CEAS Aeronautical Journal (2023) 14:591–606
https://doi.org/10.1007/s13272-023-00642-5

ORIGINAL PAPER

System integration based on packing, piping and harness routing automation using graph-based design languages

J. Dinkelacker[1,2] · D. Kaiser[1] · M. Panzeri[3] · P. Parmentier[3] · M. Neumaier[2] · C. Tonhäuser[2] · S. Rudolph[2]

Received: 7 March 2022 / Revised: 11 January 2023 / Accepted: 16 January 2023 / Published online: 9 March 2023

Abstract
The implementation of a fully instrumented, automated and simulation-enabled engineering software platform capable of automating the currently still manual model-based systems engineering (MBSE) design process for physical systems architecture generation and optimization in an aircraft wing is presented. The software platform uses graph-based design languages to integrate and entirely automate the mainly manual packing, piping and harness routing design. This design automation and optimization is achieved by a novel software stack of an optimization software coupled with a design compiler. It is shown that through rule-based model generation by a design compiler in the form of a design graph as a central data model, a cross-domain data consistency is achieved. This allows for automated execution and coupling of engineering tasks over several different domains such as packing, piping and routing design to converge to an optimized wing physical architecture design variant in agreement with given predetermined design constraints.

Keywords Design automation · Model-based system engineering · Graph-based design languages · Packing, piping, routing

1 Introduction

The work in this paper describes the implementation of a simulation-enabled engineering software platform which is capable to automate the model-based systems engineering (MBSE) design process for physical systems architecture generation and optimization. The automation of the MBSE process is demonstrated on the optimal packing, piping and routing of an aircraft wing. The implementation is based mainly on parts of the outcome of the CLEANSKY2 research project PHAROS (*Physical Architecture Optimization System*) [1–3] of the project partners IFB, IILS, NOESIS, and AIRBUS.

In order to fully automate the system integration effort based on a sequence of packing, piping and harness routing algorithms encoded in graph-based design languages, the formerly manual process chain is automated step by step. This is achieved within a novel integrated engineering software stack based on an *optimizer*[1] with a so-called *design compiler*[2]. The design compiler translates the graph-based design languages into models which are executable and simulated in internal or external third party solvers or programs and the optimizer reads back in the results and modifies the relevant model parameters accordingly to achieve optimal system variants.

Graph-based design languages take their inspiration from human language, in which a grammar, formed by vocabulary and rules, can be used to formulate an expression. Similarly in a *design language*, design vocabulary and rules form a production system (compare grammar) formulate the expression of a valid design. In *graph-based* design languages, this design is represented as a graph, in which each node

The content of this paper reflects only the opinions and views of the authors and the JU is not responsible for any use that may be made of the information this paper contains.

✉ J. Dinkelacker
dinkelacker@iils.de

1 IILS Ingenieurgesellschaft für intelligente Lösungen und Systeme, Trochtelfingen, Germany

2 University of Stuttgart, Institute of Aircraft Design (IFB), Stuttgart, Germany

3 Noesis Solutions, Leuven, Belgium

[1] The *optimizer* used here is OPTIMUS®, a software product of the PHAROS project partner NOESIS bv, Belgium. See the website `www.noesissolutions.com` for information on OPTIMUS®.

[2] The *design compiler* used in this work to compile graph-based design languages is the Design Compiler 43®, a software product of the PHAROS project partner IILS mbH, Germany. See the website `www.iils.de` for information on the Design Compiler 43®.

represents a requirement, a product function, a solution principle, a component or an assembly, or any other engineering concept. This design graph encodes the complete knowledge of a product design and its associated design requirements and goals in a machine-readable and machine-executable form. The executor is the *design compiler*, which translates a design language into a design graph.

For the implementation, diagram types of the internationally standardized Unified Modeling Language (UML) are used. The vocabulary is defined in UML class diagrams and rules are defined as Model-to-Model (M2M) transformations, which only allow modification according to the corresponding class diagrams. The rules are combined into the production system as a UML activity diagram, which is translated by the design compiler into the design graph in form of an UML instance diagram. These representations in the UML standard allow for great flexibility in the formulation of any design problem as a design language and also permit further developments through UML's predefined extension mechanism. After compilation the instance diagram of the design graph can be translated into domain-specific languages (DSL) by dedicated plug-ins for further processing. Results of the processing can be returned into the design graph, or even be used to alter the program sequence of the production system. More information on graph-based design languages can be found in [4–6]. The design language presented here includes the disciplinary models of *packing*, *piping* and *routing* in order to draw conclusions for the optimization of the design of the physical system architecture of an aircraft flight control system. Because of the integration of these three design domains into one design language, consistency between the individual engineering models of each domain is automatically guaranteed by their shared source, the design graph.

2 Related work

An overlook of related works on algorithmic approaches to packing, piping, routing and optimization problems is shortly given hereafter. Since the integration of these processes in a single software platform is a novel development, this review of related works is focused on the individual application of the algorithmic approaches. Multidisciplinary optimization of complex physical systems has been identified as a tool to help navigate the complexity of preliminary aircraft design as soon as computer-aided design became available for engineers [7].

Different approaches are discussed in [8, 9]. The optimization approach of the presented problem is closest to the Individual Discipline Feasible Design (IDF) [8], in that each individual algorithm produces a feasible design solution for the according discipline, while the global optimizer has control over the parameters in each algorithm, to reach a globally optimal design solution.

A comparable problem definition of a preliminary design of an integrated Flight Control System is presented in [10], but differs greatly in the degree of automation, sub-system-linking and design intelligence. The fact that the integrated automated packing, piping and routing all belong to the class of NP-complete mathematical problems demonstrates the theoretical difficulty for individual and global design solutions.

Packing. Optimized packing of objects plays an important role in many areas of engineering design. The problem is known in literature as 3-D bin packing problem (BPP) [11]. The developed packing algorithm uses a Particle Swarm Optimization (PSO) for the exploration of the design space. It has been shown to give good results in the context of aircraft wing design [12] although not in the same problem class. In other recent work in [13, 14] the combined optimization of packing and routing of simplified bars for components and routing paths is discussed. This approach enables the use of gradient-based methods to generate feasible layouts, but is not necessarily applicable to more complex geometries.

