E., Banaszuk, A., Khire, R., and Reeve, H. M.,Design System for Managing Complexity in Aerospace Systems," proc of the 13th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, Texas, 2010

[2] Pinto A., Becz S., Reeve H.M., Correct-by-construction design of aircraft electric power systems, proc of 10th AIAA Aviation Technology, Integration, and Operations Conf., 2010. Published on line https://doi.org/10.2514/6.2010-9263.

[3] Zeidner L, Reeve H, Khire R, Becz S., Architectural Enumeration and Evaluation for Identification of Low-Complexity Systems, proc of 10th AIAA Aviation Technology, Integration, and Operations Conf., 2010. Published on line https://doi.org/10.2514/6.2010-9264

[4] Bieber P., J.P Bodeveix J.P., Castel C., Doose D., Filali M., Minot F., Pralet C., Constraint-based Design of Avionics Platform - Preliminary Design Exploration, proc of ERTS 2008 Toulouse, January 2008.

[5] Albarello, N., Welcomme J.B. and Reyterou C., A formal design synthesis and optimization for systems architectures, proc of MOSIM'12, Bordeaux, France, 2012.

[6] Creff S, Le Noir J, Lenormand E & Madelénat S. Towards Facilities for Modeling and Synthesis of Architectures for Resource Allocation Problem in Systems Engineering. Proc of 24th Systems and Software Product Line Conference. Montreal, 2020.

[7] Shah A.A., Paredis C.J.J., Burkhart R. & Schaefer D. Combining Mathematical Programming and SysML for automated Component Sizing of Hydraulic Systems. Journal of Computing and Information Science in Engineering (JCISE), 12(4), 2012. https://doi.org/10.1115/1.4007764

[8] Yvars P.A., Zimmer L., DEPS un langage pour la spécification de problèmes de conception de systèmes, proc of MOSIM'14, Nancy, 2014.

[9] DEPS Link, www.depslink.com

[10] Yvars P.A., Zimmer L., "DEPS Studio : Un environnement intégré de modélisation et de résolution de problèmes de conception de systèmes", proc of 8ème Conférence en ingénierie du logiciel (CIEL 2019), Toulouse, 2019.

[11] ARINC Specifications 653-1 Avionics Application Software Standard Interface, SAE ITC, 2017

[12] DO-297 Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations. RTCA, 2017. Available on www.rtca.org

[13] DO-178C Software Considerations in Airborne Systems and Equipment Certification. RTCA, 2011. Available on www.rtca.org

[14] E. Tsang, Foundations of Constraint Satisfaction. London and San Diego: Academic Press, 1993. ISBN 0-12-701610-4

[15] Benhamou F., Goualard F., Granvilliers L., Puget J.F., Revising Hull and Box consistency,proc of 16th International Conference on Logic Programming, 1993