Toward a correct by construction design of complex systems : The MBSS approach

Pierre-Alain Yvars, Laurent Zimmer

Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 00 (2022) 000–000

www.elsevier.com/locate/procedia

32nd CIRP Design Conference

Towards a correct by construction design of complex systems: The MBSS approach

Pierre-Alain Yvars[a]*, Laurent Zimmer[b]

[a]*ISAE-Supméca, QUARTZ EA7393, 3 rue Fernand Hainaut, 93407 Saint Ouen Cedex, France*
[b]*Dassault Aviation, Direction de la prospective,78 quai Marcel Dassault , 92552 Saint Cloud, France*

* Corresponding author. Tel.: +33-(0)49-452-2925; fax: +33-(0)49-452-2929. *E-mail address:* pierre-alain.yvars@isae-supmeca.fr

Abstract

We present in this paper the Model Based System Synthesis (MBSS) approach for the design of complex systems that are correct by construction. Where the usual Model Based System Engineering (MBSE) approach offers formalisms and tools to represent a candidate system, to analyze it, to simulate it and even to optimize it, MBSS proposes to represent the global design problem using a problem representation language and then to solve it by using adapted synthesis tools producing one or several solutions necessarily satisfying the expressed requirements. The two approaches are therefore complementary; the MBSS being more adapted to the preliminary design and system integration stages. After presenting the different categories of problems encountered in system design (sizing, configuration, allocation, architecture generation), MBSS and MBSE will be positioned in relation to each other. The main concepts of MBSS will be detailed in order to understand the specific representation needs of the approach. The structural and behavioral notions related to the sub-definite systems will be explained as well as the links to be established with the functional and non-functional requirements. The approach is illustrated using the DEPS design problem specification language and the DEPS Studio modeling and solving tool on a system design case study. The DEPS language combines structural modeling features specific to object-oriented principles and ontology definition capabilities for engineers with problem specification features from constraint programming. DEPS Studio is an integrated modeling and solving environment designed to model and resolve system synthesis problems. It allows the engineer to edit, compile, debug and solve problems expressed in DEPS. It integrates a mixed constraint programming solver. The approach can be applied on physical systems, software intensive or mixed systems (embedded or cyber-physical).

Keywords: Model-based approach; System Synthesis; constraint programming; sub-definite system.

1. Introduction

The current general principle in system design is to start with a candidate system, then evaluate its performance and compare its performance with the expected requirements. If the requirements are verified, then the candidate is a solution, otherwise a new candidate should be proposed and the process repeated (cf Fig.1). It is an iterative method of point-to-point design [1] and it is also a generate-and-test method [2] where we move from one candidate to another until we find a valid solution, i.e. a solution that verifies all the requirements of the specifications. The IEEE1220 standard [3] proposes a system design approach based on this method (cf Fig. 1), modeled on the V-cycle [4]. Depending on the point of view adopted with regard to the type of requirements to be validated, we speak of Design For X [5]: Design For Sustainability, Design For Safety, Design for Cost, ...

Thus, several formalisms exist and allow the user to express the definition of a candidate system. The formalisms are adapted to the systems to be defined but also to the behaviors to be modeled: AADL [6], SysML [7], Modelica [8], ...

