Ingenieurgesellschaft für intelligente Lösungen und Systeme mbH
www.iils.de

White Paper

Total Engineering Automation

Vision and Realization with Graph-based Design Languages and the Design Cockpit 43® software suite

compiled by

Dipl.-Ing. Jens Schmidt
schmidt@iils.de

Management Summary

Vision: Imagine a world where your engineering design and analysis work is free of error-prone and time-consuming routine tasks so that you as an engineer can stay focused on tackling engaging questions and finding great solutions. Imagine a world where all your domain models for your engineering analysis needs are created automatically and consistently to each other and require no more repetitive manual interventions when updated. Imagine a world where the engineering design time of your system is significantly reduced down to the *theoretical limit* of the addition of the run time of the algorithms, therefore enabling you to concentrate on the steering of the design process and the creation of the best solutions candidates – rather than wasting most of your time on eliminating errors or filling the gaps in your iterative design process loops. . . .

Reality: Sounds to good to be true? Well, we at IILS believe that your time is your most valuable and precious engineering resource, which cannot be brought back once it has been consumed – it is gone forever. Therefore, rather then wasting your time at work with menial recurring tasks (e.g. manual model updates) that can be automated or battling broken processes that are not aligned to each other, your time is far better invested in creative engineering work. For this reason we developed a novel method and a supporting software tool for the total automation of engineering along the whole product life-cycle including knowledge representation and execution of engineering activities. Combined they enable you to actually harness the benefits of automatic consistent model creation and significant reduced design times mentioned above. The method is the use of so called *graph-based design languages* as a novel way to store engineering knowledge in a computer readable and re-executable graph representation. The software tool is the *Design Cockpit 43®*. It takes the design language as input and generates your requested design including all necessary models. This tandem can even respond automatically to new design challenges or requirement changes if the necessary solution principles are encoded in the design language.

Implementation: Our approach to design can be best compared to the widely used technology of programming languages and their corresponding powerful compiler suites. Graph-based design languages in UML correspond to the source code – here all your specific knowledge is stored. And the Design Cockpit 43® corresponds to the compiler – herewith the design language is compiled. The result is in the former case an executable computer program and in the later case the finished design of your system. If this sparked your interest but you're short on time, make sure to check at least the figures in section 3 (starting at page 10) to get a general idea of what is possible and has already been done. **(Clicking the blue reference numbers jumps directly to the content.)**

Contents

Conventions

Blue denotes click-able links.
Orange denotes reading hints.

Disclaimer

The term *mechanical design compiler* was first coined in 1989 by Alan Ward [1, 2] from MIT as the direct analogon to the so-called *silicon compilers* used at the time for VLSI chip design. In consequence, the term (mechanical design) *compiler* in this document refers only to the generic category of programs which can compile (i.e. translate) a high-level (mechanical design) language into a low-level language better suited for subsequent (mechanical design) simulation, evaluation and visualization.

1. Introduction – Current Challenges and new Technologies

The industrial success in the future will strongly depend on the usage of appropriate knowledge-based engineering tools in the design and manufacturing processes. With the ongoing spread of information technologies in the overall corporate structure (leading to a broad digital transformation of a company also called *"Industry 4.0"*) companies face four major challenges that stem from historically grown structures:

1) **Scattered Data Sources.** Many corporate data and information structures resemble more or less the structure of an archipelago – many loosely connected data islands. Here time and effort is spent searching for the required information and preparing them for the current process needs.
2) **Incoherent Processes** Some processes are already established. With new challenges and customer needs new processes are defined. This meta-process is usual and goes on over the lifetime of a company, thus forming an individual company process mesh/landscape. However, there is seldom a concentrated effort to actually revise *all* processes on their interplay, or their usefulness in the scope of new technologies. So, processes that may have been useful in the past may now very well be overcome.
3) **Manual Data Exchange via Documents.** Knowledge exchange between experts and different design phases is mainly document driven. This means there is an additional overhead in generating and reading those documents – time that could be better spent on actually designing the system.
4) **Manual Model Creation and Update.** "Modern products are typically complex and may comprise components and functions from a multitude of different domains such as electrical, mechanical, hydraulic, and thermal. Each domain usually requires a dedicated simulation model to be instantiated in a specific software tool" [3]. The creation of those specific domain models is usually done manually. Especially in early design phases, where model updates are quite frequent, this manual process is error prone (model consistency) and slow (manual iterations).

However with the advent of new information technologies these challenges now bear a huge potential for time and cost savings. This document presents a novel way of design by the use of the so-called *graph-based design languages* on the basis of the Unified Modeling Language (UML) as an innovative solution approach to encode all aspects of engineering know-how in an universally applicable and executable central model. This design language is then processed by a so-called *mechanical design compiler*[1], a supporting software tool (e.g. the Design Cockpit 43® by IILS), to generate all domain models automatically.

In this context it is worth mentioning that there is an ongoing controversial scientific debate whether or not such a mechanical design compiler for design compilation is feasible [4, 5]. As always, the *optimists* claim the answer to be *"yes"*, while the *pessimists* claim that the answer is *"no"* or even *"never"*. IILS just asks you for no more than to make your own engineering jugdement after having gone through the results shown in this paper.

This white paper is structured as follows: Section 2 analyses the current state of the design of complex systems and shows our solutions to the challenges mentioned. Section 3 shows graph-based design languages with the Design Cockpit 43® in action with examples from industry and research. Section 4 summarizes the findings, sets the challenges found here into context with our solution and provides a glimpse into a possible future.

[1]The term *mechanical design compiler* was first coined in 1989 by Alan Ward [1, 2] from MIT in several publications [1, 2], as the analogon to the at the time so-called *silicon compilers* for VLSI-chip design.

2. Theory – Problem Analysis and an Instrumental Solution

To get the idea how our approach works and why we believe that it is the best solution to the challenges mentioned, a short analysis of the design of complex systems (also called cyber-physical systems) is presented in this section followed by condensed descriptions of graph-based design languages and the supporting compiler Design Cockpit 43®.

2.1. Theory Problem or Management Problem

From the literature on the design of complex systems it can be stated that *"system integration [is] currently the largest obstacle to effective cyber-physical system (CPS) design, which is due primarily due to a lack of a solid scientific theoretical foundation for the subject. [...] Most large system builders have given up on any science or engineering discipline for system integration, they simply treat it as a management problem"* [6]. That means, currently many companies try to resolve the lack of knowledge regarding the actual physical couplings in the design of complex systems with manual rework in the case of an insufficient nominal product performance – manual rework that is at least partly or in some cases even entirely automatable, thus freeing time and resources for other productive tasks.

Furthermore it is stated that *"system integration today relies on ad hoc methods: After all the components have been designed and manufactured, existing integration methods aim simply at 'making it work somehow'. As the complexity of engineered systems continues to increase, our lack of a systematic theory for systems integration creates more and more problems"* [6]. This reflects the experiences of manufacturers, that the integration problem gets more and more difficult with increasing complexity and number of components. For this reason, a support with knowledge-based methods and tools could be very beneficial.

Lastly, it is concluded that *"finding a solution is difficult because system integration is the phase where essential design concerns usually separate into physical systems, software, and platform engineering come together and the hidden, poorly understood interactions and conflicts across design domains suddenly surface"* [6]. This leads to the conclusion, that the design of complex systems is a highly coupled network of interacting entities in various domains, e.g. mechanical, electrical, thermal, hydraulic, etc. that can only be mastered if there is a comprehensive theoretical understanding of the complex system and of its design process. In order to achieve this, an adequate support in information technology resp. a support with clever software tools is mandatory. Exactly this requirement of a design methodology and a supporting software tool suite for the design automation of complex systems is fulfilled by graph-based design languages and the Design Cockpit 43®, which are now industrially available.