Piping. Some approaches for automated pipe design have been proposed in the past. There are grid-based approaches, like [15], but these have the drawback that the bend position and the bend angles are limited by the available grid points. Therefore, a dense grid is preferable, which leads to high computational requirements.

More sophisticated approaches take different optimization objectives, like the length and the bend angles, as well as further constraints, such as the maximum bend angle and a minimal straight length between two bends, into account and are not limited to a grid. Such an approach, without the interaction of the pipes between each other, is shown in [16]. A combined packing and piping approach was proposed by [17]. In this approach a 2-D problem without obstacles is solved by a gradient-based algorithm using sensitivities. A multi-objective ant colony optimization is presented in [18].

Routing. The industry standard for wire harness development is divided into the electrical and mechanical design process, which have to be integrated manually into one coherent wire design, using electrical workbenches for CAD tools. Routing algorithms for path finding have been predominantly formulated as 2-D problems, like Dijkstra- and A*-algorithms. An enhanced A*-algorithm with attraction and repulsion is presented in [19]. Fundamental work in [20, 21] was expanded in [22] to develop a virtual environment for the 3-D wire harness design problem. An automatic clamp placement and hybrid multi-objective optimization in combination with harness routing is presented in [23].

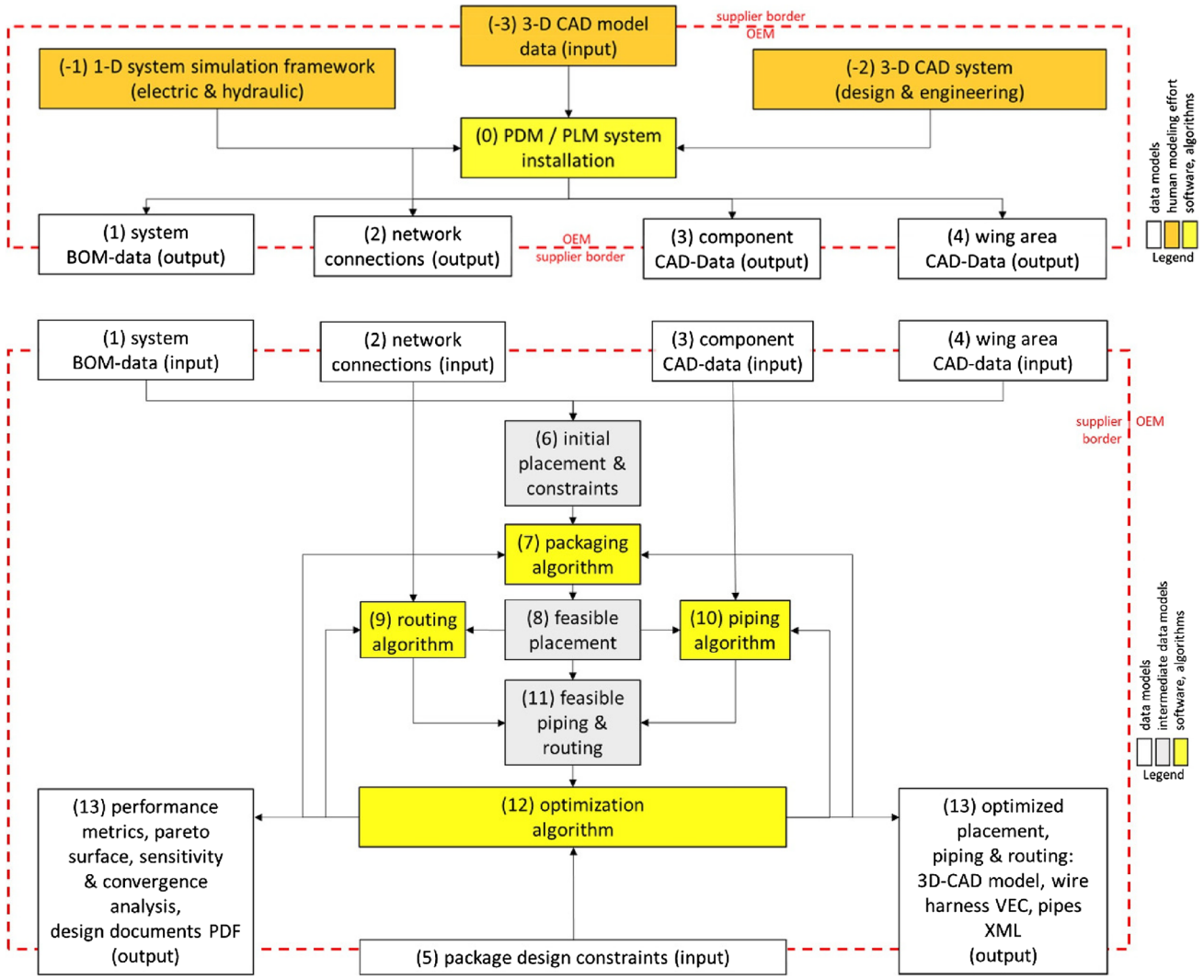

Fig. 1 Models, data flows and algorithm sequence overview in the PHAROS project context [1]

3 Program structure

The structure of the software platform is based on the models, algorithms and data flows in the establishment of a physical architecture of an aircraft wing, individually numbered from (− 3) to (13) and defined in the context of the PHAROS project [1] in Fig. 1.

The task of designing a physical system architecture for a wing is a joint effort led by the OEM (Original Equipment Manufacturer) and their suppliers. In the upfront part, the OEM maps the requirements onto the definition of the functional and logical system architecture in a manually-driven MBSE process, as shown in the upper half of Fig. 1. The expected future system behavior is validated preliminary through 1-D simulations, in this special case in form of a Modelica[3] model (marked (− 1) in Fig. 1).