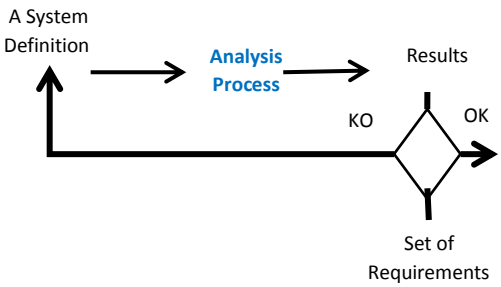

Fig. 1. Iterative design process

From these formalisms, the computational tools that are used allow the user to evaluate performances either by analytical approach or by simulation. The calculation models used are direct models; performance indicator values are calculated according to the given values of the design variables of the candidate system. It is thus possible to verify a posteriori on a candidate system the validity of the requirements, specialty by specialty with the appropriate behavioral models and tools. Thus the requirements are considered separately, discipline by discipline. For a given and fully defined candidate architecture, the structural specialist will, for example, build an adapted finite-element model to enable him to verify the structural requirements. The acoustical specialist will do the same on his side. The reliability specialist will build a reliability model of the architecture to evaluate the reliability of the system. Similarly, the eco-design specialist will perform an Life Cycle Analysis (LCA) of the system in order to evaluate the environmental performance. The performances are evaluated using dedicated simulation and analysis tools such as Modelica [8] for the dynamics of the system and Simapro [9] for the LCA studies. Mathematical optimization methods or meta-heuristics can be used in order to try to approach an admissible solution, i.e. one that satisfies all the requirements. In the multidisciplinary case, this approach is generalized by methods such as Multidisciplinary Design Optimization (MDO) [10].

We propose in this paper a new approach called MBSS for Model Based System Synthesis. Where classical MBSE approaches represent a candidate system, MBSE models the design problem. Where MBSE evaluates the performance of the modeled candidate system to verify a posteriori the validity or invalidity of the requirements, MBSS proposes to use the requirements to build one or several solutions guaranteed correct-by-construction since they satisfy the requirements.

The main question we address in this paper is the design of correct systems by construction. We propose for this purpose a way of looking at system design problems from the synthesis point of view called MBSS. This point of view is equipped with a formal problem modeling language called DEPS and with an integrated modeling and solving environment called DEPS Studio. The observation made by several authors is that the current MBSE tools do not allow a correct by construction approach and we detail in this paper our proposal to lift this lock.

This paper is organized as follows: After presenting the typologies of system design problems and their characteristics, we present the case study that will be used to illustrate our point of view. Then the MBSS approach is detailed from the point of view of the methodology, the modeling and the tool-based problem solving. A comparison with the MBSE will also be proposed before a conclusion.

2. Complex systems design characteristics

2.1. Typology of design problems

Several categories of system design problems can be identified:

- System sizing: problem for which the architecture of the system is known but the values of its structural parameters are unknown (e.g. the length of an object). Unknowns are variables that are often continuous, sometimes discrete. Functional requirements can be quite complex and can be expressed as linear or non-linear algebraic relationships of constants and variables.
- System configuration: Configuration problems involve the choice of components based on a set of compatibility relationships, options, and cardinality. They are most often discrete-dominant problems.
- Resource allocation: Allocation problems involve allocating physical resources to system functions based on a set of functional and non-functional requirements.
- Architecture generation: System architecture problems combine the three previous problems. They are based on a specification combining requirements and constraints to produce architectures that will meet the specification. We can also talk about architecture synthesis.

2.2. Characteristics

A technical system is generally characterized by the structure of its architecture and its behavior. A system design problem must therefore contain a set of unknowns related to the structure of the architecture (size, dimensions, number of components, type of components, allocation ...) and unknowns of behavior (performance and functioning variables).

A system can have several functioning modes subject to the respect of different properties depending on the mode considered.

A system is also subject to the respect of a set of functional requirements (laws of physics and technology, missions to be fulfilled ...) and non-functional requirements (safety, security, eco-design ...).

Depending on the case, these requirements can be:

- Minimum or maximum threshold values on performance indicators calculated by behavioral models. For example, when we want the power delivered by a system to be between two values. We call this “behavioral requirements”.
- Necessary properties for the structure of any solution architecture. For example, in the field of operational safety, when specialists recommend duplicating or triplicating certain functional chains in an onboard aircraft system. This will be expressed by properties on the system architecture that we call “structural requirements”.

3. Use case

3.1. Description

This problem, described in its entirety in [11], is that of the configuration and positioning of an onboard camera. This system is composed of sensors (one to n) whose role is to ensure the stabilization of the image as well as the automatic focus. These sensors are themselves connected to processors (CPUs) (one to n) via a digital video bus. Then, the processors will process the image and compress it to detect obstacles and trigger the vehicle's automatic decision making by transferring the information through "Transceivers" (one to n). In addition, the transceivers are connected to the CPUs through a parallel digital port. Sensors, processors and transceivers have their cost and reliability characteristics available in catalogs. Thus, the camera is stereoscopic when it has two sensor-cpu-transceiver chains or simple in the case of a single chain. All sensors, CPUs and transceivers in the catalogs are not compatible with each other.
In order to obtain a camera that is both efficient and as inexpensive as possible, it is necessary to optimize cost and reliability, which are the two main performance indicators when designing the system. This is done by minimizing the total cost of the system and maximizing the reliability. In addition to optimizing the cost and reliability of the system, it is also necessary to optimize the performance of the system by maximizing it. To do this, the positioning of the camera in the vehicle must be addressed. Correctly positioned, the camera must be able to detect an obstacle that can be very close or far away.