2.2. Design of Complex Systems – A Brief Overview

In the following section, a generic design sequence of complex systems is shown in fig. 1. Red arrows symbolize the topological order in design decision making, i.e. the sequential nature of design steps due to existing pre- and postconditions. Engineering design often is done manually, which leads to the usual bottleneck in time and budget, or can be automated. The latter requires the development of a deep understanding of information processes and algorithms for design. Yellow arrows symbolize the iterative nature of the design process which occur when a dead-end is reached and one or even more former design decisions have to be undone or revised (so-called backtracking) and a new design decision path has to be restarted from there.

In fig. 1 the outer iteration “0) Product Design Space Exploration / Product Design Optimization Loop” makes only sense if all inner design activities are automated. The generic sequence of the necessary steps inside (numbered 1 through 9) in the design of complex systems reads as follows: Starting with “1) design requirements”, subsequent mappings are performed after the German design methodology by Pahl and Beitz [7], starting with a mapping from the requirements to the abstract product functions, followed by a mapping of product functions to solution principles and lastly a mapping from solution principles to embodiments or components.

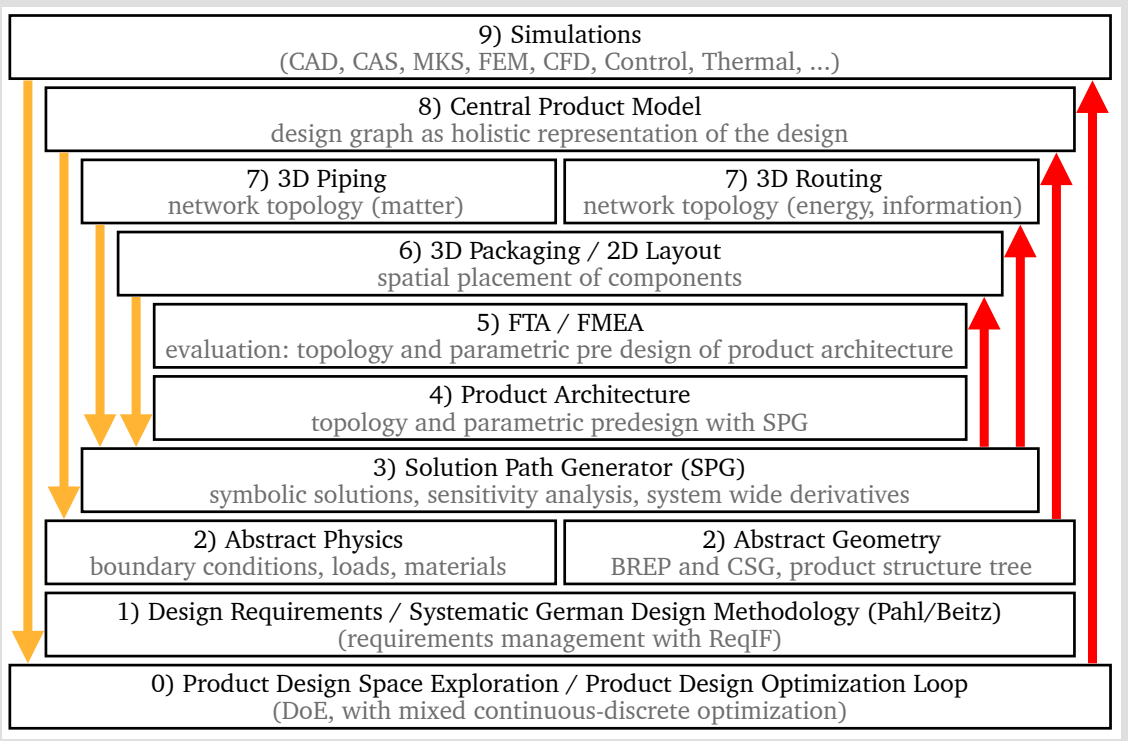

Figure 1: Necessary steps of the design of complex systems in their natural order. Red arrows symbolize the sequential procedure of designing – usually the next step can only be started if the previous design tasks were completed successfully. Yellow arrows symbolize the iterative nature of the design process – results of later design steps may prompt design changes in earlier stages and the design has to be redone from there.

Due to the antagonistic design principles *“form follows function”* and *“function follows form”*, geometry and physics are potentially both at the same time a requirement or result of a design process, depending on the design context [8]. The next step in the process is the representation, generation and handling of “2) abstract geometry and abstract physics”. In graph-based design languages the representation of geometry and physics is called *abstract*, since both geometry (represented as BREP, CSG and product tree) and physics (i.e. loads, boundary conditions, materials) are represented independently of any vendor-specific, proprietary data format or tool in the Unified Modeling Language (UML). All the design knowledge is therefore stored in UML, an open international ISO-Standard.

In the conceptual design phase geometry can often be represented with parameters and constraints. Also physics can at first be modeled in a simplified manner with non-linear algebraic equations without the need for partial differential equations. The “3) solution path generator” (SPG) handles and solves the resulting equation systems. The next step defines the “4) product architecture” of the system, i.e. determines which components exist and how they are interconnected. This product architecture is then evaluated top-down with a “5) fault-tree analysis (FTA)” and bottom-up with “5) failure-mode effects analysis

(FMEA)", depending on the properties of the defined architecture (i.e. component reliability and interconnections) and is modified automatically if necessary. Once the product architecture is determined and the components are selected, the system components with their geometry are spatially arranged in step "6) 3D-Packaging / 2D-Layout". After the packaging is completed, the interconnections of the components can be created physically in step "7) 3D-Routing / 3D-Piping)", either with electrical cables for transport of energy or information or pipes for the transport of gases or liquids.

The steps in fig. 1 allow the compilation of the graph-based design languages with the Design Cockpit 43® into a consistent "8) central product model" in form of the design graph which holds all the system design information. Based on the design graph, individual simulation plug-ins (i.e. CAD, MKS, FEM, CFD, etc.) of the Design Cockpit 43® map the abstract geometry and physics into domain-specific "9) simulation" tools (i.e. solvers). With this capability of mapping the abstract knowledge representation towards almost any proprietary piece of software with an API (application programming interface), arbitrary software landscapes found in companies can be interfaced. The remaining outer design loop "0) Design Space Exploration" finally represents an optimization loop, which is often employed with design of experiments (DoE) methods for computer-aided exploration of design spaces, see sections 3.1 and 3.2.

2.3. Graph-based Design Languages – (Re-)Executable Design Knowledge

Graph-based design languages are a new means for the holistic description of the task of engineering design which follows the composition scheme of natural languages where words form a vocabulary and rules form a grammar. Words in the context of graph-based design languages are the building blocks of the design on which rules operate. The set of all words (building blocks) in the language is called the vocabulary. It is encoded as an UML class diagram. Rules encode model transformations, they instantiate and operate on individual building blocks from the vocabulary which in turn are encoded in an extended UML instance diagram. The set of all rules is called a production system, which is encoded in an UML activity diagram. These three parts form the graph-based design language and have to be created manually by one or several human(s) as an upfront investment to the engineering design process. When the production system is executed by a so-called mechanical design compiler (here the Design Cockpit 43® see section 2.4) the complete design is created automatically and stored in a central model called the design graph, i.e. a holistic digital model of the system containing all parts, interconnections and parameters. From this central data model all other necessary domain models of a system, e.g. a CAD-model or a wire harness model, can also be generated automatically.

Here *design language* means that all allowed sentences in the grammar (i.e. all regular combinations of words) are a valid system design. The term *graph-based* means that an individual node in the graph serves as an abstract placeholder for a piece of design knowledge (i.e. a concept, a value, a physical component or an assembly thereof) and the edges in the graph express the (potentially N-dimensional and multidisciplinary) couplings between the different nodes (i.e. the different pieces of design knowledge). A (very) simple example for a car could look like this: Words are *Car*, *Chassis*, *Door*. Rules are (A) if there is nothing, create a car, (B) if there is a car, create its chassis, (C) if there is a car chassis, attach a door. Then a production system could look like this: (A) once, (B) once, (C) four times. The resulting design graph has one car with a chassis and 4 doors – how neat that the way you talk about your system can be encoded and used for design automation!

This simple design language could be extended by other words and rules to eventually design the whole car, including power calculations, geometry, all components down to the last screw and so on – see section 3 for applications in various fields.

