Designing physical systems through a model-based synthesis approach. Example of a Li-ion battery for electrical vehicles

Sephora Diampovesa, Arnaud Hubert, Pierre-Alain Yvars

▶ **To cite this version:**

Sephora Diampovesa, Arnaud Hubert, Pierre-Alain Yvars. Designing physical systems through a model-based synthesis approach. Example of a Li-ion battery for electrical vehicles. Computers in Industry, 2021, 129, pp.103440. ⟨10.1016/j.compind.2021.103440⟩. ⟨hal-03716299⟩

HAL Id: hal-03716299

https://hal.science/hal-03716299v1

Submitted on 22 Mar 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons CC BY-NC 4.0 - Attribution - Non-commercial use - International License

Version of Record: https://www.sciencedirect.com/science/article/pii/S0166361521000476
Manuscript_6a96fd061ac2746e8dcd730075540741

Designing Physical Systems through a Model-Based Synthesis Approach. Example of a Li-ion Battery for Electrical Vehicles

Sephora Diampovesa, Arnaud Hubert
Universite de technologie de Compiegne,
CNRS, Roberval (Mechanics energy and electricity),
Centre de recherche Royallieu – CS 60319
60203 Compiègne Cedex, France
Email: sephora.diampovesa@utc.fr ; arnaud.hubert@utc.fr

Pierre-Alain Yvars
Laboratoire QUARTZ
Institut Supérieur de Mécanique de Paris (SupMeca)
3 rue Fernand Hainaut
93403 Saint-Ouen Cedex, France
Email: pierre-alain.yvars@supmeca.fr

ABSTRACT

This paper presents an approach for the preliminary design of complex physical systems in the framework of systems engineering. This approach is centered on the activities and actors involved in the pre-design tasks. It focuses on the modeling of design problems with DEPS (DEsign Problem Specification), a recent formalism for specifying and modeling design problems in engineering. This design methodology completes simulation-based analysis approaches, which are mainly used today for the design of physical systems. In particular, our approach allows the synthesis of pre-design architectures, which analysis/simulation approaches cannot do. Starting from the textual specification of the requirements, the proposed approach builds a formal model of the design problem and solves it using Constraint Programming. Ideas and concepts related to this approach are discussed: the issue of *reusability* of problem models, the concepts of *problem, knowledge and solutions spaces* as well as the formal specification of requirements and everything else that distinguishe a *design problem model* from a *designed system model*. An example of a Li-ion battery design for an electric vehicle acts as a practical use case for this article.

Keywords – Preliminary design, variability, electrical systems, subdefinite system, reusability, system synthesis.

1 Corresponding author

Arnaud Hubert
Université de technologie de Compiègne,
CNRS, Roberval (Mechanics energy and electricity),
Centre de recherche Royallieu – CS 60319
60203 Compiègne Cedex, France
Email: arnaud.hubert@utc.fr

2 Acknowledgment

The authors would like to thank the Hauts-de-France Region and the European Regional Development Fund (ERDF) 2014/2020 for the funding of the PhD grant relative to the project “HETSPEC”. The authors also extend their thanks to the DEPS Link Association.

© 2021 published by Elsevier. This manuscript is made available under the CC BY NC user license
https://creativecommons.org/licenses/by-nc/4.0/

1 Introduction

In engineering, the **design** of a system aims at defining one or a set of **admissible** architectures and components. Admissible architectures and components are those that satisfy all the design constraints specified in the **specification of requirements**. In this paper, the authors refer to the specification of requirements as the textual and formal document that provides the system requirements and the customer's needs in a clear, precise, objective and explicit manner, to which each stakeholder has agreed to [1, 2]. Optimal design also includes a criterion to be optimized. Hence, an optimal architecture is necessarily first admissible and minimizes (or maximizes) in addition one or several "cost" criteria. Optimality is usually a secondary concern that adds on top of admissibility, especially if the number of constraints to satisfy is very large.

The common activities in a design procedure are summarized in Figure 1 (adapted from [3, 4]). The design cycle processes and activities addressed by the presented work are those highlighted in red. They occur between the analysis of the requirements and the simulation activities required for detailed design or for the verification and validation phases ("evaluation"). This synthesis process covers what is commonly referred to as preliminary design. However, synthesis implies activities that are usually difficult and require a lot of expertise. Indeed, the result of synthesis ("expected properties") is to be evaluated and design decisions to be taken upon it, though requirements may still evolve and the system architecture still appears as a "draft" design. The main goal of this paper is to illustrate practically the use of new methods and tools to equip the synthesis process in design.

A design problem is most likely to be modified during the design process by changing, adding or removing **design constraints**. Some requirements may be missing or ill-posed (contradictory, leading to the absence of solutions). It is necessary to wonder if a design problem admits solutions. The answer depends first on the specification of requirements and on the designers' choice (ie the design constraints) and then, on the way the design problem is formalized. Modeling the design problem comes right after the production of the specification of requirements and it impacts consequently the after coming solving process.

At the begining of **preliminary design (pre-design)**, the system is said to be **subdefinite** [5, 6]. This means that its architecture is at least partially unknown and presents **variability**, ie **structural** unknowns of the design problem (**degrees of freedom, DoFs**). Conversely, the variables in the design models include not only these DoFs but also the **behavioural** variables (i.e. the output of direct models: speed, voltage, etc.) and structural criteria (deduced from the DoFs). Let us note that the modeling of **architectures variants** refers to the modeling of **definite** architectures and then differs from the

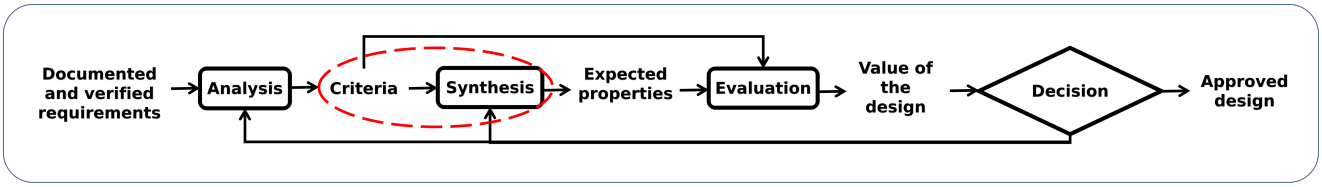


Figure 1. Activities of a design cycle. The red circle emphasizes activities adressed by this paper (adapted from [3, 4]).

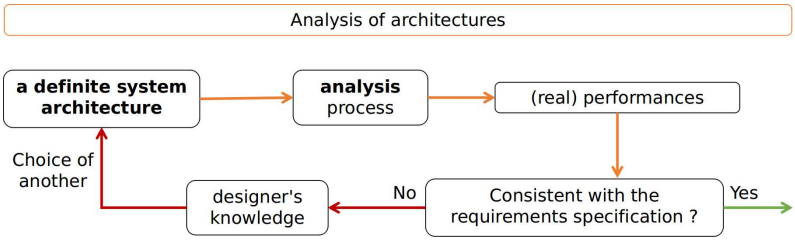
modeling of architectures with variability. The choice in the granularity of subdefinite system models is important. In pre-design and for "physical" applications (mechanical or electrical engineering), **coarse** models are usually chosen because they are more suitable. By coarse, we mean **algebraic** models (ie analytical or semi-analytical), such as surrogate models or equivalent circuit models. In most cases, the design process for complex systems in electrical or mechanical engineering is performed using an **analysis** approach as shown in Fig. 2 (a). This procedure is mainly based on **simulation** tools and iterative loops to improve or optimize the resulting solution [7, 8]. However, this approach has shows its limits when many combinatorial variables are present. A significant number of evaluations are needed and simulation times can become excessive. Moreover, this **simulation-based** procedure hardly manage or detect conflicting or contradictory requirements. In addition, solutions are not always robust to a small change in specifications. Regarding modeling languages, a distinction must be made between *system modeling languages* and *problem modeling languages*. As far as analysis and simulation are concerned, system modeling languages are well adapted because they allow to represent a fully definite system from several points of view (structural, behavioral, safety, life cycle assessment, etc). At this stage, it is necessary to emphasize that only a *known*, i.e. a *definite*, system can be simulated or analyzed. In systems engineering, designers usually rely on Model-Based System Engineering (MBSE) languages, either based on UML such as MARTE [9] and SysML [10] or AADL [11], EAST-ADL [12] or Modelica [13]. MARTE, as its name indicates, "Modeling and Analysing Real Time and Embedded system" is dedicated to the modeling and analysis of real-time fully defined systems. In the same way, AADL and EAST-ADL, as indicated by their surnames "Architecture Analysis and Design Language", are embedded system architecture description languages for analysis. All these languages are used to describe a *definite* system for the purpose of its analysis. They were made to model *systems* and not *design problems*, in connection with the development of simulation and analysis tools. In this case, designers must first select and model well-defined architectures before checking their admissibility with such tools. The methodology presented in this paper is different, it focuses on the synthesis of architectures (initially subdefinite) based on the modeling of the design problem for producing a *correct-by-construction* architecture [14]. Regarding the search for a solution, we are dealing with synthesis problems: the system is subdefinite and we need a formalism allowing to express this concern. In this perspective, some research works have recently coupled SysML with solvers or optimizers, but no integrated approach emerges from these works. [15] points out the difficulties of using formalisms such as SysML, initially designed to represent totally definite systems (SysML is a *System Modeling* Language), to model problems. As an alternative, they propose to use `Clafer` problem modeling formalism [16], associated with the Choco Constraint Programming library [17] to model and solve the problem of allocating calculators to embedded tasks. Nevertheless, `Clafer` remains a language dedicated to the configuration of software product lines. As another alternative, [18] proposes to add a first level of variability to SysML with the help of the Choco library to solve simple configuration problems. However, since these initial research works, dealing with synthesis problems using SysML has not progressed significantly. Such a design approach is currently limited by the following: the low level of variability that can be taken into account, the use of a solver handling essentially discrete constraints and a weak coupling between the formalism and the solver. The later means that the development of models is mainly carried out in the solver language rather than in SysML. These limitations have been pointed out in [19].