To model this configuration, we rely on the principle of stereoscopic optics to calculate the distance that separates us from an object from the comparison of images in the two views of the cameras. This is called the disparity which is a nonlinear function of the height of the obstacle, the horizontal position of the obstacle, the till angle of the camera, the height of the camera and the characteristics of the sensors (focal length, and size of a pixel).

The system must also be able to operate in two modes: detection of a close obstacle and detection of a distant obstacle. For that, we rule that the image of a distant obstacle on the sensors must have a minimal size that we fix at 50 pixels and the image of the close obstacle must be able to be contained by the sensors.

3.2. Modeling and solving the problem with the common way

This is a problem of architecture configuration and simultaneous positioning.

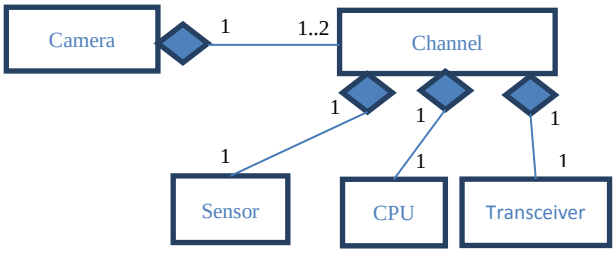

Fig. 2. Architecture of the embedded camera

The camera is characterized by the following six to nine design variables:

- The number of channels *nbChan* (one or two) of discrete nature.
- The reference of the *sensorRef* chosen in each discrete channel.
- The reference of the CPU *CPURef* chosen in each channel of discrete nature
- The reference of the transceiver *TransRef* chosen in each channel of discrete nature.
- The positioning height h of continuous nature
- The positioning angle θ of continuous nature.

The requirements are as follows:

- Compatibility of the components in each channel.
- Minimum cost of the architecture.
- Maximum reliability of the architecture.
- Detection of near obstacles.
- Detection of distant obstacles.

The classical iterative design process will therefore consist in checking each requirement for a given candidate and iterating until a valid solution is found (cf Fig.3).

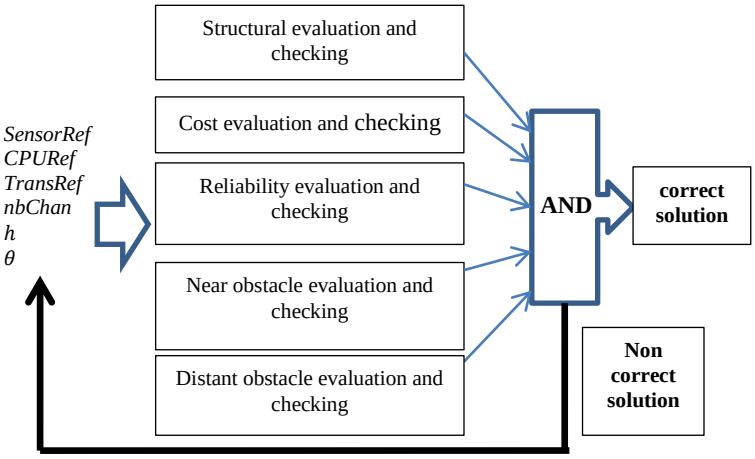

Fig. 3. MBSE way for modeling and solving the problem

4. MBSS approach

4.1. Methodological point of view

Several basic principles characterize the MBSS approach:

- The Set-based design [1] approach in which the design space is reduced as the requirements are placed on the potential solutions.
- Modeling the design problem rather than a candidate system.
- Taking into account simultaneously several heterogeneous requirements.
- Using requirements to reduce the space of requirement based solutions.