The underlying graph data structure can be incrementally modified, by means of rule-based graphical data model transformations, and enriched with design knowledge until the design is completed in the aspired level of detail. The resulting *design graph* data structure can be automatically compiled and transformed into different domain-specific models (such as 3D geometry in a CAD program, a finite element representation in a FEM program, a finite volume representation in a CFD program, a lumped parameter model in a simulation program, etc.). Design languages support teams of design engineers by automatic consistent model and simulation generation, thus relieving the design team from frequent tedious and error-prone routine tasks. Figure 2 shows the integrated information flow with graph-based design languages and the Design Cockpit 43®.

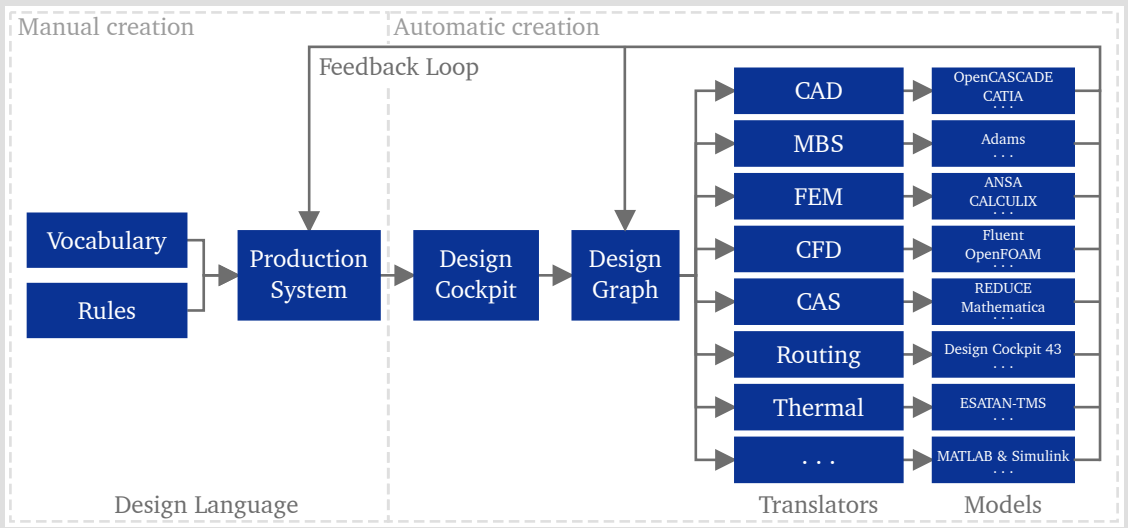

Figure 2: Information flow and automatic model creation with design languages [9]: A design language consisting of a vocabulary, rules and a production system is processed by a compiler (here: Design Cockpit 43®) to generate a holistic central model of a system - the design graph. Translators, such as for CAD, then automatically generate specific domain models, e.g. 3D-geometry in OpenCASCADE or CATIA [3].

Consequently graph-based design languages are a powerful concept which in combination with the Design Cockpit 43® allows to encode and automate design and manufacturing knowledge. The advantages of such an automation support are three-fold: firstly, the individual parameter sets in the different physical models are consistent to each other, secondly, after either a topological or parametrical change all the models are automatically updated to restore and guarantee the overall model consistency, and thirdly, the design process it automated, thus reducing the time needed for design cycles down to the execution time of the program.

2.4. The Design Cockpit 43 – A Mechanical Design Compiler Implementation

The Design Cockpit 43® creates a design graph during the compilation of the design language. This design graph is a complete, holistic model of the system design and contains all parts, parameters and interconnections. Design languages thus can be directly compared to well-known computer programming languages (such as C or JAVA) and their compiler suites. All design knowledge is encoded in the design language (source code),

the compiler only transforms the code into an executable program, or, in the case of a mechanical design compiler, into the finished design of a system.

One implementation of such a mechanical design compiler is the Design Cockpit 43® developed by IILS. It features an integrated development environment (IDE) for the graph-based design languages including, but not limited to, means for creating, running and debugging graph-based design languages. Also included are translator plug-ins for various domains, e.g. CAD, MBS, FEM, CFD and many more, see fig. 2. These plug-ins transform the abstract knowledge stored in the design graph into actual domain models of the system for analysis or further (manual) design. For example, the abstract geometry of a system is instantiated with a CAD kernel such as OpenCASCADE or CATIA, this geometry can then be used as input for other processes. Since the knowledge is stored in an abstract way, the change of CAD kernel requires no changes in the model, just the selection of another output format, see [3]. This mapping of abstract design knowledge to a specific software tool can be done for any domain, if there is a pair of domain design language and translator plug-in.

The Design Cockpit 43® features three operation modes:

1.) *Full mode* for the creation, modification and debugging of graph-based design languages. This mode requires also the highest level of expertise by the user.
2.) *Batch mode* as a “headless” mode, for embedding into other program suites. This mode requires no expertise by the user.
3.) *Slim mode* is an extended batch mode with user interaction (via wizards) at predefined steps in the design language. This mode can be used to build product configurators, see section 3.6. It’s also used to expose only a reduced set of design decisions to other engineers and requires therefore only little expertise by the user.

3. Applications – Examples from Industry and Research

Graph-based design languages are a powerful method which can encompass many of the design tasks in engineering. This is on the one hand the possibility to support the whole width of multidisciplinary design with all its many different domain models and on the other hand the possibility to support the entire depth of one domain with its specific design rules. Both aspects of design, width and depth, can be used at the same time in varying granularity, leading to the optimal support of the engineer in the needed detail.

For example in the early design phases of a satellite design width covering all domains is more important compared to design depth of one detailed domain. A satellite is a highly coupled system spanning many domains, e.g. mechanical, electrical, thermal, etc. A valid system design is only found when it is feasible in all domains. This can lead to many design iterations since a change in one domain usually leads to many consequences in other domains, e.g. the following sequence: If the power requirement increases, this leads to bigger solar panels, so in turn the moments of inertia and the system mass change, which leads to an increase in required propulsion power, this leads to increased tank volume which in turn changes the moments of inertia again and so on. These iteration cycles usually involve many dedicated domain experts.

Model creation or model updates which involve heavy topological and parametrical changes had therefore always to be done manually in the past. The reason was that the model consistency couldn’t be ensured otherwise. Nowadays, with the novel use of the Design Cockpit 43®, the model consistency can be automatically ensured, see section 3.1.

When more detailed models become necessary in later design phases, the depth and width of the design modeling can be increased incrementally using the same approach

of graph-based design languages. Dedicated design languages spanning the whole design depth for a specific sub-system can be created independently from the overall system design language. They can be later plugged into the system design language to replace the rough model generation from earlier design phases. Section 3.2 shows automatic design and model creation for one sub-system of a satellite.

Sometimes whole system considerations are not necessary, e.g. in the design of less strongly coupled systems, or when only a specific sub-task of the overall design process is of interest, e.g. in dedicated departments which only design one aspect of a system. In that case, graph-based design languages can be used to model and automate only the required sub-tasks of the design. Section 3.3 shows the design automation of the complete routing process including variations of the cabin layouts for aircrafts. In fact, automatic routing can be done in arbitrary complex 3D geometry since the Design Cockpit 43® includes a generic wire harness (routing) algorithm – its capabilities are shown in section 3.4.

Graph-based design languages can therefore be used in the early design phases to provide knowledge-based techniques “upstream” to the design process. Additionally design languages may also be applicable to any other “downstream” engineering activity of the whole product life-cycle. Since the creation of the digital factory can be seen and understood as a design task on its own, design languages also support the subsequent automatic generation of a digital factory according to the system specification of the earlier designed virtual product. Section 3.5 shows the continuous process chain from first concepts over the actual system to the factory work cell which manufactures said system on the example of a skin panel of an aircraft. The continuous integration in this process chain can also be used to include rapid prototyping as is shown in section 3.6 on the example of a 3D-printed coffee maker which has been designed with a product configurator based on graph-based design languages.