In the following, the authors present an integrated approach for modeling and solving design problems of complex physical systems during pre-design through the architecture synthesis of Fig. 2 (b). It relies on a recent formalism named `DEPS` (`DE`sign `P`roblem `S`pecification [20]) designed to specify design problems and to overcome previously identified limitations [21, 22]. The *practical* modeling with `DEPS` language and solving tasks are realized using an integrated environment called `DEPS Studio` [23]. The first part of this paper focuses on the presentation of this design methodology. The second part is devoted to its illustration on the design problem of a Li-ion battery for electric vehicles.

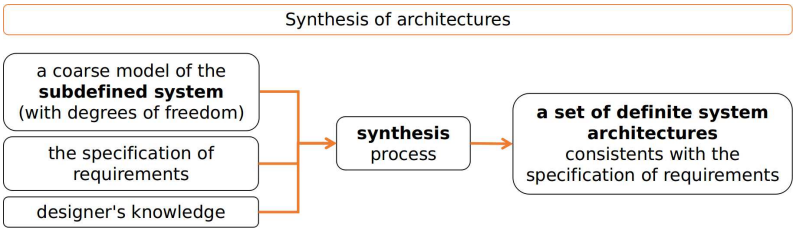
2 Designing through synthesis approach

When designing a new system, it seems natural to think that the architectures would be the ouputs of a process using the specification of requirements as inputs. This is the point of view called **synthesis of architectures** as shown in Fig. 2 (b). Because it presents two main advantages, this is a relevant alternative to simulation-based design:

1. The architecture of the system goes from subdefinite (with variability) to definite (totally known, without variability).
2. The existence of an admissible architecture is ensured at the soonest during the overall design project.



(a) Analysis of the system architectures.



(b) Synthesis of the system architectures.

Figure 2. Two complementary points of view on system architectures for the preliminary design of complex physical systems.

Let us note that model variables can be both structural (DoFs of design) and behavioral in a synthesis approach whereas structural variables must be fully defined in a simulation approach. The concepts of **problem space**, **knowledge space** and **solutions space** are meaningful in explaining the differences between modeling a system and modeling its **design problem** (see Fig. 3). This segmentation illustrates the difference between an approach based on synthesis (in the problem space) and an approach based on analysis and simulation (in the solution space) [24, 25, 16]. In the problem space, designers focus on defining and representing the design problem, that relies on the specification of requirements and on the support of experts from the relevant engineering disciplines (Knowledge space). Such a representation does not attempt to describe a specific solution. The problem of designing a physical system architecture is always very complex from the structural, the mathematical as well as the algorithmic points of view. Therefore, the most impacting decisions must be made as early as possible, ensuring that they will lead to acceptable final solutions. By using the Problem space, the Knowledge space and Constraint Programming, the design approach proposed in this article shows that it is possible to control this complexity by modeling the variability and by eliminating as quickly as possible the irrelevant design choices which will not ensure the admissibility of solutions. The synthesis process thus generates one or more admissible solutions as soon as possible.

2.1 Relation to the ISO/IEEE 42020 design standard

Systems are currently designed according to the IEEE 1220 standard [26]. This standard specifies a design process for software-intensive systems engineering. Nevertheless, it is also used for designing physical systems. It mainly recommands the use of analysis tools for designing new products. As a result, the development of tools for synthesis have been put aside for the benefit of analysis. However, limitations have been more and more pointed out by design stakeholders, specially regarding the expression of variability. The ever-increasing complexity of new systems necessitate to rethink their design approaches, particularly during pre-design. These observations have led to the very recent ISO/IEEE 42020 standard [27], offering a design process more focused on **architecture conceptualization**, ie the synthesis of architectures. Besides, this standard also insists on the concepts of problem and solutions spaces. While very actual, this new design standard is not really employed as modeling and solving tools are currently lacking.

2.2 A Design approach for the synthesis of architectures

The pre-design methodology proposes by the authors is summarized in Fig. 4. It consists in two successive steps and aims at offering a general manner to design a system architecture during pre-design, by relying mainly on the **synthesis of architectures**. Regarding the ISO/IEEE 42020 standard, this approach covers the "Architecture conceptualization" and the "Architecture elaboration". Step 1 proposes the formal modeling of a design problem using `DEPS`. Step 2 focuses on solving this problem in `DEPS Studio`.

Step 1 begins with the textual specification of requirements expressing the design problem in a natural language. This specification is the result of a Trade-off between various stakeholders, including the client, the designer, engineering discipline experts as well as the project manager. At this stage, the Knowledge space helps in choosing a coarse model of the subdefinite system to design (resulting from discussions between expert and designer). During the problem modeling, this space helps in defining the design variables and their domains (as a result of technological constraints for example).

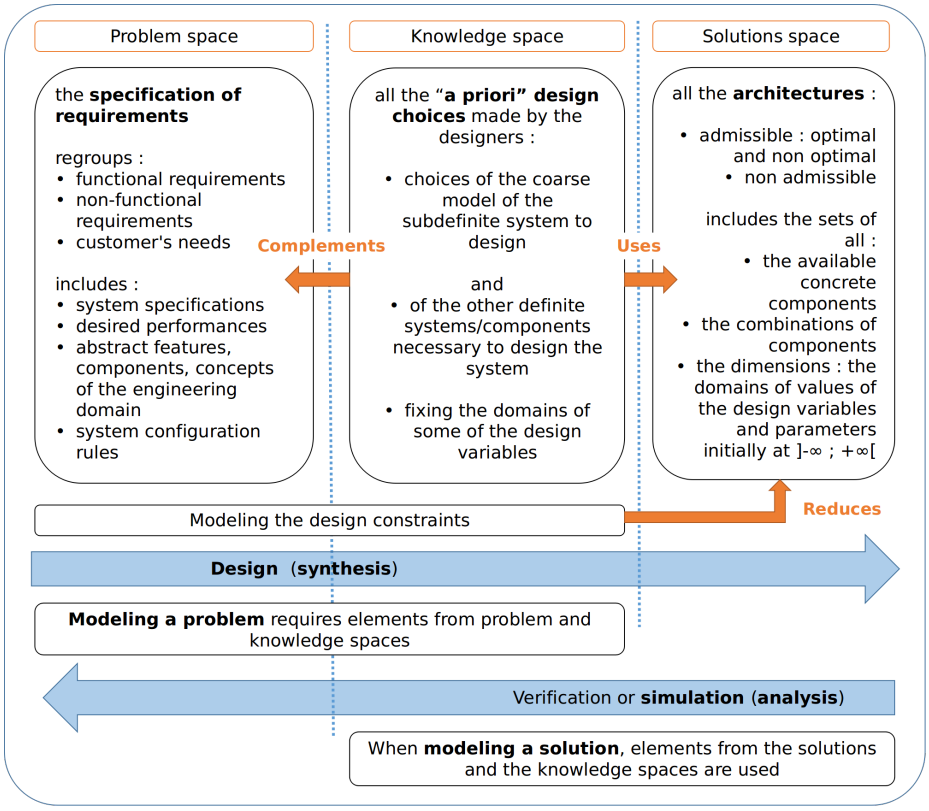


Figure 3. The three subspaces of design space and their interactions (Design, Knowledge and Solution spaces).

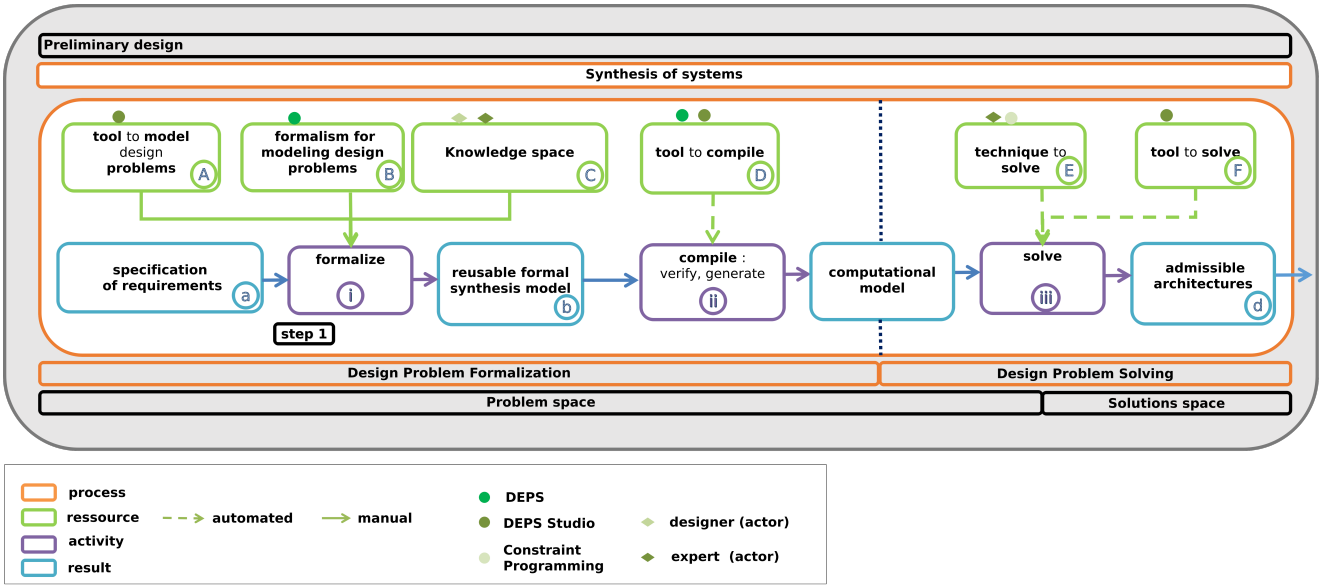


Figure 4. An approach for pre-design through the synthesis of architectures.