The approach for solving the problem is said to be "by satisfying requirements" and not by "validation and verification", since at all times we try to find solutions within the boundaries of the design space represented by the

requirements. The solutions found are then necessarily correct by construction. We then tend towards an I-cycle (cf Fig. 4) of design rather than a V-cycle one.

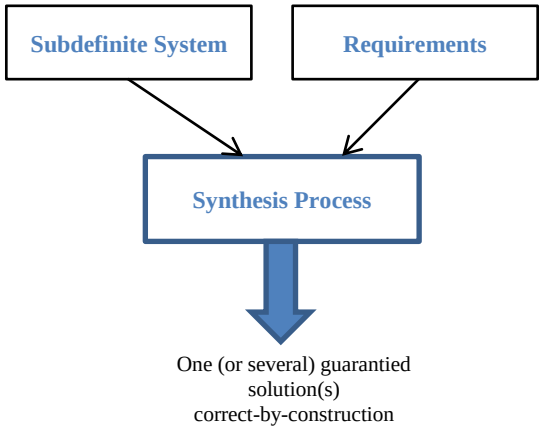

Fig. 4. Toward an I cycle

4.2. Modeling the problem

An object whose unknown values are not fixed a priori will be called a sub-definite object.

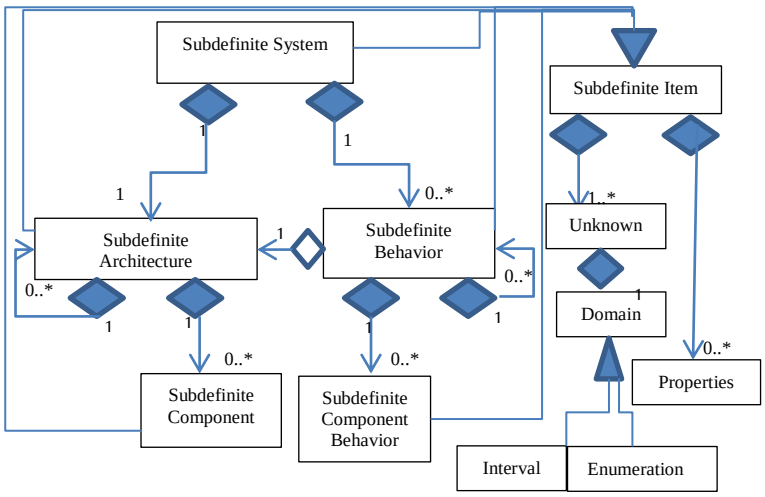

Fig. 5.Part of the MBSS meta-model

Modeling the problem requires the following capabilities within a dedicated formalism:

- Modeling a sub-definite architecture: the architecture unknowns, their possible domain of values, the structural properties to be satisfied necessarily by any solution architecture and the hierarchical decomposition of the sub-definite architecture.
- Modeling a set of sub-definite behaviors: the behavioral unknowns, their possible domain of values, the behavioral properties and the hierarchical decomposition of the behaviors.
- Sharing of the sub-definite architecture between several behaviors. Indeed, a solution of the problem will be an full instantiation of the architecture model satisfying the union of all the expressed behaviors.
- Setting requirements: in the form of structural and behavioral properties on the sub-definite architecture and behavior models.
- Because of the diversity of the design problems addressed (cf §2.1), the unknowns must be able to take their values in mixed domains: intervals of real numbers, intervals of integers, enumerations of real values, enumerations of integer values.
- The expressed properties must be able to be expressed in the form of linear and non-linear algebraic equations and inequalities but also in the form of specialized properties such as values to be taken in catalogs.

All these capabilities are represented in a UML format on the MBSS meta-model of figure 5. A sub-definite system is thus composed of a sub-definite architecture and a set of sub-definite behaviors and takes the subdefinite architecture as an argument. The sub-definite architecture can be decomposed in a tree-like manner down to the sub-definite components. The same reasoning applies to each sub-definite behavior. The system, behaviors and sub-definite architecture are composed of a set of unknown values inside domains and a set of properties to be satisfied.