The 1-D simulation models and simulation results are stored in a PDM / PLM (Product Data / Lifecycle Management) system (marked (0) in Fig. 1). From several other departments of the OEM and (from a potential multitude of) suppliers, all relevant 3-D CAD models (marked (− 2) and (− 3) in Fig. 1) of all system components and the available installation space (i.e., the wing structure) are stored in the PDM/PLM system. As output, the geometry of the installation space (marked (4) in Fig. 1), the CAD-data of the system components (marked (3) in Fig. 1), the network

[3] Modelica® is a unified object-oriented language for systems modeling developed by and registered trademark of the Modelica Association. See the website `www.modelica.org`.

connection lists (marked (2) in Fig. 1), and a bill of material (BOM) (marked (1) in Fig. 1) are available as design data for the subsequent tasks of packing, routing and piping by a supplier (here implemented by algorithms).

These data from the several design domains (1-D simulation and 3-D geometry) are considered as input to the described automation process. To simulate the integration into the described OEM process, the data are created manually and its components are marked across the domains with a unique identifier, to ensure consistency between the models.

The lower half of Fig. 1 shows the current State-of-the-Art (SotA) engineering design process of packing, piping and routing in form of a software tool and design process landscape. The output data sets explained for the upper half of Fig. 1 (BOM, network lists, CAD models of components and installation space) serve as input (identically marked (1, 2, 3 and 4) in Fig. 1) for the execution of the three developed algorithms of packing, piping and routing.

A set of given design constraints (marked (5) in Fig. 1) are translated into mathematical rules or ontologies, which are combined with a potential initial placement (marked (6)) as a starting point for a packing algorithm (marked (7)). This is embedded in and steered by an optimization strategy (marked (12)) which computes a feasible placement of the systems components (marked (8)). Based on this feasible placement, a piping and routing algorithm (marked (9, 10)) compute a feasible piping and routing (marked (11)). Finally, as a result, the optimized placement, piping and routing models (as 3-D CAD), the electrical wire harness (as XML) and the hydraulic pipe network (as XML) are provided. These results include the requested performance metrics and other related design documents (both marked as (13) in Fig. 1).

In the following, the sequential interplay of the completely automated algorithmic model generation of linkage extraction, design space geometry generation, packing, piping and routing is illustrated. This sequence of algorithms is implemented in design languages, embedded in an optimization loop, which is executed in the PHAROS software stack consisting of *OPTIMUS*® and *Design Compiler 43*®. The different parts of the software platform are described in order:

- Sect. 3.1 *Modelica linkage extraction* explains and demonstrates in short the extraction of the linkage information of the Modelica system components contained in a Modelica system simulation file into a design graph, from which a so-called "from-to" connection list as input for the routing and piping algorithms is derived.
- Sect. 3.2 *Geometry Enrichment* shows the import of the geometry data as input for the packing, piping and harness routing algorithms.
- Sect. 3.3 *Packing*, 3.4 *Piping* and 3.5 *Harness Routing* illustrate the results of the running process chain of the packing, piping and routing algorithms and give a brief summary of the fundamental functionalities of the respective algorithms. The focus is on the automated generation of the individual program results to prove the automated execution.
- Sect. 3.6 gives an overview of the optimization strategy and the performed optimizations of the software stack.

Each domain algorithm is encoded in a design language, specifically tasked to solve its corresponding problem in the design process. They posses their individual vocabulary, rules and production system. All individual design languages are aggregated into a single execution design language, which itself is embedded in an super ordinate optimization loop controlled by *OPTIMUS*®. This means, that each design language can first be developed and described independently. Individual design languages may have dependencies on other design languages and external programs, needed for the solution of their respective task. General re-executable algorithms can be formulated in their own design language, that can be called for a specific application of a problem. Integratability of the different parts is ensured by the design language approach.

The results of the software stack rely on academic data of a flight control system architecture and the wing and component geometries. This academic data consists of a manually created CAD-model of the design space which exhibits the most important topological and parametric characteristics expected in an industrial CAD-model such as type of parts and size as well as a corresponding system model in form of a Modelica model. The use of academic data may obscure some difficulties, which can only become apparent, if industrial data from a manufacturer are used. However, the software stack is transferable to such data, since the processes and algorithms are applicable to arbitrary geometric input and system data.

3.1 Modelica linkage extraction

According to Fig. 1, information on system component connections of the electrical wire harness and hydraulic piping is implicitly contained in the Modelica system model. This Modelica model would be used in an early design phase for the preliminary dimensioning and 1-D simulation of a system and is created manually as input for the optimization process.

Such a Modelica model of a flight control system simulation is shown in Fig. 2 as presented in the OpenModelica editor OMEdit. The system model of a flight control system includes a main actuator control unit (ACU) and several peripheral actuators and control surfaces. Additionally to the