3.1. Multidisciplinary Design – System Level

The design of a complex system spanning many domains can be completely automated through the use of graph-based design languages and the Design Cockpit 43®. General system design rules and requirements can be encoded in a graph-based design language as well as the design of all necessary sub-systems. At each step in the design process models consistent to the current state of design and hence each other can be generated automatically enabling optimal design choices.

A first almost completely automated design of a complex system was developed in a PhD thesis [10] for a small satellite (the FireSAT [11] mission), based on previous work [12]. This includes design based on a set of requirements, automated thermal analysis and simulation of orbit behaviour in different operational modes. The FireSAT design language also includes the automatic design of some of the sub-systems: payload (telescope), power, communication and AOCS (attitude and orbit control). Additionally the design language of the FireSAT makes use of other design languages created by their respective domain experts, namely for: abstract geometry [3, 13], thermal analysis [14] and automated routing [15, 16, 17]. Figure 3 shows some of the models of the FireSAT generated with the design language for small satellites.

Models can be created in analytic domains, e.g. for controls as well as in geometric domains, e.g. for thermal analysis or routing. Other models are: the functional model of the satellite and its sub-systems including all relevant design equations and boundary conditions and a first concept of automatic finite element analysis utilizing the Design Cockpit 43®’s capability for abstract physics (in development).

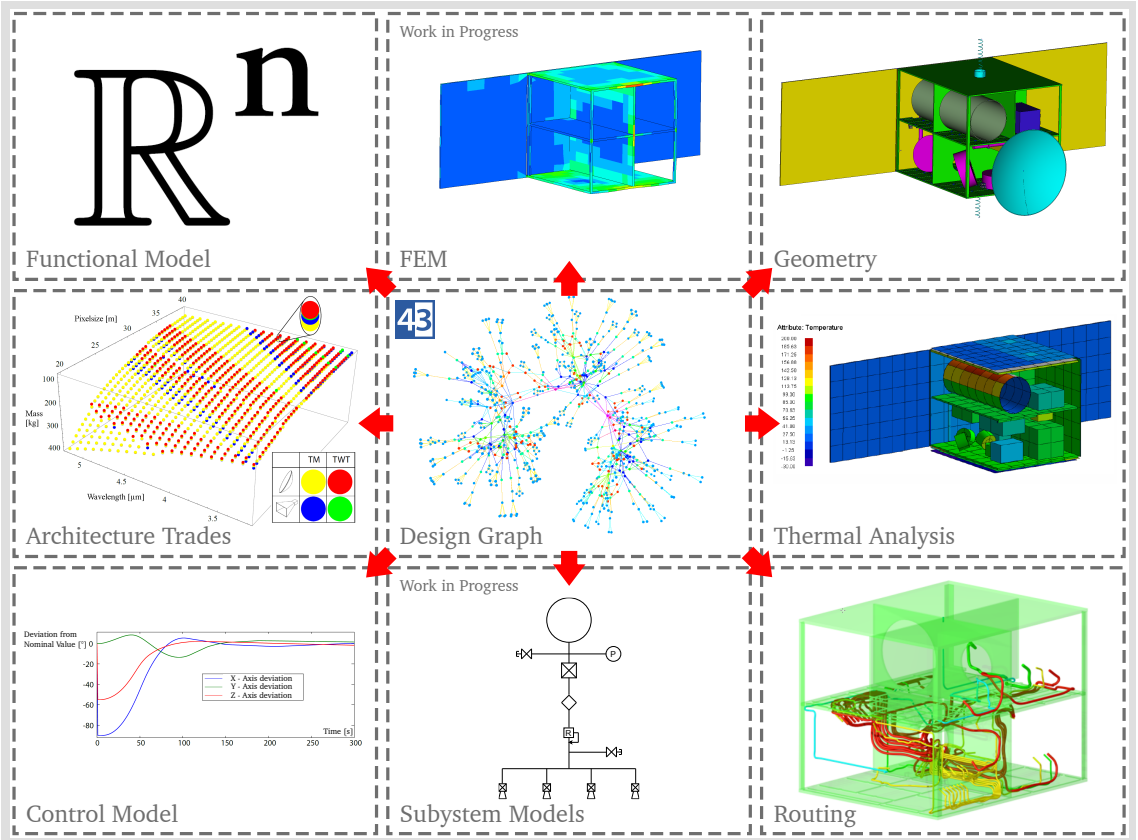

Figure 3: Overview of automatically generated consistent domain analysis models of a small satellite from a central model (design graph). Models can be created in geometric domains, e.g. for thermal analysis, as well as in analytic domains, e.g. for controls. The automated model creation then allows to raster the design space in a design of experiment (DoE), thus identifying optimal architecture choices for the current set of requirements [3, 10, 13, 14, 18]. **(If you're reading on a screen, zoom in for more details.)**

Automatic model generation follows the principles laid out in section 2. The design language is processed by the Design Cockpit 43® which generates the design graph, the holistic model of the system under design - it stores all information in an abstract manner. Several translators for specific domains then instantiate the model in the requested granularity in their own domain and software tool, e.g. the geometry model is created with the Design Cockpit 43®'s capability for abstract geometry, either with OpenCASCADE or CATIA, depending on the used translator plug-in, i.e. the CAD program can be changed at run-time with the push of a button without any change in the underlying model. Support for other systems is in development. This switch of a specific domain tool with another one can be done in any other domain as well – provided there is a software tool alternative and a corresponding translator plug-in.

The creation of a design language, e.g for the highly complex task of satellite design, can take up to several months. However once created, the design time of the encoded system is reduced to the run time of the algorithms in the design language [10]. Thus enabling rapid turn around times, which then allow for a sampling of the design space to find to optimal solution for the current set of requirements. The section "Architecture Trades" in fig. 3 shows such a design space sampling with up to 4 alternative topologies of the communication system of the satellite, see [10, 18] for more details.

Next steps in the development of the design language for the FireSAT could strive to include all aspects of satellite design. Right now the design language starts with the mission statement, the preceding design phase could be modeled to include the mission analysis

as well. For each subsystem more solution variants could be encoded, e.g. to expand the power subsystem from solar panels to include radio thermal generators (RTG) and fuel cells. Another addition could incorporate more choices in the satellite structure, e.g. additionally to the layered cuboid a segmented cylinder etc. [10].

3.2. Multidisciplinary Design – Sub-System Level

A system and its sub-systems can be modeled in one design language as is shown in section 3.1. It is also possible to use a more modular approach where specific sub-systems are modeled in their own design language. These design languages can be triggered either on their own or in a larger context, e.g. to only lay out one sub-system or to replace the original design language included in the overall design language.

In fact this modularization is not limited to sub-systems but can be done for individual domains as well, as was also shown in section 3.1. This allows for each domain expert to express his design knowledge in a dedicated and detailed design language. Other experts than can simply trigger this design language in their own design language. Figure 4 shows the production system of a design language for satellite propulsion systems which in turn uses design languages for reliability analysis [19], geometry [3] and routing [16].