The problem modeling aspects of the MBSS approach are available by using the DEPS language. DEPS (Design Problem Specification) is a declarative problem modeling language for system design problems [12]. The distribution and evolution of the language is now supported by the DEPS Link non-profit organization [13]. Its grammar is defined with a "context free" BNF (Backus-Naur-Form) notation. DEPS combines some structural modeling features of object-oriented and ontology description capabilities for engineers with problem specification features from constraint programming.. From the former, it has borrowed the characteristics of structuring and abstraction that allow us to represent the components and the (possibly partial) architecture of the system under study. From the latter, the possibility to describe ordinal and cardinal quantities used by the engineer (current, voltage, length, angle ...) were borrowed. From the third, the logical-mathematical concepts necessary to solve the engineer's problems were borrowed.

```
Model Camera()
Constants
Hc : HeightPix = 480;      v0 : PositionPix = Hc/2 ;
f : Distance = 0.0028;     tu : Height = 0.000015;
alpha : Real = f/tu;
Variables
theta : Angle in [0,1] ;          h : Height in [0.2,1] ;
expr v : PositionPix;             expr TotalCost : Cost;
expr TotalRel : Reliability;
Elements
ch1 : Channel();                  ch2 : Channel();
exclCh : OneOrTwoChannel(ch1, ch2);
Properties
TotalCost <= 200;      TotalCost := ch1.Cist + ch2.Cost;
TotalRel :=  ch1.Rel*ch2.Rel;
End
```

Fig. 6. DEPS model of the architecture of the camera

This combination of declarative paradigms leads to a declarative language that allows the designer to model the organization of the studied systems and the properties that govern them. DEPS addresses problems of sizing,

configuration, allocation and more generally of architecture generation or synthesis encountered in system design. The systems can be physical systems, software intensive or mixed systems (embedded or cyber-physical). The main feature of the language is the *Model*. A *Model* encapsulates constants, variables, instances of other *Models* called *Elements* and properties. Models have a signature and can be overloaded, extended, shared (aggregation) and composed (composition)
In the case of the problem of configuration and positioning of the camera, we describe in DEPS the architecture model of the camera (cf Fig. 6) consisting of one or two channel, a channel being composed of a sensor, a CPU and a transceiver. The latter are models derived from an abstract component model (cf Fig. 7).

```
Model Channel()
Constants
Variables
isIn : Boolean ;
expr Cost : Cost ;
expr Rel : Reliability;
Elements
s : Sensor();
t : Trans();
cpu : CPU();
Properties
Catalog([s.c1.Ref, t.c1.Ref, cpu.c1.Ref], CompatibilityTable);
Catalog([s.c2.Ref, t.c2.Ref, cpu.c2.Ref], CompatibilityTable);
Cost := IsIn*(s.Cost+t.Cost+cpu.Cost)
Rel := max(1-isIn, s.Rel*t.Rel*cpu.Rel);
End

Model Component() abstract
Constants
Variables
Ref : Reference;
Cost : Cost;
Rel : Reliability;
Elements
Properties
End

Model Sensor() extends Component[]
Constants
Variables
Elements
Properties
Catalog([Ref, Cost, Rel], Sensors);
End
```

Fig. 7. DEPS models of Channel, Component and Sensor

```
Model Obstacle(Cam,Y)
Constants
Y : Position; (*distance between obstacle and camera*)
Z : Height = 1.0; default ; (*default size *)
Variables
Elements
Cam : Camera[];
Properties
Cam.v := ((Cam.v0*cos(Cam.theta)+Cam.alpha*sin(Cam.theta))*Y+
(Cam.v0*sin(Cam.theta)-Cam.alpha*cos(Cam.theta))*(Z-
Cam.h))/(Y*cos(Cam.theta)+(Z-Cam.h)*sin(Cam.theta));
End

Model NearObstacle() extends Obstacle[Camera[], Position]
Constants
Z : Height = 0.22; redefine ; (* football balloon*)
Variables
Elements
Properties
Cam.v < Cam.Hc;
End

Model ObstacleLoin() extends Obstacle[Camera[], Position]
Constants
Z : Height = 1.8; redefine ; (* pedestrian *)
Variables
Elements
Properties
Cam.v > 50;
End
```