Formal models are constructed, with the aim to ease the understanding of the problem. Then, these models are verified by the compiler of `DEPS Studio`. During step 2, a generic computational model is generated for solving. The set of all admissible architectures is synthetised with the built-in solver of `DEPS Studio`. An example of a design is further discussed to illustrate the relevance of the proposed design methodology.

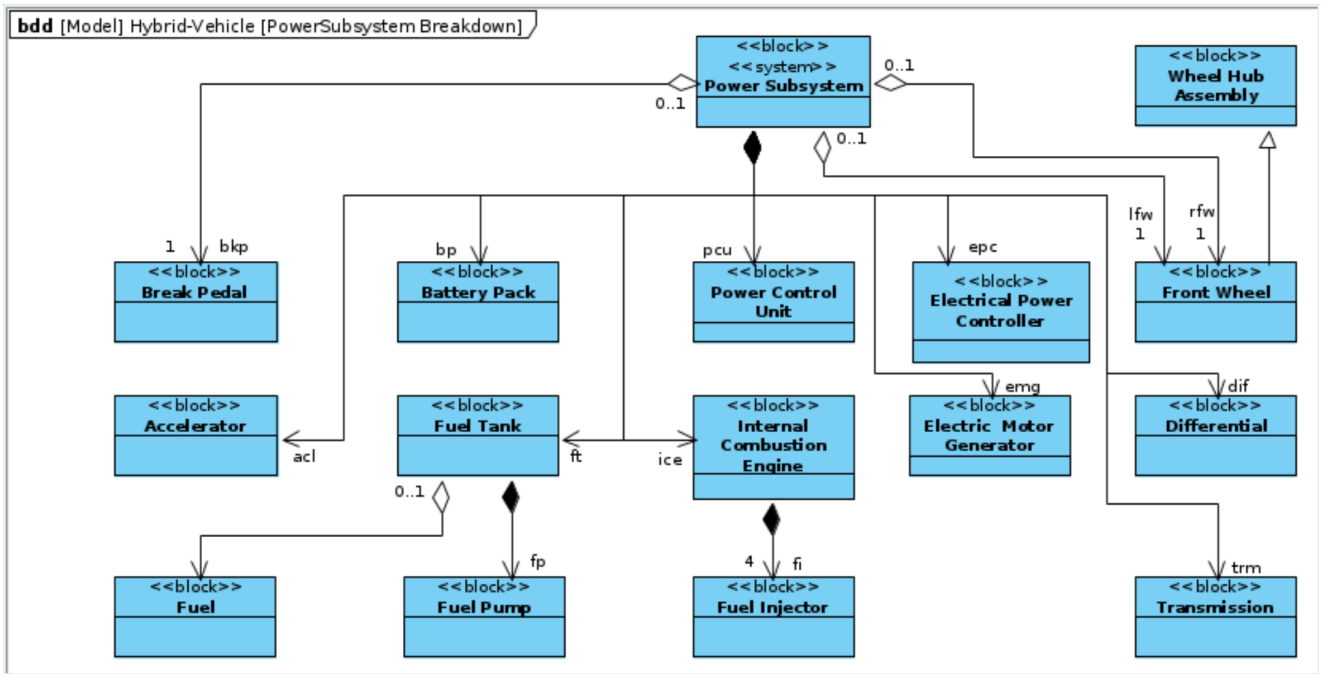


Figure 5. Example of structure for a hybrid vehicle power subsystem (SysML *Block Definition Diagram*).

3 The study case: a Li-ion battery for electrical vehicles

Energy management is and will remain one of the major challenges of the 21st century, both in terms of reducing consumption and the use of renewable energies. These scientific issues lead to the design of increasingly complex systems to reduce energy consumption, pollution and CO2 emissions. The increase in complexity often comes from the need to alternate between different technologies according to operating conditions, regulation/standards or cost constraints. The transportation industry is directly impacted by these changes, which can be seen, for example, in the growth of electric or hybrid vehicles [28]. As an illustration, the SysML block definition diagram of Figure 5 shows the structure of an hybrid vehicle power subsystem. It will be noted that the need to simultaneously manage different power sources leads to a significant increase in the structural complexity of the systems to be designed. In order not to overload this article, only a sub-part (the battery pack) of such an assembly will be considered subsequently. Adopting an object-oriented approach in our design methodology has proven to facilitate the re-usability of the developed models. As a result, it will always be possible to re-use these sub-models to design a more complex system (power sub-system or complete vehicle). The problem of designing a Li-ion battery for electric vehicle traction seems relevant and of sufficient complexity to show the current or future challenges in designing complex systems without obstructing the reader with unnecessary technological implementations details. The choice of this example illustrates the typical elements of a pre-design problem, and key mathematical difficulties (i.e. mixed types, non-convex and nonlinear, selection of components from manufacturers' catalogs).

Although the design of batteries for electric vehicles is a hot topic in research and development, the vast majority of current research focuses on material, technology and detailed modeling for simulation [29, 30, 31]. The main objectives of the research conducted in this area are to propose alternative solutions and technologies that are more efficient than current systems for the storage of electrical energy in vehicles. However, as explained previously, such detailed models for analysis and simulations are poorly adapted to the pre-design of complex systems. Thus, battery design problems for complex systems remain a challenge for manufacturers who strive to provide powerful, hybrid or all-electric vehicles. Indeed, the performances of rechargeable batteries are limited in three aspects: their size, their mass and their high cost. Industrials have to meet many contradictory goals during the overall design process, such as providing functional and safe systems while reducing masses, costs, and considering the recyclability of products. Hence, to help finding new or more suited architectures of systems, the first steps of design are decisive. Complexity manifests itself in many forms, as mentioned in [32]. Hence, it is necessary to specify which ones are dealt with in our framework. Is it the complexity of the system ? Nothing is less certain. Actually this paper focuses more on the *complexity of the synthesis problem* than of the *complexity of the system*. Thus, the first can be divided into two interacting axes:

1. The *complexity of the problem modeling*. Indeed, very often it is already difficult to express the problem. Concerning the problem of configurating/sizing a battery, the complexity of modeling lies in the fact that: first, the variables are mixed, as some components (the cells) are to be chosen in manufacturers' catalogs and second, the battery is a subdefinite

system subject to a load profile. Thus, this load profile must be modelled as well as its relations with the subdefinite model of the battery. We therefore need a formalism capable of capturing this modeling complexity in order to generate one or more admissible architectures. Our goal is not to build an analysis or simulation model (which would then be the model of *a* solution) but to generate a *model of the design problem* and to solve it. The use of reusable (object-oriented) models to construct templates simplifies this part for other similar designs.

2. The *Resolution complexity*, also called computational complexity. Since we use Constraint Programming techniques, this complexity is also much related to the model of the problem. Decreasing the complexity of resolution is generally related to the modeling of the problem: it is by minimizing the number of unknowns and maximizing the number of constraints that we generally obtain a decrease in computation time. The modeling language must therefore include a mechanism capable of identifying the true unknowns among all the parameters used in the problem model. Careful choices of heuristics and algorithms that efficiently manage global constraints also help to reduce computation time.