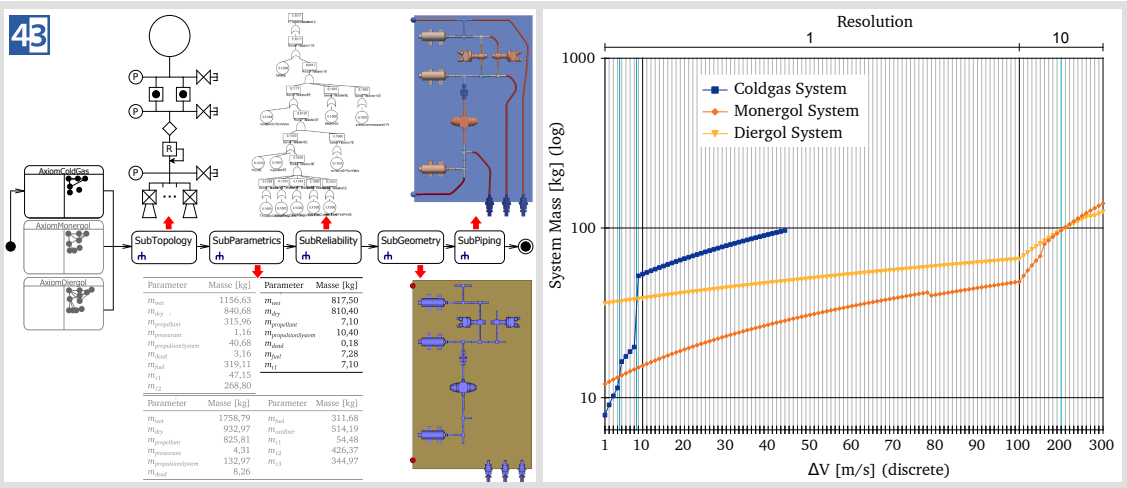

Figure 4: *(LEFT)* Design knowledge for satellite propulsion systems encoded in a productions system with design rules in sub programs (horizontally) and overlay of corresponding generated models (vertically). Starting with one of three system types, the design and models are created for topology, parametrics, reliability, geometry and piping [20]. *(RIGHT)* Sampling of the design space in a design of experiment (DoE) that can be spanned with the design language. Each data point is a completely laid out system in topology and parametrics. Turquoise lines show a change of "best" system in terms of mass. This version of the image is an excerpt showing only the left half, the full version can be found in [20]. **(If you're reading on a screen, zoom in for more details.)**

The production system shown on the left of fig. 4 follows an usual design process. Starting with one of three possible system types different variants of the selected system type can be created. Specific domain models are created successively with the design process. For a more detailed explanation see the next page and [20].

First the system topology is generated, i.e. all functions, parts and their interconnections. The topology can be visualized in a flow schematic. The next step adds parameters, e.g. mass, volumes, power and equations and iteration loops to the design and solves those resulting in the mass balance. Here also the parts of the system are selected. Right now part selection is done with a fixed set with one type per function, e.g. one type of check valve, except for the tanks, they are selected via a lookup-table depending on the needed

storage volume. With topology and parametrics in place the reliability can be evaluated, e.g. with a fault tree analysis [19]. If the system suffices, the next step creates the geometry of the parts of the system. As a last step in the design language, the parts of the propulsion system are then interconnected with pipes generated with the Design Cockpit 43®'s automatic routing capability to create the so-called equipment panels [20].

With the system and model generation automated, the design space of possible propulsion systems that can be generated with this design language, can be visualized as shown on the right side of fig. 4. The figure shows the left half of an extended DoE run which generated 453 propulsion systems in about 5 hours on a regular personal computer (Quadcore processor at 2,8GHz). Each data point is a completely laid out system in topology and parametrics. Points are linearly connected for better visualization of trends. The advantages of the design space exploration in that way are twofold: Firstly, this allows to immediately identify critical points in the design, where a small change in requirements facilitates a system change, and secondly, to discover more benign areas where changes in requirements do not promote a system change, can be found.

Future versions of the design language for satellite propulsion systems could improve on the part selection to include more parts in the lookup-table. This would allow for automatic trade-offs, e.g. to build-in less reliable and cheaper parts or a few expensive but more reliable parts. The calculation methods could also be improved to get a better prediction of fuel masses per maneuver, e.g. to include temperature changes in the satellite or real gas effects. Further improvements could include the automated integration of the propulsion system into a satellite.

3.3. Design Automation – Process Integration

Additionally to the design of whole (sub-)systems the approach of graph-based design languages is also applicable in a smaller scope, e.g. the automation of a specific task in an established design process. Figure 5 shows the automation of parts of the design process of an aircraft cabin, with the generation of floorplan, 3D-model and wire harness model.

The composition of the cabin harness is directly coupled with the cabin layout. For each seating row there is an overhead panel which includes reading lamps, buttons to call for flight attendees etc. additionally there may be an in-flight entertainment system with screens and audio jacks. These functions are driven by a small electronic box installed on top of the ceiling panel of the seating row. Depending on the chosen electrical architecture these small boxes are in turn connected to bigger management nodes. The number of managed boxes determines the size of those management nodes - a point for optimization, a few big or many small ones. Thus the number and positions of the seats determines directly the position and number of the electronic components which in turn define the harness length and architecture.

The automated design process begins with the generation of a cabin layout from a set of requirements, e.g. evacuation times, seat distances, passenger capacity, and so on. The cabin layout can be visualized with an automatically generated floorplan. Once the layout is fixed the CAD-model of the cabin interiour is generated by loading pre-defined 3D-geometry, e.g. seats or overhead bins, at the respective coordinates. Now the routing space can be extracted, this is the maximum possible volume where cables could be placed. Components of the electrical network, e.g. electronic boxes, are placed inside the routing space as start and endpoints for the routing algorithm. In a last step, the Design Cockpit 43®'s routing algorithm generates the wire harness of the aircraft cabin.

With this automated process in place, quick variation studies in the form of *"What happens if. . . "* are possible, e.g. the door is moved by one (fraction of a) frame or the lavatory

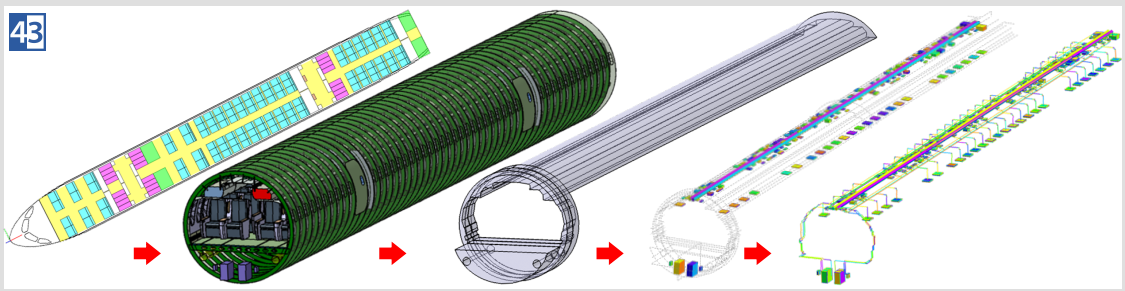

Figure 5: Automated design of an aircraft cabin including routing. From left to right: a set of requirements (not shown) drives the generation of the **cabin layout**, manual intervention is possible, e.g. to move the door to another frame if desired. According to the layout pre-constructed geometry is placed to generate a **CAD**-model. This then allows to calculate the maximum installation space for cables, the **routing space**. In this routing space **network components**, e.g. electronic boxes, are placed as start and end points for the routing algorithm. As last step the Design Cockpit 43®'s routing algorithm generate the cables and the electrical **wire harness** [15, 21]. **(If you're reading on a screen, zoom in for more details.)**

to passenger ratio is changed. Without automation such architecture trades would take up to several days/weeks/months and were simply not feasible for many iterations - since for each trade all models have to be updated by hand. Now a complete analysis cycle in the cabin design is completed within hours including consistent domain models for cabin layout (for/from operations), a CAD geometry (for/from construction in 3-D) and a wire harness (from/for network analysis) – all generated form a central model which covers several distinct engineering disciplines, the design graph [15]. To give some numbers, the problem size includes about 745 equipment boxes which are connected with 1357 cables – of course this number depends on the chosen cabin layout.

"A future task will be an even more detailed model implementation of this system integration problem to support the developing engineer in the design by further process automation. The data, collected from block diagrams and technical documentations of former aircrafts, can be replaced by the real data of the actual aircraft development. After that, algorithms considering functional requirements can be developed, to determine the number of equipment boxes automatically and to achieve automatic placement (e.g. packaging) by suitable algorithms", as elaborated in more detail in [15].