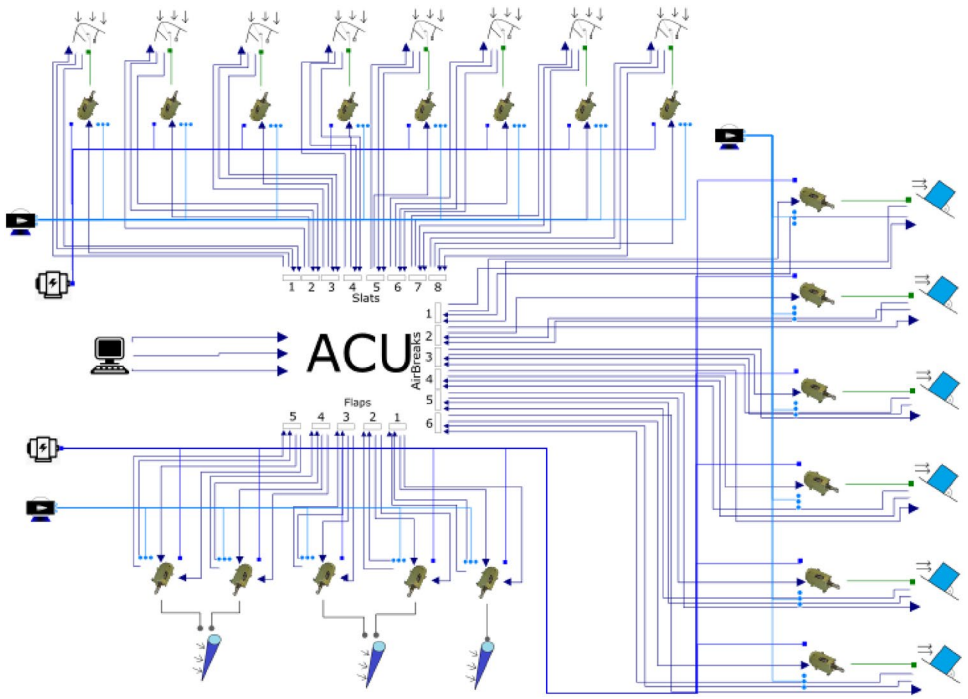

Fig. 2 Modelica system model of a flight control system simulation in OpenModelica OMEdit

electric actuators, the flight control system possesses a redundant hydraulic actuation system. The top row of Fig. 2 shows the eight slats, the right column shows the six spoilers and the bottom row shows the three flaps with their respective corresponding actuators. The model has two generators depicted on the left and three hydraulic pumps, for each control surface type, depicted next to them. Electric connections for signal or power are colored dark blue and hydraulic connections are colored light blue. A comprehensive list of the system components can be read from the design graph in Fig. 6.

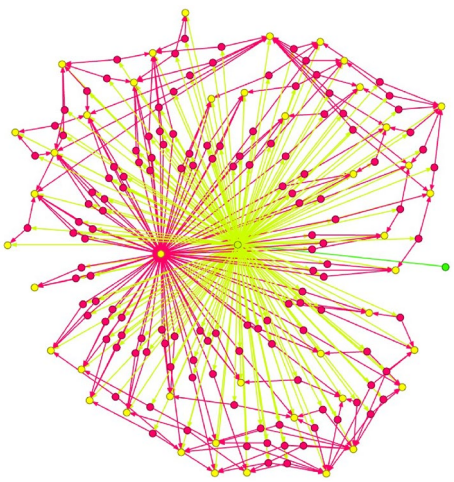

Fig. 3 Translated design graph of the Modelica system model shown in Fig. 2

A key functionality of the Modelica modeling language is the ability to rely on external libraries to build and simulate complex user-specific models. At the core of the various free and commercial libraries is the standard Modelica library, which contains models from multiple domains, from simple logical gates to mechanical and electrical components. Many models are built by instantiating components and interconnecting them, to reproduce the desired system behavior. As the Modelica language specification encompasses many functionalities, the subsequent methods focus on the extraction of the information of components and connections.

The object-oriented nature of the Modelica language definition makes the translation to a representation in a design graph straightforward. The extracted information of the system model includes the names, types and library paths of the model components, with their specified parameters, as well as the connections between them. Further specifications about the model components are defined in the respective Modelica file. Figure 3 shows the Modelica system model translated into a design graph in a graph-based design language in the *Design Compiler 43*®.

```
connect Component: mainACU1 Connector: spoilerspeed5
with Component: spoiler5 Connector: speedOutput
connect Component: flapactuator1 Connector: flange_b2
with Component: flap1 Connector: flange_a
connect Component: generator1 Connector: generatorOutput
with Component: flapactuator1 Connector: powerSupply
```

Fig. 4 Section of extracted connection list from Fig. 2

In Fig. 3, the Modelica components are represented as red nodes and the connections are shown as green nodes in the design graph. This design graph can be used to extract the connection list, shown in part in Fig. 4, or to be expanded on in a design language.

The linkage extraction from the Modelica model file is implemented using a parser to translate the input Modelica system model from its concrete syntax formulation into an abstract syntax tree (AST). This generic form of the AST is then interpreted using dedicated rules in order to extract the required information of the system model elements and their respective interconnections into a design graph. This design graph can be used to specify and expand on aspects of the system defined in the Modelica model and serves as a starting point for the design process for the system architecture optimization.

3.2 Geometry enrichment

The extracted design graph of the Modelica system model is appended by geometric data, which is also provided in the defined input data. In the execution of the subsequent steps