3.1 Positioning the design problem: a reflexion/analysis in Problem and Knowledge spaces

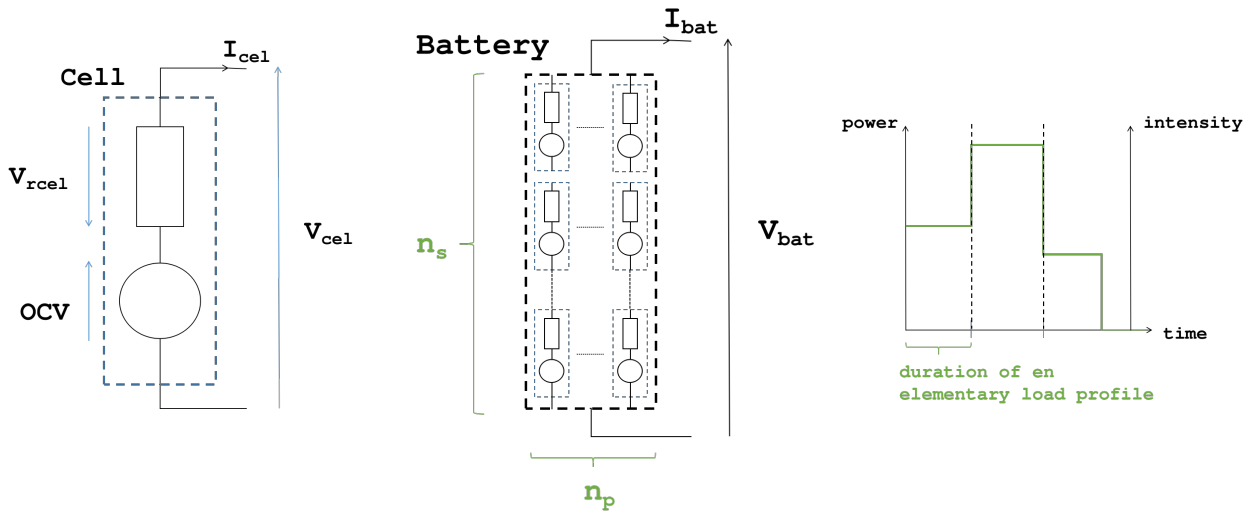
At first, a design problem shall be formulated. The positioning of the design problem requires a reflection/analysis between the Problem space and the Knowledge space. It must state the system requirements and customer's needs. The mathematical relationships describing the subdefinite battery and the other definite systems shall be coarse. [33] presents the design problem of a Li-ion battery for the traction of an electrical vehicle. In this reference, the electric vehicle is composed of a chassis and an electro-magneto-mechanical power drive. This power drive consists of a Li-ion battery, a power converter, an electric machine and a reduction gear. However, as said before, in this article we will only focus on the design of the battery. The battery is the sole power source of the vehicle. It is an association of $ns \times np$ serial and parallel cells. Each cell forms an energy storage. The goal in designing a battery is to find: 1) the type of cells, 2) the number of cells in serie, 3) the number of cells in parallel, such that the battery can supply the energy necessary for driving the vehicle. An admissible architecture of the battery is defined when these 3 variables are defined. Thus, this example is a 3 DoFs design problem. This design correspond to a configuration problem which is particularly interesting in electrical engineering, as not common in scientific literature. Moreover the problem contains **mixed-type** variables because the 3 DoFs of the design are discrete variables (integer $\in \mathbb{D}_{\text{DOF}} \subset \mathbb{N}$) but all other variables are continuous (real $\in \mathbb{D}_{\text{working}} \subset \mathbb{R}$).

3.2 Knowledge space: a space to help the "A priori" design choices

A cell is characterized by a type, a nominal capacity, an internal resistance and a mass. Three types of cells are available in a catalogue (type 0, 1 and 2). All the other real variables, such as the battery current or mass, are deduced from these 3 DoFs. Mixed-type variables problems are common in design but still pose many problems in terms of resolution. Discrete and continuous design problems are well-handled separately, however, the combinatorial aspect added to a continuous problem makes it harder to solve. In this paper, some assumptions are made for the modelling of the battery. These assumptions are relevant as far as pre-design is concerned. In pre-design, electrical engineers usually rely on an equivalent electric circuit to model a battery (Fig. 6 a and b). Each cell is composed of an ideal voltage source (which voltage is commonly called **open circuit voltage** - ***OCV***) and an internal serial resistor (which voltage is refered here as V_{rcel}), in order to reflect the real voltage source (which voltage is commonly called "terminal voltage" - refered here as V_{cel}). Furthermore, all cells are viewed as identical regarding their discharge (*Rint model*, [34]). The identical behavior of the cells enables to view the battery as a uniform cell close to a factor. This factor is linked to ns and np. Also, the cell OCV, V_{cel} and V_{rcel} are linked by linear relationships. Thus, the battery is represented by an equivalent ideal voltage source with an equivalent serial resistor.

The design problem also includes a **load profile** to be powered. The load is composed of the entire vehicle except for the battery. The load profile is the energy demand necessary to move the vehicle. This energy must be supplied by the battery. The battery copper losses can be viewed as an extra energy added to the load profile, that the battery has to supply. This load profile imposes the battery discharge current. This instantaneous discharge current varies over time and is provided during the vehicle movement. The load profile is a definite system. The electrical model of the load is also simplified, with some strong assumptions made. The load profile, that is continous, is discretized into several elementary load profiles. An elementary load profile is defined by a constant power required during a given duration (Fig. 6 c). Indeed, in pre-design, the models of the systems shall represents the behavior of the system **quasi-statically**, ie either in steady state or at maximal operation. Thus, algebraic relationships are kept by avoiding integral sums and instantaneous time dependencies. Time dependency is implicitly eluded. The current is also discretized. The current of an elementary load profile, and thus the battery discharge current, is considered constant during the elementary duration.

As a result, the **state of charge (*SOC*)** of the battery can be estimated. The *SOC* is an important parameter for the battery. It enables to quantify the remaining level of energy of the battery during the vehicle movement, and thus indicates if the battery can supply the required energy. The battery *SOC* is also an equivalent model, considering that each cell has the same *SOC* during a given duration. The *SOC* of a cell is expressed as a percentage of the cell nominal capacity. The *SOC* is linearly linked to the *OCV*. The values of the *OCV* depends on the cell technology. For a Li-ion cell, the minimum *OCV* is 2.7V and the maximum *OCV* is 4.2V. A *SOC* of 1 (100% of the nominal capacity) means that the cell *OCV* is 4.2V. However,



(a) A cell modeled as an electrical circuit. (b) A battery constituted with $ns \times np$ cells. (c) Load profil used as requirement of design.

Figure 6. Elements of modeling for pre-design chosen in the Knowledge space: (a) the coarse model of a cell (R_{int} model), (b) the coarse model of a battery as an interconnection of $ns \times np$ cells, (c) the load profil discretized into 3 elementary load profiles.

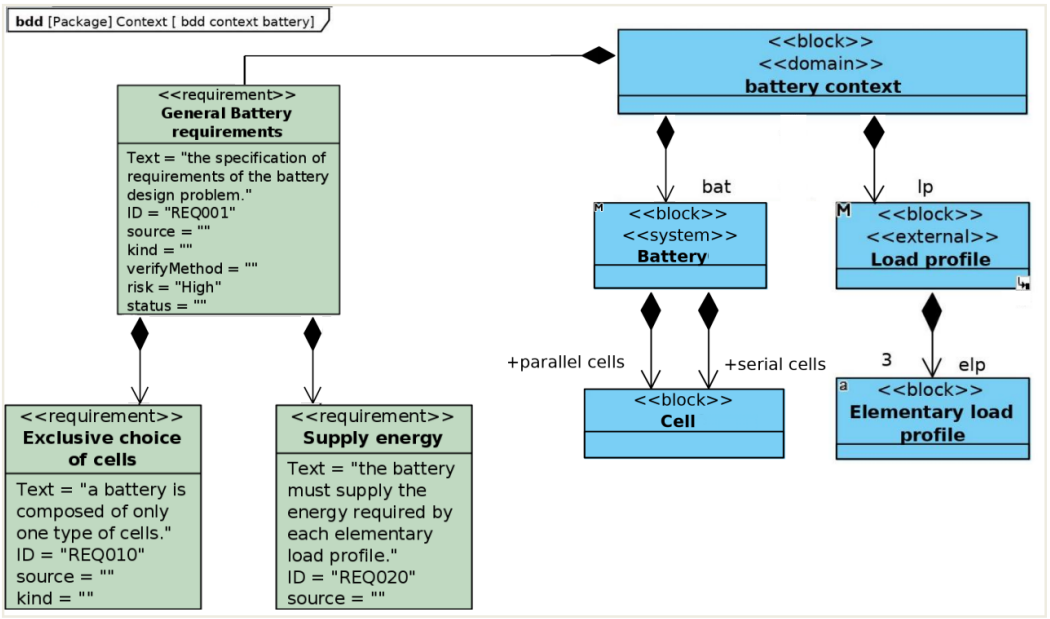


Figure 7. Decomposition of the battery design problem (SySML *Block Definition Diagram*).

the minimum value of the cell *SOC* is settled at 0.2 (20%) and corresponds to the minimum *OCV* value of 2.7V. Indeed, an important design constraint imposes the battery to keep a *SOC* above or equal to 20%. If the cell voltage drops below 2.7V, it may be destructive for the cell. This design constraint has to be fulfilled for each energy demand, ie at the beginning, during and at the end of every elementary load profile. Between 20% and 100% of charge, we assume variations of internal resistance as negligeable. Since the relationships between a cell and a battery are linear, the *SOC* of the battery is the *SOC* of one cell. The elements of the design problems are summarized in Fig. 7 using a SySML *Block Definition Diagram*.

With all these assumptions, the solution to the design problem is to find all the admissible battery architectures, so that at each elementary load profile, the power that the battery must supply is greater than or equal to the power required by the load plus the copper losses of the battery.

Table 1: Catalog of available cells.