Fig. 8. Some DEPS behavioral models of the camera

The behavioral models of the sub-definite camera architecture will then be realized (cf Fig.8).
The architectural requirements will be set in terms of properties on the camera, while the behavioral requirements will be expressed in behavioral models.
Finally the problem will be posed by specifying that we create a sub-definite camera architecture and a set of behaviors to be satisfied (cf Fig. 9).

```
Problem CameraProblem
Constants
Variables
Elements
Camera : Camera();

Near1 : NearObstacle(Camera,0);
Near2 : NearObstacle(Camera, 1) ;
Near3 : NearObstacle(Camera, 2) ;

Far1 :  FarObstacle(Camera,3);
Far2 :  FarObstacle(Camera,4);
Far3 :  FarObstacle(Camera,5);
Far4 :  FarObstacle(Camera,6);
Far5 :  FarObstacle(Camera,7);
Properties
End
```

Fig. 9. DEPS problem modeling

4.3. Solving the problem

To solve the problems modeled in this way, we propose to use the framework of constraint programming (CP) on mixed domains. Each variable of the problem naturally takes its values in an expressed domain. Properties can be considered as constraints. A set of variables, domains and constraints is called a CSP (Constraint Satisfaction Problem)[18]. A CSP is declarative. The CP methods for solving CSP follow the set-based-design principle. The domains of the variables are iteratively reduced so that at any time the reduced domains contain all the solutions of the problem. It is therefore a guaranteed method of generating a solution that is correct-by-construction with respect to the constraints expressed [18-20]. If the solver does not find a solution, it guarantees that there is none. If the solver finds one or more solutions, it guarantees that they all satisfy the set of constraints expressed. It is also possible to minimize or maximize variable values.

DEPS Studio [12], the environment associated with the DEPS language includes a model editor, a project manager, a compiler and a solver. Experience shows that the first specification of a system design problem is never the right one and that many modeling errors are only detectable by calculation. DEPS Studio integrates a dedicated constraint programming solver on mixed domains so that the solution finding contributes effectively to the adjustment of the problem modeling process. This is a rapid model development approach (analogous to a RAD approach) which, contrary to a model transformation approach, reduces the execution time of the model development loop. It also allows errors to be traced back to the correct level of abstraction.

DEPS and DEPS Studio have been used for the design of electrical batteries [14], mechanical systems [15] and embedded systems such as integrated avionics [16] or aircraft

electrical generation and distribution system [17] under functional and safety requirements.
As for the problem of positioning and configuration of the embedded camera modeled in DEPS, it is solved in DEPS Studio in a few seconds on a Pentium CoreI5 type machine.

4.4. Related work

From the point of view of the MBSS approach as we have explained and formalized previously (cf fig 5), we have not found any equivalent work. Concerning the modeling formalisms, the authors of [21] underline the difficulties of using formalisms such as SysML (System Modeling Language), initially built for representing fully defined systems to model problems. They propose in their work to use as us a dedicated formalism called the Clafer associated with the Choco constraint programming library to model and solve a problem of allocating computers to embedded tasks. Nevertheless, Clafer remains a language dedicated to the configuration of software product lines with discrete variables and constraints. On the other hand, [11] proposes to add a first level of variability to SysML. The approach is coupled with the Choco constraint programming library to solve simple configuration problems. However, since these first research works, this initiative has not progressed significantly because it remains difficult to introduce more structural variability in a system definition language. The current limitations of the approach are the low level of variability that can be taken into account, the use of a solver dealing mainly with discrete constraints and a weak coupling between the formalism and the solver. This last point means that the debugging of the models is done, in case of bug or modeling problem, in the host language of the solver, in this case Java and not in the SysML language.

5. Conclusion

	MBSE	**MBSS**
Method	Point to point design Solution oriented Generate and test V cycle	Set based design Problem oriented Requirement based I cycle
Modeling	System modeling Defined System Local	Problem modeling Sub-definite system Global
Computing	Iterations Evaluation Simulation Optimization Verification and validation	Problem Solving Constraint Programming Correct by construction Satisfaction
Languages	UML, SysMl, AADL, DEVS, Modelica, …	DEPS
Tools	Capella, OpenModelica, MDO, …	DEPS Studio

Fig. 10. MBSE vs MBSS

In this paper we have proposed a possible evolution of MBSE summarized in figure 10. The MBSS approach supports the idea of modeling the problem rather than the solution candidate and then solving the problem instead of evaluating the performance of a solution candidate. The approach is tooling with the declarative problem modeling language DEPS and the DEPS Studio IDE for modeling and solving based on mixed domain constraint programming. The formalism allows the creation of generic, reusable and extensible sub-definite models. Modeling and solving are integrated in a single environment for easy model development.