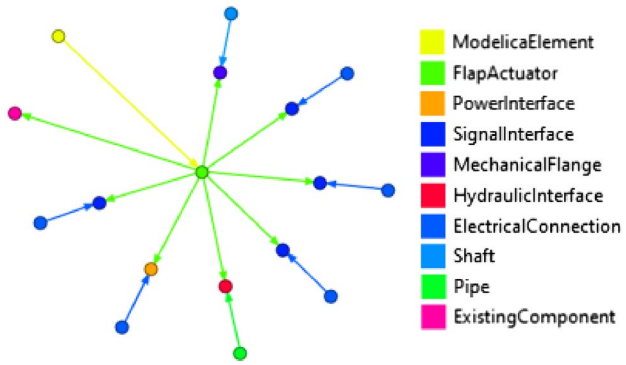

Fig. 5 Example of a flap actuator in the design graph

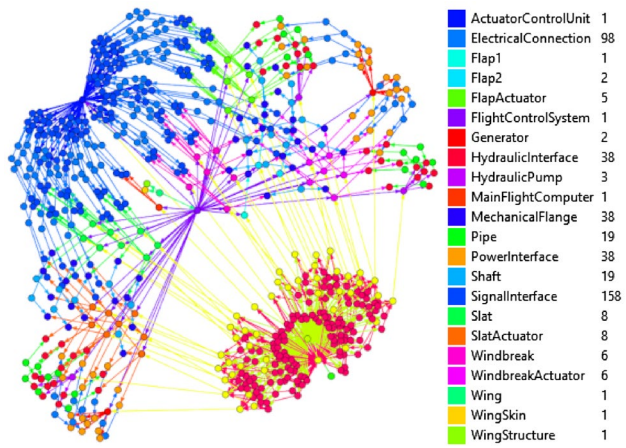

Fig. 6 Extended design graph including geometrical data

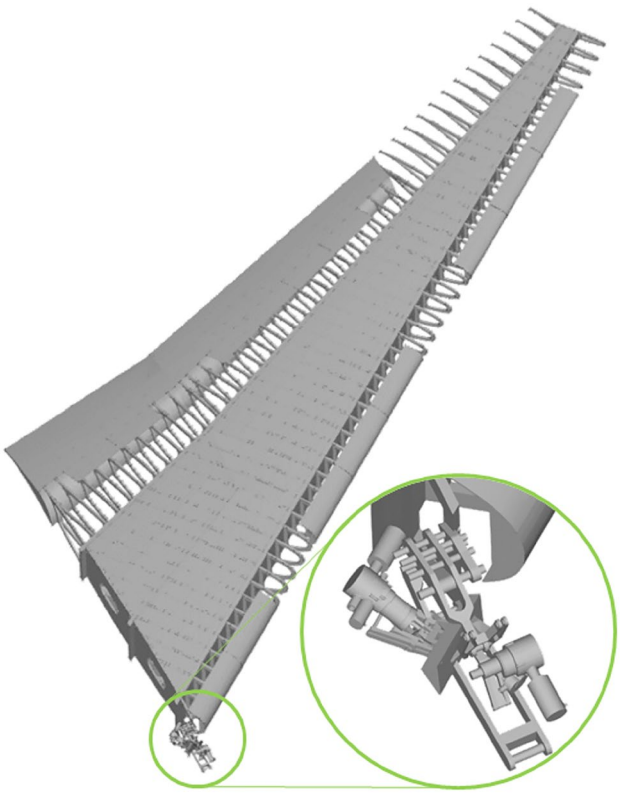

Fig. 7 Visualization of the non-packed geometries

of the design language, all flight control system components (previously shown in red) are identified and the corresponding geometrical graph instances are created and linked.

As an example for this specification, Fig. 5 shows the graphical representation of the nodes in the design graph in relation to a flap actuator specified from a Modelica element, with all connection interfaces and a link to its geometric data (existing component).

The assignment of the respective geometry data (as STEP file) is done automatically via matching unique identifiers of the component input data. The extended graph of the flight control system with all specified components is shown in Fig. 6.

From this geometric information, the design compiler can generate the 3-D view shown in Fig. 7. Note that the component geometries are only added but the positioning is not yet done. In this state, the components to be positioned by the packing algorithm are placed at the origin of the model. The STEP files linked to the components serve as input for the following packing operation, as well as the piping and routing in the subsequent process steps. The geometries used here are based on a academic model of an AIRBUS A380 wing with a wing span of about 80 m. Note that the geometry data are not adapted to the optimization results in the software stack. Since the creation of the data lies outside of

the optimization problem in the described process, the construction information is not accessible in the design graph, therefore, the models are not modifiable.

3.3 Packing

The developed packing algorithm is implemented as a design language, that can apply various heuristic optimization strategies to a problem of positioning multiple components in a 3-D assembly space while complying with boundary conditions such as:

- collision detection and collision avoidance
- contact or geometric membership of components
- proximity or distance of packed components
- go and no-go areas for placement of components in assembly space

The user can adapt the packing design language to the specific application by setting parameters and constraints in rules. Since the packing problem in this application is limited to a small number of components with great differences in their respective positional freedom, it is split into a pre-processing step, that positions the actuators and a package variation for the global optimization, that positions the electronic components according to control parameters. These parameters are accessible to the global optimizer to control design variation. This split reduces computing time, because it removes the necessity of positioning the actuators for each run, which have significantly fewer degrees of freedom, restricted by their respective control surface.

Pre-processing. The geometrical input for the pre-processed package is:

- wing structure (packing space)
- wing skin (packing space)
- 3 flaps (already positioned, treated as packing space)
- 5 flap actuators (to be positioned)
- 8 slats (already positioned, treated as packing space)
- 8 slat actuators (to be positioned)
- 6 spoilers (already positioned, treated as packing space)
- 6 spoiler actuators (to be positioned)

For the initial optimization of the actuator positions multiple objective functions can be applied. In this particular case uniform spatial distribution was selected. Before the packing process is started, all necessary requirements and constraints need to be integrated by the user as mathematical constraints in the rules of the design language. The geometrical information of the packing space and of all components to be positioned are read from the design graph by the design compiler. One boundary condition is the contact between the actuators and the associated flight control surfaces. This means that the packing needs to maintain a predefined distance and orientation to each slat actuator - slat pair, flap actuator - flap pair and spoiler actuator - spoiler pair. In consequence, the actuators are only movable along a predefined line, defined by the respective control surface. This has the effect of reducing the problem complexity, as all coordinates can be computed directly from one optimization parameter (i.e., optimize x and position $y(x)$, $z(x)$ and orientation $\psi(x)$, $\phi(x)$ and $\theta(x)$ are implicitly given). This way the degrees of freedom can be decreased by linear coupling of the component coordinates for data reduction and to save computation time. Further constraints are the collision avoidance of the components among each other and between each component and the packing space.

Optimization problem. For the integrated wing optimization in the PHAROS software stack, a geometrically less complex scenario is chosen for the packing problem, in order to show adaptability of successive piping and routing algorithm results. Here the control surfaces and actuators are fixed at their positions, while electronic boxes are introduced as components. The boxes represent the start and destination of the routing algorithm as well as obstacles for the routing and piping algorithms. The geometrical input for the packing optimization is:

- wing structure (packing space)
- wing skin (packing space)
- all flaps, slats and spoilers (packing space)
- all previously placed actuators (packing space)
- ACU box (to position)
- 2 generator boxes (to position)

The first degree of freedom of each box is the positioning inside of the chambers (numbered from 0 to 88) in the wing structure as shown in Fig. 8. The second degree of freedom is the height along the wing's z-axis as a discrete variable that can assume the values shown in Fig. 9:

- "bottom" (bottom edge of wing structure + 0.25 · structure height)
- "middle" (bottom edge of wing structure + 0.5 · structure height)
- "top" (bottom edge of wing structure + 0.75 · structure height)

Boundary conditions taken into account are the collision avoidance between the boxes and the packing space as well as the collision avoidance among each other. If a collision is detected, the solution is evaluated as not valid, and the overall software stack optimization loop is interrupted to save on computationally intensive routing and piping procedures. Subsequently, the next optimization loop with new variable values is executed.