Type	**Capacity C_3 (A.h)**	**Mass (kg)**	**Internal resistance (m Ω)**	**OCV (V)**
0	39	1.05	7	2.7 - 4.2
1	25	0.75	16.75	2.7 - 4.2
2	16	0.68	23	2.7 - 4.2

3.3 Problem space: a space to specify formally the design requirements

In examining this design problem from the Knowledge space, the authors suggest dealing with the following specification of requirements as an exemple composed of 12 requirements:

R1 : the battery must supply the energy required by the load profile (driving mission).
R2 : the battery copper losses must be compensated as an additional load.
R3 : the battery state of charge must never drop below 20%.
R4 : the power/energy profile of the load is as following:
the driving mission lasts 30 min (1800 s).
Three elementary load profiles are defined (operating points): 10831 W@22.8 A, 45210 W@102 A et 740 W@50 A, each one during 600 s.
R5 : three types of cells are available in a catalog: 0, 1, 2. Each one is characterized by a type, a nominal capacity, an internal resistance and a mass (cf. Tab. 1).
R6 : a battery is composed of a unique type of cells.
R7 : the maximal number of serial cells is set to 5000.
R8 : the maximal number of parallel cells is set to 5000.
R9 : the battery mass cannot excede 800 kg.
R10 : the maximal cell discharge current is limited to 40 A to protect the cells.
R11 : the useful battery energy cannot excede 100 kWh.
R12 : the minimum battery efficiency is 90%.

4 Formal modeling of the design problem with DEPS

The objective of this section is to model the problem of battery design using a model-based approach that can be solved (solving methods must be addressed at the same time as problem modeling). This resolution will then lead to the generation of one or more admissible solutions, i.e. satisfying all the design requirements. To model and solve design problems in "physical systems" engineering, the use of *Constraint Satisfaction Problem* (CSP) has proven to be very useful (see for example [35, 36] in mechanical engineering, [37] in electrical engineering or [38] in microelectronics). A CSP is defined by a triplet (X,D,C) such that [39, 40]:

$X = \{x_1, x_2, x_3, \cdots, x_n\}$ is a finite set of *n constraint variables*, with $n \in \mathbb{N}$;
$D = \{d_1, d_2, d_3, \cdots, d_n\}$ is a finite set of the *domains* of the constraint variables such as the domain of x_i is d_i, $\forall i \in \{1, \cdots, n\}$. d_i forms the set of possibilities for x_i.
The domains can either be discrete, with a closed and bounded set of possible values $\{x^1, \cdots, x^l\}$. $l \in \mathbb{N}$; or continuous, as an interval of reals $[x^{\min}, x^{\max}]$, with $x^{\min}$ and $x^{\max}$ two *finite* reals.
$C = \{c_1, c_2, c_3, \cdots, c_p\}$ is a finite set of *constraints* to be satisfied by the constraint variables, with $p \in \mathbb{N}$, i.e. $\forall j \in \{1, \cdots, p\}, \exists! X_i \subseteq X / c_j(X_i)$.

A constraint is any type of mathematical relations (linear, quadratic, non-linear, boolean,...) involving at least one variable. It can be logical, explicit, etc. The constraints can be algebraic equations and inequalities or even global constraints [41]. Thus representing the battery design problem as a CSP would be like identifying : (i) the set of variables of the problem, (ii) the set of possible value domains for each variable and, (iii) all the constraints of the problem. However, it would be a low-level representation of the problem without the possibility to explicitly express the structure of the subdefinite physical system (i.e. the engineer's physical quantities, the relations between subsystems, etc). If we use the analogy with computer programming, the model (X,D,C) is a very low-level design model like the "assembler" in computer science. For the design of complex systems, we actually need a problem representation language capable of capturing much more than just variables, domains and constraints. In particular, this language must be able to model the structure of the problem and to manipulate

ordinal and *cardinal* quantities, that are essential for engineers and experts. To facilitate the development and the use of CSP, the scientic community has developed a number of available libraries such as Choco [17], IBEX [42] or RealPaver [43]. Those libraries are built using object-oriented programming languages as C++ or Java. However, these are *programming languages* (dedicated to programmers) and not *modeling language* (dedicated to designers and engineering experts). Our purpose is therefore to adopt a high-level modeling language and an integrated environment which can on the one hand model and solve the design problem but which is also specially dedicated to designers. For the resolution part, this language will use Constraint Programming on mixed domains. To our knowledge, the only two solutions that currently exist for this purpose are `Clafer` [16] and `DEPS` [21]. However, `Clafer` has two limits:

1. `Clafer`'s solver, i.e. Choco, is not integrated into the language. It is necessary to transform `Clafer` models to Choco, which makes it extremely difficult to develop `Clafer` models. Indeed, if results produced by Choco are not satisfactory (solving problems), models modifications can only be done in Java and not in the initial `Clafer` modeling language.
2. Choco, is designed to handle domains of discrete variables while design problems often involve mixed variables. For example, the battery design problem used to illustrate this article essentially involves real continuous variables (physical quantities).

Thus, `DEPS` is a language better suited for modeling design problems because it does not have these two limitations. The `DEPS` research project aims at developing a textual formalism to specify design problems, `DEPS`, and a dedicated framework to model and solve them in the same environment `DEPS Studio`. `DEPS` is a **Domain-Specific Modeling Language** (**DSML**) dedicated to the **formalization of design problems** of systems (including embedded, real-time, electrical, cyber-physical systems). `DEPS` is intended for pre-design and aspires to deal with sizing, configuration, architecture generation and allocation problems, that can be mixed-type and ill-posed (including over-constrained). `DEPS` is a formal, declarative, object-oriented language. Contrary to some relatively theoretical formalisms, such as *Axiomatic design* [44] or *CK-theory* [45], the `DEPS` formalism is the result of an inductive approach carried out in partnership between industrial and academic researchers. Advances around `DEPS` started with very concrete industrial questions and, thanks to an analysis of the limits and strengths of the tools and languages currently available, led to a first proposal for a language and an associated IT tool (`DEPS Studio`). The results of this work, far from being definitive, are in constant progress, according to the new needs in the face of new design problems. Dedicated to designers in industrial design departments, the approach, the language and the implementation tool are designed in a pragmatic way so as not to be rejected by potential users. Previous versions of `DEPS` have already been employed to formalize continuous (in the field of robotics [21]) and discrete (in the field of integrated avionics [22]) design problems, but many more different examples must be treated to see advantages and limits of problems modelling with `DEPS`. Its use in the context of numerous examples of designs with different characteristics is also a means of proposing evolutions of the language. To date, several upgrade versions have been offered in just a few years. Especially, the battery design example was chosen in this submission because it addresses features of the language that had not been explored before:

1. Mixed-type design problems (impact on solver and resolution),
2. Applications in electrical engineering (impact on the Knowledge space and design constraints),
3. Design problems with constraints tables (manipulation et implementation of catalogs of components).

`DEPS` is a model-oriented language that supports inheritance, aggregation (shared aggregation), composition (composite aggregation) and polymorphism. These characteristics enhance the **reusability** of models of objects. Each model can be characterized by a set of properties that each instance of this type of model must necessarily satisfy.

4.1 Modeling in DEPS

In `DEPS` modeling, a model of object is refered by the keyword `Model` (i.e. *systems*), and `Problem` (i.e. *design problems*). A model is constructed as following: the keyword `Model` is followed by its name and the list of its arguments in brackets (this list can be empty). Arguments can be constants or instances of other models. Parameterized models can thus be constructed. Semantically, a problem do not take any arguments. If so, this means that the problem is in fact a model. In `DEPS`, constants are separated from the variables. Constants are declared and defined in the `Constants` section. Declarations and definitions end with a semicolon. Variables (DoFs) are declared in the `Variables` section. This section regroups both the design variables and the deduced variables or **named expressions**. Indeed, the deduced variables are expressions to which a name is given. To differentiate a design variable from a named expression, a named expression is declared with the keyword `expr`. Instances of models can be declared and created in the `Elements` section. The elements are all instances of other models that were priorly created.

At last, relevant design constraints can be defined in the `Properties` section as algebraic relationships: equalities, inequalities, affectation (definition of the named expressions) and properties as extension. Properties as extension (extended properties) express for example the compatibility constraint between some design variables and a tuple of possible values. The keyword `End` specifies the end of a model.

```
1  Quantity INTENSITY
2  Kind : ECINTENSITY ;
3  Min  : -maxreal ;
4  Max  : +maxreal ;
5  Unit : A ;
6  End
```

```
8  QuantityKind ECINTENSITY
9  Type : real ;
10 Min  : -maxreal ;
11 Max  : +maxreal ;
12 Dim  : I ;
13 End
```

Figure 8. How to declare a model of Quantity and QuantityKind in DEPS.

```
14 Model Pin()
15 Constants
16 Variables
17 expr i : INTENSITY ;
18 phi : ELECTRICPOTENTIAL ;
19 Elements
20 Properties
21 End

23 Model Connect(b1, b2)
```

```
24 Constants
25 Variables
26 Elements
27 b1 : Pin ; (* system 1 terminal *)
28 b2 : Pin ; (* system 2 terminal *)
29 Properties
30 b1.phi = b2.phi ; (* compatibility of potentials *)
31 b1.i + b2.i = 0 ; (* continuity of flow *)
32 End
```

Figure 9. DEPS models. Left: A model of electric terminal in DEPS ("Pin"), Right: Modeling the interconnection between two systems.

DEPS is a typed language: constants as variables must be declared with a type of **quantity** (physical, technological, ordinal or cardinal). Variables and constants are both declared as: `name : Quantity ;`. *Constant* `name` can be defined by a value: `name : Quantity = value ;`. *Variable* `name`, i.e. a DoF in the design problem, can be defined by a domain of values: `name : Quantity in [domain-of-values] ;`. The structure of a `Quantity` model in DEPS is shown on Fig. 8 for `INTENSITY`, an electric current. This is a model specified by a `Kind`, a `Min` value, a `Max` value and a `Unit`. `Kind` keyword refers to an existing `Quantitykind`, that sets the dimension of the quantity. For this example, `Quantitykind` model is specified by a basic type (`real`), a `Min` value, a `Max` value and a `Dim`. Types can be either predefined or user-defined. For both `Quantity` and `Quantitykind`, the `Min` (respectively `Max`) keyword specify the minimum (resp. maximal) value of this type of constants or variables. The `Unit` keyword specifies a unit for the quantity. If the quantity does not have units, this is specified as `u`. The `Dim` keyword specifies the dimension of the quantity (as for dimensional analysis). If the quantity does not have dimension, this is specified as `U`.