3.4. Design Automation – Automated 3D-Routing and Harness Generation

An important design effort constitutes the connections between different system parts, i.e. the cables for the transport of energy and information and pipes/hoses for the transport of matter. Manual routing can occupy an entire division of a company, where an army of engineers is tasked with "painting splines in 3D". This recurring task can entirely be automated, thus freeing engineers to do actual analysis and design, e.g. exploring different topology variants of the connection networks or finding trade-of candidates, e.g. the idea to vary size and number of management nodes in section 3.3. To support this design step, the Design Cockpit 43® includes a graph-based design language for automated routing in arbitrary complex 3D-geometry. Figure 6 shows selected examples of various routing scenarios generated with the Design Cockpit 43®.

The automated routing of cables including harness generation has now reached industrial quality and is used in automotive as well as aerospace applications. The automated routing of pipes is in the late stages of development, first applications are in the maritime industry with automated design of SCR-Systems [22].

A proof of concept video of the routing capability using a spatial maze can be found on our website [17]. Automatic routing for aerospace applications has been achieved in the

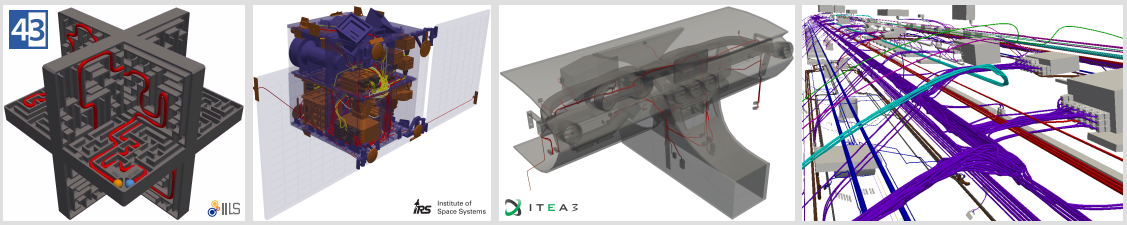

Figure 6: Selection of examples created with the Design Cockpit 43's generic routing algorithm capability for arbitrary complex 3D-geometry on the basis of graph-based design languages. From left to right: **IILS** Proof of concept, automatic routing in a complex spatial maze [17]. **Astronautics** Automatic Routing of an entire small satellite (Flying Laptop [23]) in a few minutes [16]. **Automotive** "Cockpit in 3 weeks" automatic routing as part of an integrated design process developed for the IDEaliSM Use Case 3 [24]. **Aeronautics** Close up of the generated routing, in an early state of the algorithm development, for the entire aircraft cabin of section 3.3 [15]. **(If you're reading on a screen, zoom in for more details.)**

context of two satellite projects. Namely the FireSAT [11] as example in a PhD thesis [10] (also see section 3.1) and the Flying Laptop [23] (launch scheduled 2017) as a real world comparison test case for the routing algorithm [16]. Further applications are developed right now for the automotive industry in the context of the EU-Project IDEaliSM [24] (use case 3), here the objective is to significantly reduce the overall design time of an entire car cockpit down to 3 weeks - automatic routing is an important step in that process. Last but not least routing of the entire aircraft cabin described in section 3.3 was achieved with an early version (2013) of the routing capability. In fact the Design Cockpit 43®'s automatic routing capabilities can be utilized in any engineering discipline where routing is required.

The Design Cockpit 43®'s automatic routing capability can either be used stand-alone or in conjunction with other design languages, thus allowing the automation of parts of already established processes as well as new ones. Future version of the Design Cockpit 43®'s routing capability will include routing of pipes and hoses.

3.5. Continuous Process Chain – Concept to System to Factory Planning

Changes in early design phases can have quite large effects in later design stages, when a small change at the beginning leads to huge changes and efforts later on, e.g. the diameter and length of an aircraft cabin is changed, this in turn changes the curvature and decomposition of the panels which comprise the fuselage, this can lead to a change in manufacturing sequence or tools needed. As of right now there seems to be no way to easily predict the consequences of design changes down the process chain in a economic manner, to see what early change accounts for the cascades of later changes – or is there? Graph-based design languages may handle that easily.

With the central model, incremental model modifications and automatic domain model creation within graph-based design languages are an excellent way to track and analyze the consequences of design changes/decisions in every design phase. Figure 7 shows a continuous process chain for an aircraft panel, from the design of the outer shape of an aircraft down to the layout of the work cell in a factory which assembles the panel.

With the process chain built, it is now possible to automatically design the outer shape of an aircraft, derive shape and decomposition of the skin panels, and even design a work cell in a factory which produces the panels. This in turn allows the investigation of the behavior of the entire design and manufacturing sequence. Since now any change in any design phase is automatically propagated to later design phases, the consequences of design changes can easily be assessed. With this new knowledge, manufacturing constraints

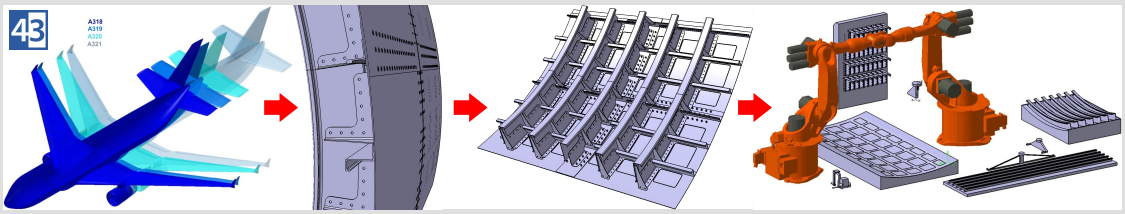

Figure 7: Example of a continuous process chain built with graph-based design languages. From left to right: The outer **shape** of an aircraft is designed with a design language [25]. To this outer shape description another design language adds boundary conditions regarding manufacturing, e.g. maximal frame distance or rivet sizes, and creates the **panels** which are then connected to **sections of panels** forming the outer hull of the aircraft [26]. From this panel description another design language can derive shape and placement of tools as well as manufacturing sequences to build up a **work cell** in a factory which can produce the panels [27]. This process chain is fully transparent, meaning a change in requirements early in the design process may change the shape of the aircraft. This change can be propagated down the process chain to automatically redesign the panels and even the factory. **(If you're reading on a screen, zoom in for more details.)**

can then be included in the preliminary design phase, e.g. the optimal size of each panel for optimal manufacturing efficiency, thus reducing cost and effort later on.

Future Versions of this process chain could include all parts of an aircraft, which can be manufactured in a factory, and the design of all work cells in a factory. This would allow to design the optimal factory for a given system or to design the optimal system for a given factory. Of course, this process to build a continuous process chain with graph-based design languages, can be repeated for any system.

3.6. Continuous Process Chain – System Requirements to Rapid Prototyping

Requirements Engineering is the starting point of the design process in many companies. While this approach allows to design and manufacture complex systems, it is not without its drawbacks, e.g. after the design has already been fixed, individual needs of new customers are difficult if not impossible to implement. This means new customers can only chose from a set of predefined design variations, or have to make a premium invest to get an "optimal" solution. This large set of design variants also leads to increased complexity and management effort of said variants.

However, with graph-based design languages the requirements and rules governing the design of a system can be encoded in a re-executable manner with varying input data already in mind, i.e. the customer requirements of the system. A new product variant that is consistent to the requirements and design rules can then be created automatically. This would enable companies to offer "one of a kind" product configurations with little to no extra cost/effort after the initial setup of a continuous process chain.

A proof of concept implementation of such a continuous process chain has already been done with graph-based design languages in [28] for a coffee maker. Not only is a design language used to automatically create a product, but also the the Design Cockpit 43® ability to display design specific user interfaces in the design process is utilized thus creating a "true" product configurator. True in that sense means that the product variants are not read from a database - instead the customer can change topology and parameters and still gets a valid and individual design which can be manufactured. Figure 8 shows one pane of the product configurator, various generated valid coffee maker designs and a resulting product manufactured with rapid prototyping technology.