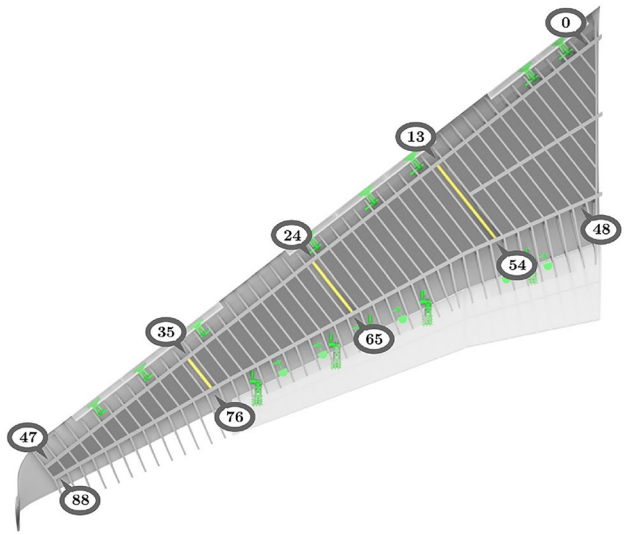

Fig. 8 Chamber numbering of the wing structure

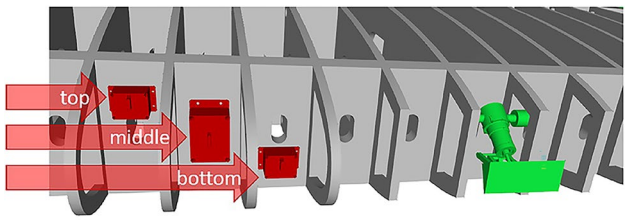

Fig. 9 Electronic boxes. From left to right: 1st generator box, ACU box, 2nd generator box

When a packing procedure is successfully performed, the calculated positions are stored as instances (nodes) in the design graph and the final packing geometry is exported as STEP file, which serves as input for the following piping and routing operations.

3.4 Piping

The developed piping algorithm can be used to generate pipes in arbitrarily complex installation spaces. A more detailed description of the piping algorithm can be found in [24]. Due to manufacturing constraints every pipe consists of a sequence of bends and straights in alternating order. Since the pipes are manufactured on bending machines, the bends have a machine-dependent bending radius. The following manufacturing constraints are recognized:

- minimal straight length between bends
- minimal straight length at beginning and end
- minimal opening angle per bend
- maximal opening angle per bend
- number of bend jaws

Table 1 Pipe types for slats, spoilers and flaps

Pipe type	Diameter (mm)	Bending radius (mm)
Slats	30	45
Spoilers	20	30
Flaps	40	60

The core of the piping algorithm is a stochastic *simulated annealing* optimization algorithm. For the optimization, an evaluation of the pipes is necessary. Therefore, an evaluation function was developed. This evaluation function considers the following criteria:

- number of bends
- sum of the opening angles of all bends
- distance to an estimated path through the geometry
- distance to a path in the center of the obstacle landscape
- straight length so no bending jaw is needed
- length of the pipe
- straight length at start/end, if a given length should be reached if possible
- distance to other pipes
- boundary violation (if allowed)

These optimization criteria can be weighted by the user. As a result, the user can influence the path of a pipe. In this part of the algorithm, additional design rules can be implemented to match given design rules of the industry.

The hydraulic pipe connections are defined in the Modelica system model and are extracted from there into the design graph. These pipe connections are routed in the wing. Three different pipe types are used, one for the slats, one for the spoilers and one for the flaps. The definition of these types is shown in Table 1.

The pipes are attached to the wing geometry with fixings. For each pipe type a fitting fixing type is used. The positioning of the fixings is rule-based, the decision for the number of fixings, necessary for each pipe, is executed by the design language. The order of the pipes is also determined by the design language. In Fig. 10, an overview of the resulting pipes is shown, while Fig. 11 shows a more detailed section, where the pipes are bypassing a positioned obstacle.

The pipe generation is a task with more constraints than the harness generation. There are more manufacturing constraints for pipes, e.g., sequences of discrete straights and bends with fixed radii and given minimal distances, while the harness must only ensure a given minimal bending radius. Also the electrical quality, the resistance of cables depends (for a given diameter) mainly on the length whereas the hydrodynamic quality of a pipe, e.g., pressure drop or flow uniformity, depends on the length and the number and

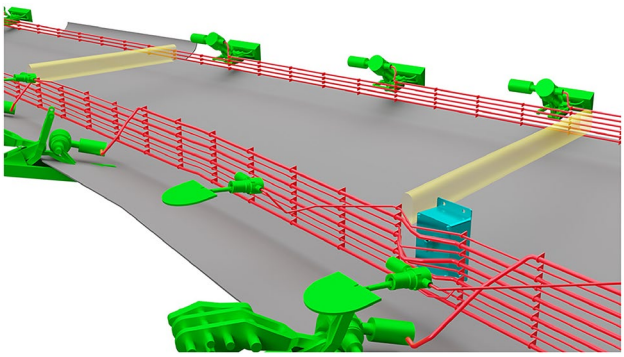

Fig. 10 Overview of the routed pipes (red) in the installation space

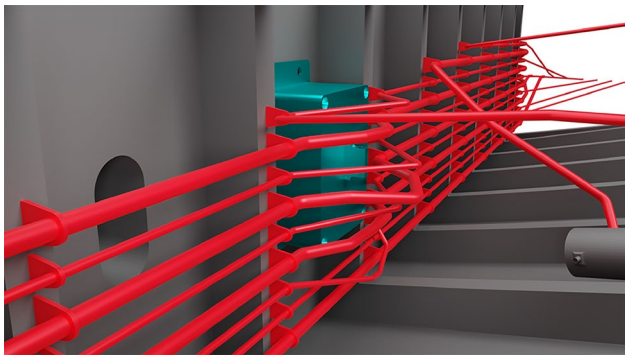

Fig. 11 Detailed section view of the routed pipes (red) around the obstacle ACU unit (teal)

kinds of bends. As a result of this, the pipe generation is preferably done in advance of the harness generation.