4.2 Models construction

There are many ways to model a design problem, all the more if the design also aims at enhancing the reusability of models. This secondary goal needs to be considered right from the start of modeling. Indeed, the more reusable the models become, the higher the level of difficulty becomes. As with any object-oriented modeling, a rigorous preliminary domain analysis must be conducted from the Knowledge space to determine the *correct* objects and the *correct* structure of the design problem. Iterations on this step are sometimes, and often, necessary. In this sense, problem modeling is constructive and inductive. In addition, designers need to think about the level of information to reveal to the users of the models. Users should not be aware of the specific implementation details, for models to be easier to understand and to use. This property is known as "encapsulation" in object-oriented modeling.

4.2.1 Models and reuseability

Electric terminals of interconnected multipoles are features that can be modeled to enhance the **reuseability** of models of physical objects or components. This feature is, for example, used in component- or object-oriented simulation languages (Spice, VHDL, Verilog, Modelica). This can be easily implemented in DEPS as shown in the `Pin` models Fig. 9 left. Thus, the modeling approach based on the two **Generalized Kirchhoff Laws** (as in **Energy** and **Networks methods** [46]) can be used in DEPS. These two laws can be defined in a model as on Fig. 9 right - lines 30-31. Similar models can be constructed to connect more systems together. In DEPS, comments are defined for both lines and sections inside the `(*` and `*)` delimiters.

This way of modeling physical systems allows an **acausal** definition of the constitutive laws for every interconnected physical system. Using a representation based on dual electrical variables (current, voltage) to describe interconnections between physical components also makes it possible to automatically satisfy energy conservation laws [46]. This representation allows to model the power through a system as a flow that can flow in and out. The system behavior is acausal from a physical point of view. Indeed, until the battery is connected to the other systems, its behavior is unknown. If the battery is connected to a load, the power flows out. If the battery is connected to a charger device, the power flows in. This unique model of the battery can be used for both discharge and charge instead of specifying a model for each operating mode.

```
34 Model Battery()
35 Constants
36 Variables
37 ns : INTEGER in [1,10000] ;(* number of serial cells
       *)
38 np : INTEGER in [1,10000] ;(* number of parallel
       cells *)
39 expr ibat : INTENSITY ;(* current *)
40 expr ocvbat : VOLTAGE ;(* ocv *)
41 expr vbat : VOLTAGE ;(* terminal voltage *)
42 expr rbat : RESISTANCE ;(* internal resistance *)
43 expr mbat : MASS ;(* battery mass *)
44 expr energy : ENERGY ; (* available energy *)
45 (...)
46 Elements
47 n : Pin() ;
48 p : Pin() ;
49 cell : LiIonCell() ;
50 Properties
51 ...
52 End
```

```
54 Model LiIonCell()
55 Constants
56 peukert : REAL = 1.1 ; (* Peukert coefficient*)
57 Variables
58 cellType : INDEX ; (* type of cell *)
59 r0 : RESISTANCE ;  (* static resistance *)
60 c3 : CHARGECAPACITY ; (* nominal capacity *)
61 mcel : MASS ; (* mass *)
62 icel : INTENSITY ; (* current through a cell *)
63 expr vcel : VOLTAGE ; (* cell voltage *)
64 ...
65 Elements
66 e0 : VoltageSource(2.7, 4.2) ; (* ideal DC voltage
       source *)
67 rcel : Resistor() ; (* resistor *)
68 n : Pin() ; (* negative pin *)
69 p : Pin() ; (* positive pin *)
70 connectER : Connect(e0.p, rcel.p) ;
71 Properties
72 Catalog([cellType, r0, c3, mcel], cellTable) ;
73 ...
74 End
```

Figure 10. DEPS Models. Left: Modeling a subdefinite battery, Right: Modeling a subdefinite Li-ion cell.

4.2.2 Models for definite and subdefinite systems

The content of the `Variables` section of a `DEPS` model allows to distinguish the model of a definite system from the model of a subdefinite one. The model of a definite system do not contain design variables, i.e. structural DoFs. On the other hand, the model of subdefinite systems always contains variables in the `Variables` section. Figure 10 left partially shows a subdefinite battery model with two DoFs: `ns` and `np`. These two design variables are declared inside this model. Their domains of values are also chosen as part of designers' knowleges. Minimum values are both set to "1" for realistic purposes, and maximal values to a number small enough to reduce the solutions space, but high enough not to create overconstraints. The named expressions (e.g. `ibat`) are then declared. A battery is viewed as a system composed of a negative terminal `n` and a positive one `p`. It is modeled as the composition of a `cell` (line 49) which is an instance of the subdefinite `LiIonCell` model. A part of `LiIonCell` is shown on Fig. 10 right.

4.2.3 Modeling a configuration problem in DEPS

One of the difficulties of configuration problems is that components to choose are defined by tuples: in this example, cells are defined by the following tuple {type, mass, nominal capacity, internal resistance}. Thus, if one of its attributes is defined during solving, a cell is implicitly chosen and all the other attributes are defined as well. As a result, the solutions space is reduced into particular sets of values (e.g. {0, 1.05, 39, 0.007} for cells of type 0). This kind of design constraint is called a **catalog constraint** (as the component is chosen inside a catalog). A catalog constraint is a **global constraint** like, for example, the "*All different*" constraint [41]. In `DEPS`, the configuration problem associated with the choice of a particular cell is therefore modeled in two stages.

1. First, the table of available cells is modeled. In `DEPS`, catalogs are modeled throught the concept of `Table` (Fig. 11 left). A Table is a object that contains only structural data without properties. The four attributes of a cell are declared in the `Attributes` section. They are defined for each cell as `Tuples` separated by coma. Any number of cells can be specified (only three are given in this example).
2. Then, the catalog constraint is modeled inside the model `LionCell`. First, attributes of tables are declared as design variables in the `Variables` section. In the `Elements` section, the instances of `e0`, `rcel`, `p` and `n` (resp. instance of the `VoltageSource`, `Resistor` and `Pin` models) are declared and created. These two components are then interconnected through the use of a `Connect` model. Finally, the catalog constraint is modeled as a property using `Catalog` keyword (line 72). This implicitly specifies that an instance of `LiIonCell` can solely take the values of one tuple inside `cellTable`.

4.2.4 Modeling an elementary load profile

Figure 11 right shows how a **definite** elementary load profile can be modeled. An elementary load profile is modeled as a system with two terminals (`p` and `n`). It is defined by a duration (constant `deltat`), a required power (constant `ploadk`) and a required current (constant `iloadk`). The corresponding load voltage `vloadk` is also declared (line 97) as a difference of potentials (line 104).

```
75 Table cellTable
76 Attributes
77 type : INDEX ;
78 resistance : RESISTANCE ;
79 capacity : CHARGECAPACITY ;
80 mass : MASS ;
81 Tuples
82 [0, 0.007, 39, 1.05], (* cell 0 *)
83 [1, 0.01675, 25, 0.75], (* cell 1 *)
84 [2, 0.023, 16, 0.68], (* cell 2 *)
85 End
```

```
91 Model ELProfile(deltat, ploadk, iloadk)
92 Constants
93 ploadk : POWER ; (* required power *)
94 iloadk : INTENSITY ; (* required current*)
95 deltat : TIME ; (* duration *)
96 Variables
97 expr vloadk : VOLTAGE ; (* required voltage *)
98 Elements
99 p : Pin() ;
100 n : Pin() ;
101 Properties
102 p.i := iloadk ; (* receptor convention *)
103 n.i := - p.i ;
104 vloadk := p.phi - n.phi ;
105 vloadk = ploadk / iloadk ;
106 End
```

Figure 11. DEPS models. Left: Modeling the catalog of cells, Right: Modeling an elementary profile of load.

```
107 Model Todo(elp, b)
108 Constants
109 Variables
110 socini : REAL in [0.2,1] ;
111 expr socfin : REAL ;
112 Elements
113 elp : ELProfile[...] ;
114 b : Battery ;
115 g : Ground() ;
116 c : Connect(elp, b, g) ;
117 Properties
118 elp.iloadk <= b.ibat ;
119 elp.vloadk <= b.vbat ;
120 socfin := socini - b.cell.icel/b.cell.c3 * elp.deltat
        ;
121 socfin >= 0.2 ;
122 End
```

```
133 (* supply energy for the load profile *)

135 Model Supply(elp, elp2, elp3, b)
136 Constants
137 Variables
138 expr socini : REAL ;
139 expr socfin : REAL ;
140 expr energy : ENERGY ;
141 expr s : BOOLEAN ;
142 Elements
143 elp1 : ELProfile[...] ;
144 elp2 : ELProfile[...] ;
145 elp3 : ELProfile[...] ;
146 b : Battery ;
147 td1 : Todo(elp1, b) ;
148 td2 : Todo(elp2, b) ;
149 td3 : Todo(elp3, b) ;
150 Properties
151 socini = 1 ;
152 td1.socini = socini ;
153 td2.socini = td1.socfin ;
154 td3.socini = td2.socfin ;
155 energy := (elp1.ploadk * elp1.deltat + elp2.ploadk *
      elp2.deltat + elp3.ploadk * elp3.deltat) ;
156 b.energy >= energy ;
157 ...
158 End
```

Figure 12. DEPS models. Left: Modeling an energy supply demand, Right: Connecting the load profile and the battery.