The MBSS approach, the DEPS language and the DEPS Studio tool evolve according to the design problems addressed and the requirements considered.

References

[1] Singer DJ, Doerry N, Buckley ME. What is Set-Based Design?. Naval Engineering Journal 121(4): 31-43; October 2009.

[2] Bass L. Generate and test as a software architecture design approach. Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture; 2009.

[3] IEEE 1220-2005 - IEEE Standard for Application and Management of the Systems Engineering Process; 2005. https://standards.ieee.org/standard/1220-2005.html

[4] Graessler I, Hentze J and Bruckmann T. V-models for interdisciplinary systems engineering. proc of International Design Conference – ICED; 2018.

[5] Kahng A.B, DfX and Signoff: The Coming Challenges and Opportunities, Keynote Address, IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2012.

[6] AADL standard web site: https://www.sae.org/standards/content/as5506c/.

[7] Friedenthal S, Moore A, Steiner R. A practical guide to SysML. In: TheSystems Modeling Language, 3rd ed. The MK/OMG Press; 2015.

[8] Fritzson P. Principles of Object-Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach; Wiley Ed; 2014.

[9] Goedkoop M, Oele M, Leijting J, Ponsioen T, Meijer E. Introduction to LCA with Simapro; 2016. https://presustainability.com/files/2014/05/SimaPro8IntroductionToLCA.pdf

[10] Papageorgiou A, Ölvander J. The role of multidisciplinary design optimization (MDO) in the development process of complex engineering products**.** Proc of the 21st International Conference on Engineering Design (ICED 17); Vancouver; Canada; 2017.

[11] Leserf P, de Saqui-Sannes P, Hugues J, Trade-off analysis for sysml models using decision points and csps, *Software and Systems Modeling*,18(6):3265–3281 2019.

[12] Yvars PA, Zimmer L. Integration of Constraint Programming and Model-Based Approach for System Synthesis. IEEE International Systems Conference - SYSCON 2021; Vancouver Canada; 2019.

[13] DEPS language website. https://www.depslink.com

[14] Diampovesa S, Hubert A, Yvars PA. Designing Physical Systems through a Model-Based Synthesis Approach - Example of a Li-ion Battery for Electrical Vehicles. Computers In Industry Journal; vol 129; 2021.

[15] Yvars P.A, Zimmer L, A Model-based Synthesis approach to system design correct by construction under environmental impact requirements, Procedia CIRP, Vol 103, 2021.

[16] Zimmer L, Yvars PA, Lafaye M. Models of requirements for avionics architecture synthesis: safety, capacity and security. Complex System Design and Management conference – CSD&M 2020; Paris France; December 2020.

[17] Zimmer L, Yvars PA. Synthesis of software architecture for the control of embedded electrical generation and distribution system for aircraft under safety constraints: The case of simple failures. Proc of the 14th International Conference of Industrial Engineering - CIGI-QUALITA; Grenoble France; 2021.

[18] Tsang E. Foundations of Constraint Satisfaction. London and San Diego:Academic Press; 1993.

[19] Beldiceanu N, Carlsson M, Rampon J.X. Global Constraint Catalog. SICS Technical Report T2010:07; 2010. http://sofdem.github.io/gccat/gccat/titlepage.html

[20] Benhamou F, Goualard F, Granvilliers L, Puget JF. Revising Hull and Box consistency. Proc of the 16th International Conference on Logic Programming; 1993.

[21] S. Creff, J. Le Noir, E. Lenormand and S. Madelenat, "Towards facilities for modeling and synthesis of architec-tures for resource allocation problem in systems engineering", In Proc of 24th Systems and Software Product Line Conference, 2020.