3.5 Harness routing

The routing algorithm has been developed to be applicable to various routing problems in complex geometrical environments. It can take the following constraints into account:

- physical properties of cables such as minimum bending radius,
- placement and mounting restrictions,
- compliance to electro-magnetic interference and
- clearance to hazardous areas

The routing design language itself uses several graph algorithms for pathfinding and collision detection. It already has a high level of maturity and has been applied and tested in research and industrial projects for multiple engineering domains [25–27]. The wire harness routing process can be divided into the following steps [27]:

- data import of the master data of all electrical and non-electrical components that interact with the wire harness, geometrical components and electrical schematics,
- definition of a pathfinding sequence for the wire harness topology
- application of a modified A*-algorithm to find the optimal individual paths in sequence
- accumulation of the resulting paths into one combined network topology.
- routing of the individual wires from the harness topology by minimizing cable length between the component connectors
- a multi-body simulation to adapt the rectangular path segments to rounded paths, that takes the stiffness and radii of the segments into account
- storing of the results in the design graph and export of 3-D-CAD and XML files

In this application, the previously referenced geometries of the wing, the packaged component geometry and the pipes are all read from the design graph and used as obstacles for the harness routing process. Analogous to the piping, the connection list of the connected electrical components, originally defined in the Modelica system file, is read from the design graph. The aforementioned routing process is performed to find the shortest path between the electrical connectors through the assembly space and generate the corresponding wire harness.

The harness is divided into two independent units, a power and a data unit. The power harness connecting all power connections is depicted in dark purple and the data harness connecting all data connections is depicted in yellow. Figure 12 shows an overview of the routed power and data harness added to the view of Fig. 10. Figure 13 shows the detailed view of the installation space section with the power and data harness. Figures 14 and 15 show the whole wing with a feasible design solution of the packaged geometries, the routed pipes and the harness as an example.

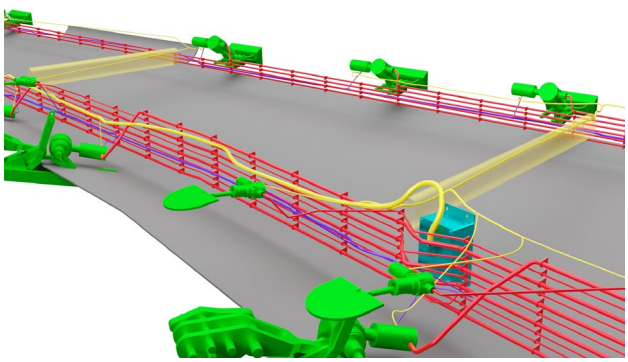

Fig. 12 Overview of the routed data (yellow) and power (dark purple) harnesses in the installation space

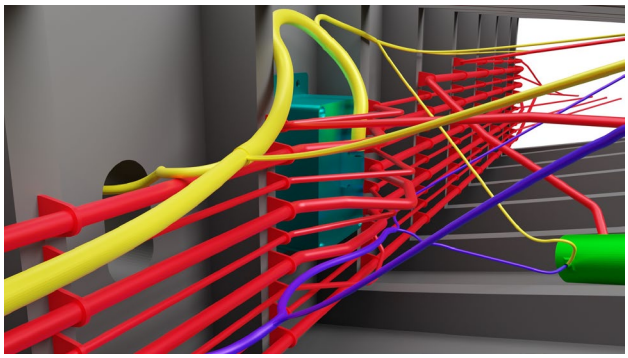

Fig. 13 Detailed section view of the routed data (yellow) and power (dark purple) harness around the obstacles ACU unit (teal) and piping (red)

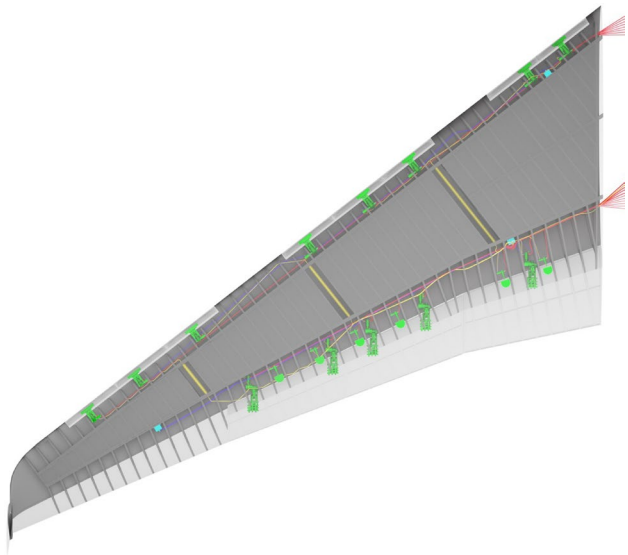

Fig. 14 Top view of the wing with packed geometries (actuators in green, electronic boxes in teal), pipes (red) and data (yellow) and power (dark purple) harness

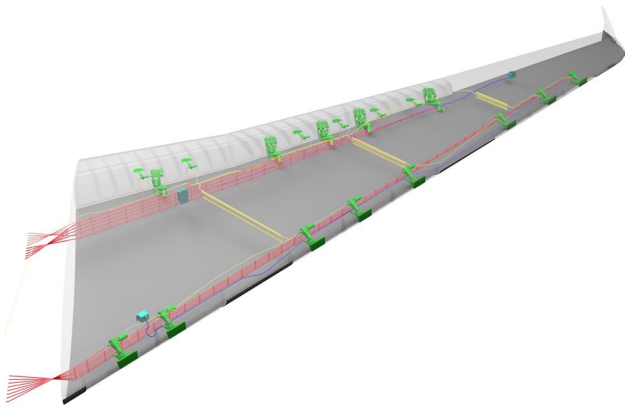

Fig. 15 Front view of the wing with packed geometries (actuators in green, electronic boxes in teal), pipes (red) and data (yellow) and power (dark purple) harness

3.6 Optimization

At this point the new software platform has created a feasible result for the wing section architecture packing, piping and routing problem. As Fig. 1 shows in step (11) and (12), the software platform embeds the algorithms in a global optimization loop, shown in more detail in Fig. 16. This is achieved by combining the software platforms of the *Design Compiler 43*® and *OPTIMUS*® into the so-called PHAROS software stack.