4.2.5 Modeling the energy supply requirement

First, to model the requirements R1 and R3, a battery can be connected to each instance of elementary profile `elp` (of the model `ELProfile`): model `Todo` of the Fig. 12 left - line 116. The model `ELProfile` is specified with its signature in square brackets replaced here by *[. . .]* for clarity purposes. The power demand is defined in terms of current and voltage to supply, resp. lines 118 and 119. The battery copper losses are implicitly considered through the terminal voltage expression ($vbat = ocvbat - rbat \cdot ibat$). The design constraints related to the battery *SOC* are expressed lines 120-121. Second, the elementary load profiles can be connected to form the load profile, as in the model `Supply` (Fig. 12 right - lines 152-154)). The model `Supply` allows a higher level of implementation by allowing a direct interconnection between a battery and a load profile. In the context of design, the initial *SOC* of a new battery is known and equals 1 (line 151). This design constraint is independent from the load and can be expressed in `Supply`. In addition, the energy demand is specified (lines 155-156).

4.2.6 Modeling the design problem of the battery

A modeling of the battery design problem is shown on Fig. 13 left. The problem model is a part of the package `BatteryPackage`. Packages are specified by the keyword `Package`. `BatteryPackage` uses the packages `Types` and `Components` (lines 159-160) with the `Uses` keyword. `Types` and `Components` respectively contain the `Quantity`, `QuantityKind` and the `Models`. `Components` actually uses the package `CatalogofComponents` inside which the `Table cellTable` is defined. Then, the `BatteryPb` problem is created (lines 162-180).

Some constants are declared and defined to set limits on battery structure (*nsmax*, *npmax*, *mbatmax*) and its behavior,

```
159 Package BatteryPackage ;
160 Uses Types, Components ;
161 (* battery problem model *)
162 Problem BatteryPb
163 Constants
164 nsmax : INTEGER = 5000 ;
165 npmax : INTEGER = 5000 ;
166 mbatmax : MASS = 800 ;
167 icelmax : INTENSITY = 40 ;
168 energymax : ENERGY = 100000 ;
169 efficiencymin : REAL = 0.9 ;
170 Variables
171 Elements
172 battery : Battery() ;
173 elp1 : ELProfile(600, 10831, 22.8) ;
174 elp2 : ELProfile(600, 45210, 102) ;
175 elp3 : ELProfile(600, 740, 18.9) ;
176 supply1 : Supply(elp1, elp2, elp3, battery) ;
177 max1 : MaxValues(battery, nsmax, npmax, mbatmax,
        icelmax, energymax) ;
178 min1 : MinValues(battery, efficiencymin) ;
179 Properties
180 End
```

```
183 (* values limitations for a battery *)

185 Model MaxValues(b, nsmx, npmx, mbatmx, icelmx,
        energymx)
186 Constants
187 nsmx : INTEGER ;
188 npmx : INTEGER ;
189 mbatmx : MASS ;
190 icelmx : INTENSITY ;
191 energymx : ENERGY ;
192 Variables
193 Elements
194 b : Battery ;
195 Properties
196 b.ns <= nsmx ;
197 b.np <= npmx ;
198 b.mbat <= mbatmx ;
199 b.cell.icel <= icelmx ;
200 b.avenergy <= energymx ;
201 End
```

Figure 13. DEPS models. Left: Modeling the battery design problem, Right: Modeling of the design constraints limitations.

(*icelmax, SOC*), as specified by the requirements R7-R12 (lines 163-169). In the `Elements` section, a single battery instance is first created (line 172) as well as the three elementary load profiles specified by R4 (lines 173-175). The next lines express the connection of each battery with the load profile (R1-R3 - line 176). At last, the design constraints setting the limitations are expressed (lines 177-178).

The design constraints could be modeled as properties of the problem. However, as models are constructed to be reuseable, factorizing out recurring design constraints into parameterized models proves to be more efficient (Fig. 13 right). Requirement R12 is modeled inside a similar `MinValues` model. Thus, `Properties` section is also empty.

5 Compiling and solving a design problem

The environment `DEPS Studio` allows:

1. The edition of the packages (that include the models).
2. The organization in problem modeling projects (regrouping `Problem` and `Models` packages).
3. The compilation of design problems (built-in compiler).
4. The solving of design problems (built-in solver).

The consistency of each model is first checked by the compiler (syntax, lexicon and models dependencies). A single instance of the problem is created and other subdefinite elements are created from this instance. The debugging of the models is directly realized in `DEPS`. It does not rely on the language used by the solver. The errors capture occurs during this step, which allows the designers to correct their models if necessary, before solving. After compiling, solving can be considered. Then, `DEPS` models are compiled in `DEPS Studio`, to produce an internal and generic computational model for solving. A computational model gathers the sets of variables, domains of definition and the constraints elements, specified as a constraint network in the `DEPS` models [23]. The computational models can be solved either with the `DEPS Studio` built-in solver, or with external solver [47]. The solving methods employed by both solvers rely on Constraint Programming for mixed-type variables [48, 49, 50]. Finally, a set of admissible architectures is synthetized as a result of the solving of this design problem. For this design problem, the resolution time is about two seconds on a personal computer equipped with a dual core Intel i7-6600U processor and 8GB of RAM (CP solver uses a first-fail heuristic and an Hull consistance filtering). Like all Constraint Programming solvers, if several solutions exist, the solver will be able to find them all. Table 2 shows *one* (among several possible) admissible architecture for the `BatteryPb` problem. The architecture of the battery corresponding to this solution is defined by: (i) "type 0" cells, (ii) "5" cells in serie and (iii) "118" cells in parallel. No optimization criteria were specified in the requirements for this design issue. However, the `DEPS Studio` solver allows to propose an optimal solution as soon as one is able to express algebraically a cost function to be minimized (mass of the system, economic cost, etc). It is simply a matter of using an iterative loop that increasingly constrains this criterion until it reaches the limit where there are no more solutions. The use of this algorithm to solve optimization problems is classic in Constraint Programming [36].

Table 2: *One* (among several possible) admissible architecture for the BatteryPb problem.

Variables	Values	Units	Requirements	Variables	Values	Units	Requirements
type	0		R6	vbat	478	V	R1
ns	5		R7	powerbat	48831	W	R1, R2
np	118		R8	energy	93354	W·h	R11
icel	20	A	R10	soc	73	%	R3
ibat	102	A	R1	efficiency	96	%	R12

6 Conclusions

The purpose of this article is twofold. First, it presents an approach for pre-designing physical systems through a model-based approach based on a synthesis process. One of the main objectives of this work is to propose a methodology and a framework that is dedicated to industrial designers facing concrete and complex design problems. In this context, the vision embraced is at the same time constructive, inductive and evolutive. One of the key aspects of this approach is to encourage the formalization of the design requirements and the design problem in order to accommodate the point of view of the different stakeholders (customers, designers, experts, project leaders). This methodology is based on the problem specification language DEPS and its modeling/solving integrated environment DEPS Studio. The DEPS language exploits the many advantages of object oriented languages (reusability, decomposition, generalization/specialization, inheritance) and the resolution steps use Constraint Programming. This approach is advisable as soon as we design structured systems (functionally and/or structurally) and that we are able to establish a set of design properties such as equations, algebraic and/or logical inequalities. This does not serve the purpose of pure innovation or creativity.

Second, this methodology is illustrated on a use case based on the design of a Li-ion battery for electric vehicle. The relevance and effectiveness of this approach are demonstrated on this example. It is obvious that, as with any new method, a phase of learning the language and computer tools is necessary and that it will be easier for users who are already familiar with object-oriented modeling. However, being the result of an industrial/academic partnership, a special effort has been made so that the formalism and language can be used by experts from industrial design departments. In this way, we encourage those interested to test it for their own design needs and to help us improve the current version (more information is available on the DEPS Association website [20]). As with any object-oriented or module-based language, models built in DEPS can be reused for different or more complex design problems (e.g. a complete vehicle for the use case or for different requirements). Currently, the authors are working on extending the battery example to account for environmental constraints and integrating the battery into a larger design problem, electric or hybrid vehicle. The resolution of the use case shows that Constraint Programming is well suited because the resolution times remain very low. Nevertheless, the overall design time is not restricted to the resolution time, it must also include the time spent on specifying the requirements and modeling the design problem. One of the objectives of these extensions is to assess whether it is indeed possible to save design time by exploiting the re-usability of the models provided by our design approach and the use of the DEPS language.

The DEPS language is recent and therefore continues to evolve. Since its first version, new features have been added and some should emerge in the future. For example, the use case discussed in this article uses its ability to handle catalog constraints and mixed variable. This paper focuses on the design of physical systems, but other papers have addressed the design of software for embedded systems, which makes it possible to consider using it to design Cyber-Physical systems or Product-Service systems. In future work, we also plan to connect the solutions generated by the synthesis process to a modelica simulation in order to illustrate the synthesis/analysis cooperation in the manner of the overall design cycle of Figure 1. As illustrated in this figure, feedback loops on the specification of requirements and on the synthesis in pre-design will then be possible if the evaluation/validation phases invalidate the admissibility of an initially pre-designed solution.