Further additions to the design language could encompass designs for similar household products, e.g. espresso machines or electric kettles. The integration of the manufacturing

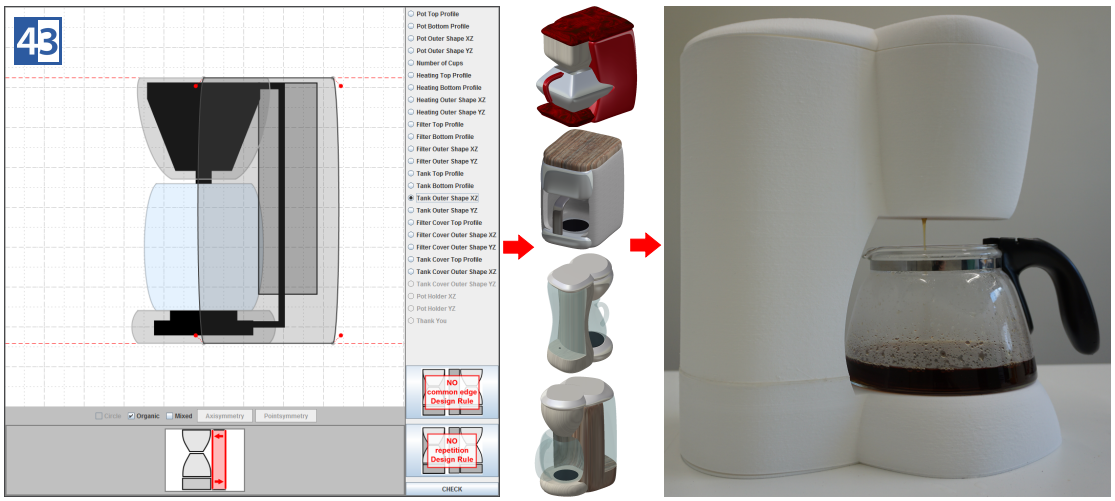

Figure 8: Proof of concept of a continuous process chain for "one of a kind" coffee makers including arbitrary user input through a product configurator created with graph-based design languages and the Design Cockpit 43®. From left to right: sample pane of the **product configurator** for coffee makers showing the modification of the shape of the water tank. From this user input the underlying design language creates **individual coffee makers** honoring the encoded requirements and design rules. One design of a coffee maker was manufactured via **rapid prototyping** to demonstrate the feasibility of the entire process chain not only with a computer model but with an actual product [28]. **(If you're reading on a screen, zoom in for more details.)**

effort could be improved upon with automatic data transfer to the 3d-printer or even addition of new processes, e.g. production and assembly techniques for components. Ultimately, the deployment of true product configurators on the base of graph-based design languages in the industry could revolutionize the way in which systems are designed and sold. This means that a solution for the quite complex and current problem of mass customization of a product with *"lot-size* 1" can be achieved straightforward by means of graph-based design languages.

4. Conclusion – Future Possibilities

The challenging problems of data structures (inconsistencies) and design processes (time to market) were identified. This prompted a brief analysis of the currently employed design methods and the design of complex systems in general, resulting in a new design method (graph-based design languages) and supporting software (the Design Cockpit 43®). The successful application of graph-based design languages has been demonstrated on various industry and R&D examples in section 3.

Automated multidisciplinary design, including automatic consistent model creation as well as Design of Experiments (DoE), is shown on system level for a satellite (section 3.1) and on sub-system level for a satellite propulsion system (section 3.2).

The integration of graph-based design languages in an existing design process is demonstrated on the example of automatic cabin layout and subsequent routing of the cabin wire harness driven by seat and electronic equipment positions (section 3.3). Stand alone routing examples are presented in section 3.4.

The possibility to build continuous process chains is shown using the example of an aircraft outer shape and the corresponding factory work cell in section 3.5. This continuous process chain allows the tracking and propagation of design decisions and changes through all design stages. A process chain is also built for the design and manufacturing

of a coffee maker (section 3.6) to demonstrate the use of graph-based design languages as “true” product configurators, which allow a wide range of parametric *and* topological changes by the customer during the design process. All examples together show the power of graph-based design languages and the Design Cockpit 43®. Yet, these examples represent only a subset of the possibilities of graph-based design languages and the Design Cockpit 43®.

Key to the vast extend of applicability of design languages in the whole product-life cycle as well as in individual applications is the underlying central graph data structure, the so-called design graph. Since a graph allows for arbitrary complex structures and relations between nodes, any aspect of engineering knowledge can be encoded with it. The central model allows seamless integration of various domains at any point during the design process in the necessary resolution. Changes in requirements or the design process are encoded in the rules and the production system of the design language leading to an incrementally growing executable knowledge base. In that knowledge base the design knowledge is stored in an abstract, implementation independent data format (i.e. UML).

The design language is processed by the Design Cockpit 43® to create the actual domain models for analysis from the abstract description. This enables the decoupling of knowledge from vendor specific software tools. As shown in fig. 2 the Design Cockpit 43® can interface various software alternatives per domain, e.g. CATIA or OpenCASCADE for CAD. New programs can be added with new translators – the design data stays the same.

To conclude, graph-based design languages and the Design Cockpit 43® are a way to avoid the shortcomings of many current company data structures and design processes identified in section 1. Scattered data sources can be eliminated with a central model, the design graph. Its creation depends on a set of valid design rules which formalize design decisions and steps, thus ensuring coherent processes that work together. Graph-based design languages also reduce the need to write lengthy documents since all design decisions are encoded in the production system and the resulting design with its parameters is stored in the design graph, i.e. exchange between your experts can be augmented with executable, and therefore validatable, information. Additionally a PDF documentation of the design language containing all classes, rules, production systems and comments, can be created automatically with the press of a button. Manual model creation and updates can be entirely a thing of the past, with graph-based design languages all domain models are created automatically for your analysis needs.

That raises an important question: what remains for the engineers to do? The answer is simple. Engineers in the future will invest their time into more productive and creative work. That means, the focus of an engineers work will shift from automatable routine tasks, e.g. manual model creation and manual model updates, back to engaging mental tasks of human problem solving and idea generation in order to come up with new solutions and more clever designs. These new designs and solutions can then be encoded into graph-based design languages for model synthesis and analysis. This shift of work is already part of the ongoing digitalization of processes and IILS is an expert in bringing these concepts to reality. If this sparked your interest, feel free to contact us.

References

[1] Alan Corlies Ward. “A Theory of Quantitaive Inference Applied to a Mechanical Design Compiler”. PhD thesis. Massachusetts Institute of Technology, 1989 (pages 2, 3).

[2] Alan Corlies Ward and Warren Seering. “Quantitative Inference in a Mechanical Design Compiler”. In: *A.I. Memo No 1062* (Jan. 1989), pp. 1–15 (pages 2, 3).

[3] Jens Schmidt and Stephan Rudolph. “Graph-Based Design Languages: A Lingua Franca for Product Design Including Abstract Geometry”. In: *IEEE Computer Graphics & Applications* 36.5 (Sept. 2016), pp. 88–93 (pages 3, 7–11).

[4] Daniel Whitney. “Why mechanical design cannot be like VLSI design”. In: *Research in Engineering Design* 8.3 (1996), pp. 125–138 (page 3).

[5] Erik Antonsson. “The potential for mechanical design compilation”. In: *Research in Engineering Design* 9 (1997), pp. 191–194 (page 3).

[6] Janos Sztipanovits, Xenofon Koutsoukos, and Gabor Karsai. “Toward a Science of Cyber-Physical System Integration”. In: *Proceedings of the IEEE* 100.1 (Jan. 2012) (page 4).

[7] Gerhard Pahl and Wolfgang Beitz. *Konstruktionslehre [Design Theory]*. 4th ed. Berlin, Germany: Springer, 1997 (page 5).

[8] Martin Motzer und Stephan Rudolph. “Über die Rolle der Geometrie im Systems Engineering [On the Role of Geometry in Systems Engineering]”. In: *Tag des Systems Engineering (TdSE) 2013*. Paderborn, Germany, Nov. 2013 (page 5).

[9] Stephan Rudolph. “Übertragung von Ähnlichkeitsbegriffen [Mapping of Similarity Concepts]”. Habilitationsschrift. Fakultät Luft– und Raumfahrttechnik und Geodäsie, Universität Stuttgart, 2002 (page 7).