References

[1] Jackson, M., and Zave, P., 1995. "Deriving Specifications from Requirements: An Example". In Proceedings 17th International Conference on Software Engineering (ICSE), pp. 15–24. doi= 10.1145/225014.225016.

[2] Jackson, M., 1997. "The meaning of requirements". *Annals of Software Engineering,* **3**(1), January, pp. 5–21. doi= 10.1023/A:1018990005598.

[3] Roozenburg, N. F. M., and Eekels, J., 1995. *Product Design: Fundamentals and Methods*. Wiley, West Sussex.

[4] Galea, A., Borg, J., Grech, A., and Farrugia, P., 2010. "Intelligent life-oriented design solution space selection". In International Design Conference (Design 2010).

[5] Narin'yan, A., 1980. "Subdefinite set - a formal model of uncompletely specified aggregate". In Symposium on Fuzzy Sets and Possibility Theory.

[6] Telerman, V., and Ushakov, D., 1996. "Subdefinite models as a variety of constraint programming". In Proceedings Eighth IEEE International Conference on Tools with Artificial Intelligence, pp. 157–164. doi=10.1109/TAI.1996.560446.

[7] Brisset, S., and Brochet, P., 2005. "Analytical model for the optimal design of a brushless DC wheel motor". *COMPEL - The international journal for COMPutation and Mathematics in ELectrical and electronic engineering,* **24**(3), September, pp. 829–848. doi= 10.1108/03321640510612952.

[8] Fontchastagner, J., Messine, F., and Lefevre, Y., 2007. "Design of electrical rotating machines by associating deterministic global optimization algorithm with combinatorial analytical and numerical models". *IEEE Transactions on Magnetics,* **43**(8), pp. 3411–3419.

[9] The official OMG MARTE web site: https://www.omg.org/omgmarte/.

[10] Friedenthal, S., Moore, A., and Steiner, R., 2015. *A Practical Guide to SysML, The Systems Modeling Language, 3rd edition*. the MK/OMG Press.

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

[12] Törner, F., Chen, D., Johansson, R., and Lönn, H. e. a., 2008. "Supporting an automotive safety case through systematic model based development - the east-adl2 approach". In SAE World Congress & Exhibition, SAE Technical Paper.

[13] The official Modelica Association web site: https://www.modelica.org/.

[14] Sun, M., Bakirtzis, G., Jafarzadeh, H., and Fleming, C., 2019. "Correct-by-construction: a contract-based semi-automated requirement decomposition process". *https://arxiv.org/pdf/1909.02070.pdf*.

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

[16] Bąk, K., Diskin, Z., Antkiewicz, M., Czarnecki, K., and Wąsowski, A., 2016. "Clafer : Unifying Class and Feature Modeling". *Journal Software and Systems Modeling,* **15**(3), July, pp. 811–845. doi= 10.1007/s10270-014-0441-1.

[17] Lorca, X., Prud'homme, C., and Fages, J.-G., 2014. *Choco3 Documentation.* TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S., Rennes, France.

[18] Leserf, P., Saqui-Sannes, P., Hugues, J., and Chaaban, K., 2015. "Sysml modeling for embedded systems design optimization: A case study.". In Proc of 3rd International Conference on Model-Driven Engineering and Software Development (MODELSWARD).

[19] Shah, A., Paredis, C., Burkhart, R., and Schaefer, D., 2012. "Combining Mathematical Programming and SysML for Automated Component Sizing of Hydraulic Systems". *JCISE - Journal of Computing and Information Science in Engineering,* **12**(4). doi= 10.1115/1.4007764.

[20] The official DEPS Association web site: https://www.depslink.com/.

[21] Yvars, P.-A., and Zimmer, L., 2014. "DEPS : Un langage pour la spécification de problèmes de conception de systèmes". In Proceeding of the 10th International Conference on Modeling, Optimization and SIMulation (MOSIM 2014).

[22] Zimmer, L., Yvars, P.-A., and Lafaye, M., 2020. "Models of requirements for avionics architecture synthesis: safety, capacity and security". In Proc of the 11th Complex System Design and Management (CSDM) conference.

[23] Yvars, P.-A., and Zimmer, L., 2019. "DEPS Studio: Un environnement intégré de modélisation et de résolution de problèmes de conception de systèmes". In Proc. of the 8th French Conference on Software Engineering CIEL'2019.

[24] Czarnecki, K., 1998. "Generative programming: Principles and techniques of software engineering based on automated configuration and fragment-based component models". PhD Thesis, Technical University Of Ilmenau, Department of Computer Science and Automation, October.

[25] Eichelberger, H., and Schmid, K., 2013. "A Systematic Analysis of Textual Variability Modeling Languages". In Proceedings of the 17th International Software Product LineConference (SPLC'13), ACM, pp. 12–21. doi= 10.1145/2491627.2491652.

[26] Standard for Application and Management of the Systems Engineering Process, 1999. , January.

[27] Technical Committee ISO/IECJTC1/SC7, 2019. ISO/IEC/IEEE 42020:2019 - Software, Systems and Enterprise — Architecture Processes. ISO/IEC/IEEE 42020:2019, 07.

[28] Badin, F., ed., 2013. *Hybrid Vehicles: From Components to System*. IFP énergies nouvelles publications. Editions Technip.

[29] Berg, H., 2015. *Batteries for Electric Vehicles: Materials and Electrochemistry*. Cambridge University Press.

[30] Warner, J. T., 2015. *The Handbook of Lithium-Ion Battery Pack Design: Chemistry, Components, Types and Terminology*. Elsevier Science.

[31] Beard, K. W., 2019. *Linden's Handbook of Batteries, 5th ed.* McGraw-Hill Education.

[32] ElMaraghy, W., ElMaraghy, H., Tomiyama, T., and Monostori, L., 2012. "Complexity in engineering design and manufacturing". *Cirp Annals – Manufacturing Technology,* **61**, pp. 793 – 814.

[33] Regnier, J., 2003. "Conception de systemes hétérogènes en génie Electrique par optimisation évolutionnaire multi-critère". PhD Thesis, Institut National Polytechnique de Toulouse, Laboratoire d'Electrotechnique et d'Electronique Industrielle de l'ENSEEIHT, December.
[34] Hongwen, H., Xiong, R., and Fan, J., 2011. "Evaluation of Lithium-Ion Battery Equivalent Circuit Models for State of Charge Estimation by an Experimental Approach". In Energies, Vol. 4, pp. 582–598. doi= 10.3390/en4040582.
[35] Yvars, P.-A., and L. Zimmer, L., 2018. "System sizing with a model-based approach: Application to the optimization of a power transmission system". *Mathematical Problems in Engineering,* **18**.
[36] Meyer, Y., and Yvars, P.-A., 2012. "Optimization of a passive structure for active vibration isolation: an interval-computation- and constraint-propagation-based approach". *Engineering Optimization,* **44**(12), pp. 1463 – 1489.
[37] Diampovesa, S., Hubert, A., Yvars, P.-A., Meyer, Y., and Zimmer, L., 2019. "Optimal design for electromagnetic devices: A synthesis approach using intervals and constraint-based methods". *International Journal of Applied Electromagnetics and Mechanics (IJAEM),* **60**(1), pp. 35 – 48.
[38] Kuchcinski, K., 2019. "Constraint programming in embedded systems design: Considered helpful". *Microprocessors and Microsystems,* **69**, pp. 24 – 34.
[39] Tsang, E., 1993. *Foundations of Constraint Satisfaction*. Academic Press, London and San Diego.
[40] Russell, S. J., and Norvig, P., 2010. *Artificial Intelligence: A Modern Approach.* Prentice Hall.
[41] Beldiceanu, N., 2007. "Special issue on global constraints". *Constraints. An international Journal. Springer-Verlag,* **12**(1).
[42] The official IBEX C++ library web site: http://www.ibex-lib.org/.
[43] Granvilliers, L. end Benhamou, F., 2006. "Algorithm 852: Realpaver: an interval solver using constraint satisfaction techniques". *ACM Transactions on Mathematical Software*, Mar.
[44] Suh, N. P., 2001. *Axiomatic Design: Advances and Applications,*. Oxford University Press.
[45] Hatchuel, A., and Weil, B., 2009. "C-K design theory: an advanced formulation.". *Research in Engineering Design,* **19**(4), pp. 181–192.
[46] Wellstead, P. E., 1979. *Introduction to Physical System Modelling*. Academic Press Ltd. ISBN:0-12-744380-0.
[47] Zimmer, L., Anglada, A., Christie, M., and Grandvillers, L., 2004. "Constraint explorer: a modelling and sizing tool for engineering design.". In Metamodelling and Constraint based problem solving for embodiment design. Support Systems in SCI.
[48] Benhamou, F., Goualard, F., Granvilliers, L., and Puget, J.-F., 1993. "Revising Hull and Box Consistency". In 16th International Conference on Logic Programming (ICLP).
[49] Davis, E., 1987. "Constraint propagation with interval labels". *Artificial Intelligence,* **32**(3), July, pp. 281–331.
[50] Lhomme, O., 1993. "Consistency techniques for numeric csps". In Proceedings of the 13th International Joint Conference on Artifical Intelligence (IJCAI), R. Bajcsy, ed., Vol. 1, p. 232–238.