[10] Johannes Groß. “Aufbau und Einsatz von Entwurfssprachen zur Auslegung von Satelliten [Architecture and application of satellite design languages]”. PhD thesis. Fakultät Luft– und Raumfahrttechnik und Geodäsie, Universität Stuttgart, 2014 (pages 9–11, 14).

[11] James Wertz and Wiley Larson, eds. *Space Mission Analysis and Design*. 3rd ed. El Segundo, California, USA: Microcosm Press, 1999 (pages 9, 14).

[12] Jörg Schäfer and Stephan Rudolph. “Satellite design by design grammars”. In: *Aerospace Science and Technology* 9.1 (Jan. 2005), pp. 81–91 (page 9).

[13] Johannes Groß and Stephan Rudolph. “Modeling Graph-Based Satellite Design Languages”. In: *Aerospace Science and Technology* 49 (Feb. 2016), pp. 63–72 (pages 9, 10).

[14] Johannes Groß and Stephan Rudolph. “Geometry and Simulation Modeling in Design Languages”. In: *Aerospace Science and Technology* 54 (Apr. 2016), pp. 183–191 (pages 9, 10).

[15] Stephan Rudolph et al. “On Multi-Disciplinary Architectural Synthesis and Analysis of Complex Systems with Graph-based Design Languages”. In: *DGLR Jahrestagung*. Stuttgart, Germany, Sept. 2013 (pages 9, 13, 14).

[16] Roland Weil. “Automatisierte Verkabelung des Kleinsatelliten Flying Laptop [Automated Routing of the Small Satellite Flying Laptop]”. Diplomarbeit. Fakultät Luft– und Raumfahrttechnik und Geodäsie, Universität Stuttgart, 2013 (pages 9, 11, 14).

[17] IILS mbH. *Kabel Routing*. last visited: 10.10.2016. url: `www.iils.de/routing.html` (pages 9, 13, 14).

[18] Johannes Groß and Stephan Rudolph. “Rule-based Spacecraft Design Space Exploration and Sensitivity Analysis”. In: *Aerospace Science and Technology* 55 (Jan. 2017), pp. 1–12 (page 10).

[19] Marius Riestenpatt genannt Richter, Jens Schmidt, and Stephan Rudolph. “Automated Fault-Tree Analysis of Complex Systems with Graph-Based Design Languages”. In: *6th International Conference on System Engineering and Concurrent Engineering in Space Applications (SECESA)*. Stuttgart, Germany, Oct. 2014 (pages 11, 12).

[20] Jens Schmidt and Stephan Rudolph. “Automation Opportunities in the Conceptual Design of Satellite Propulsion Systems”. In: *7th International Systems & Concurrent Engineering for Space Applications Conference (SECESA)*. (in print). Madrid, Spain, Oct. 2016 (pages 11, 12).

[21] Martin Motzer. “Integrierte Flugzeugrumpf- und Kabinenentwicklung mit graphenbasierten Entwurfssprachen [Integrated Design of Fuselage and Cabin for Aircrafts with Graph-based Design Languages]”. PhD thesis. Fakultät Luft– und Raumfahrttechnik und Geodäsie, Universität Stuttgart, 2016 (page 13).

[22] Samuel Vogel. “Über Ordnungsmechanismen im wissensbasierten Entwurf von SCR-Systemen [On Ordering Mechanisms in the Knowledge Based Design of SCR-Systems]”. PhD thesis. Fakultät Luft– und Raumfahrttechnik und Geodäsie, Universität Stuttgart, 2016 (page 13).

[23] Institut für Raumfahrtsysteme der Universität Stuttgart. *Flying Laptop*. last visited: 12.05.2017. url: www.irs.uni-stuttgart.de/archiv/veranstaltungsarchiv/flyinglaptop.html (page 14).

[24] EU-Project ITEA-3. *IDEaliSM Public Demonstrator*. last visited: 10.10.2016. url: www.idealism.eu (page 14).

[25] Daniel Böhnke, Markus Litz, and Stephan Rudolph. “Evaluation of Modeling Languages for Preliminary Airplane Design in Multidisciplinary Design Environments”. In: *Deutscher Luft- und Raumfahrtkongress*. Hamburg, Germany, Aug. 2010 (page 15).

[26] Stephan Rudolph, Jan-Peter Fuhr, and Laura Beilstein. “A validation method using design languages for weight approximation formulae in the early aircraft design phase”. In: *EUCOMAS*. Berlin, Germany, June 2010 (page 15).

[27] Peter Arnold and Stephan Rudolph. “Bridging the gap between product design and product manufacturing by means of graph-based design languages”. In: *9th International Symposium on Tools and Methods of Competitive Engineering (TMCE)*. Karlsruhe, Germany, May 2012 (page 15).

[28] Claudia Tonhäuser and Stephan Rudolph. “Individual Coffee Maker Design Using Graph-Based Design Languages”. In: *7th Conference on Design Cognition and Computing (DCC)*. Evanston, Illinois, USA, June 2016 (pages 15, 16).

[29] Karin Quack. *Wie Unternehmen Startup-Strukturen aufbauen [How companies create startup structures]*. Published online: 29.01.2015. url: http://www.cio.de/a/wie-unternehmen-startup-strukturen-aufbauen,3100472 (page 20).

A. IILS – About us

IILS, short for *"Ingenieurgesellschaft für intelligente Lösungen und Systeme mbH"*, which translates to *"engineering company for intelligent solutions and systems"*, was founded in 1999 as a Spin-Off of the Institute of Statics and Dynamics of Aerospace Structures at Stuttgart University by the former institute director, Prof. Dr.-Ing. Bernd-Helmut Kröplin. From 1999 to 2013, Dr.-Ing. Stephan Rudolph served as founding CEO. Today IILS is headed by the two CEOs Dipl.-Ing. Roland Weil (customer projects) and Dipl.-Ing. Peter Arnold (software development). IILS operates on two sites, the engineering development office located in the business park Echterdingen in Leinfelden-Echterdingen and the company headquarter in Trochtelfingen, Germany.

At IILS we believe that time is your most valuable and precious engineering resource, it cannot be bought with money and once consumed it is gone forever. For this reason we developed a novel method of designing and a supporting software tool. Combined they enable you to harness the benefits of automatic consistent model creation and significant reduced design times. The method is the use of so called *graph-based design languages* as a novel way to store engineering knowledge in a computer readable and re-executable graph format. The software tool is the *Design Cockpit 43®* which takes the design language as input and generates your requested design including all necessary models. In this respect our main goal is to create the *worldwide best compiler for graph-based design languages*. In addition to this, we develop innovative algorithms which optimally enhance the functionalities of graph-based design languages in the design of complex systems, e.g. 3D-Routing of electrical wire harnesses in arbitrary complex geometry or algorithms for the automated 3D-Piping of hydraulic systems. For this purpose, we closely cooperate with universities in the scope of doctorates embedded in various research projects.

Especially large companies, due to their well established management structures *"are perfectly capable of incremental innovation, however to get disruptive innovations to work, impulses from the outside are necessary"* [29]. We see therefore our role as *Think-Tank* for our clients, with the software development mainly as a means to facilitate the successful implementation of the (disruptive) innovation of graph-based design languages into their corporate structure, design and development processes.

B. IILS - Contact us

Feel free to contact us with your idea of cooperation:

Dipl.-Ing. Roland Weil (CEO, customer projects)
IILS Ingenieurgesellschaft für Intelligente Lösungen und Systeme mbH
Leinfelderstrasse 60, D-70771 Leinfelden-Echterdingen, Germany
Office +49 711 217249011
Mobile +49 163 3072760
Email weil@iils.de

Prof. Dr.-Ing. Markus Till (head of strategy)
IILS Ingenieurgesellschaft für Intelligente Lösungen und Systeme mbH
Albstrasse 6, D-72818 Trochtelfingen, Germany
Mobile +49 152 01904669
Email till@iils.de