OPTIMUS® specializes in optimizing product design solutions by managing a parametric design space exploration. Through an intelligent design parameter modification strategy to the input parameters of the individual algorithms, the system can be optimized globally. *OPTIMUS*® manages the execution of the presented design languages in *Design Compiler 43*®.

3.6.1 System optimization

The software stack is designed based on a multi-domain problem, where each domain has its own target values. For the routing problem, the optimal design solution has the shortest total wire harness length and for the piping problem, the optimal solution has the shortest total pipe length or the lowest number of bends. For the applied packing problem, each valid packing solution is rated equally. Because the solution of the packing also defines starting and end points for both wire harness and piping, it represents the biggest influencing factor for the overall system. To allow the software stack to cover a large part of the design space and give a broad understanding of the design problem and system sensitivities, the global optimization strategy of the software stack, is chosen to be a design variation and design space exploration. The optimization parameters have been reduced to packing control parameters detailed in

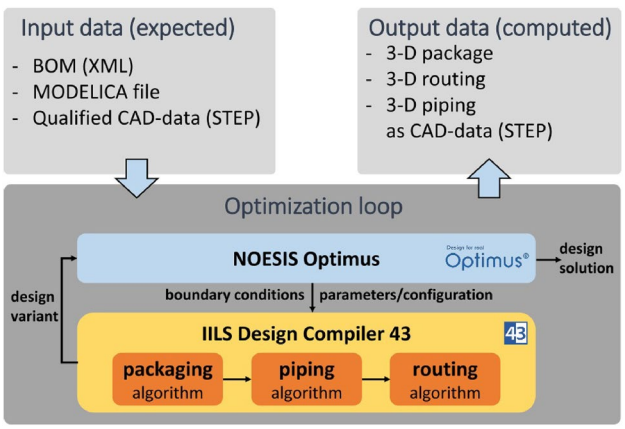

Fig. 16 Visualization of the integrated software stack of *Design Compiler 43*® and *OPTIMUS*®

Sect. 3.3 optimization problem. Still, each run contains the individual sub-system optimizations of the packing, piping and routing algorithms. Because of the natural integration capability of design languages, changes in the input of one algorithm encoded in one design language may be smoothly propagated through the complete algorithm chain encoded into other design languages, and finally results in a design solution with a consistent model in all different domains.

In a more target-oriented design approach for industrial design solutions, the optimization strategy can be easily adapted to optimize towards specific target functions. *OPTIMUS*® offers many different optimization strategies and tools for a more targeted optimization strategy [28]. For reference, another example of the optimization capabilities of *OPTIMUS*® applied to a multi-disciplinary analysis and optimization (MDAO) problem can be found in [29].

3.6.2 Harness optimization

Because of the considerably shorter run time of the routing design language than of the piping design language, a separate optimization of the harness without the piping is performed in another configuration of the software stack. For the harness optimization, the two power harnesses and the data harness can be regarded independently. Since the power harness is divided into two separate parts for the leading and the trailing edge of the wing, they are optimized independently as well.

The data harness connects components of the leading and the trailing edge of the wing. As an additional optimization parameter different variations for wing passages are introduced. The seven different passage variants are shown in Fig. 17. The colored lines show the permitted passages (meaning routing space) for the routing algorithm for each variant. They along with the variable ACU positions, defined in Sect. 3.3, lead to a total of 623 possible combinations.

To reduce computing time, the optimizations are performed without respecting the obstacle geometries. However, the passage configuration is still adhered to by activating different routing paths.

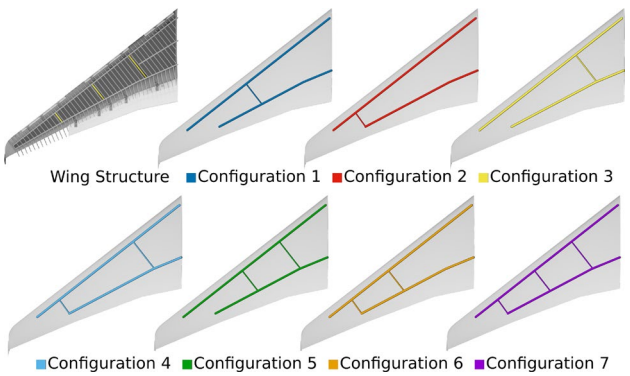

Fig. 17 Harness passage configurations of the wing compare view to Fig. 14

4 Results

The results for two separate software stack configurations are presented. First, the pre-optimizations of the routing without the piping, described in Sect. 3.6.2, are performed.

4.1 Harness optimization results

Power harness. In Fig. 18, the length of the leading edge power harness based on different positions for the corresponding generator is shown. This separate harness algorithm is performed without obstacles, shown in purple, and with all obstacles of the wing geometry, shown in teal. The results indicate the shortest and therefore, optimal harness length is achieved at the positions in the middle of the wing (positions 15–25, see Fig. 8).

The results for the trailing edge of the wing in Fig. 19 show a similar optimal position for the trailing edge generator at middle positions (positions 58–68), with a clearer optimum in the middle positions.

Both optimizations show that the results with and without respecting the obstacle geometries do not differ substantially. This is because the two power harnesses have sufficient space in their surrounding geometry not to be diverted from the shortest path by obstacles. This motivated the decision, to optimize the majority of the data harness variants without obstacles.

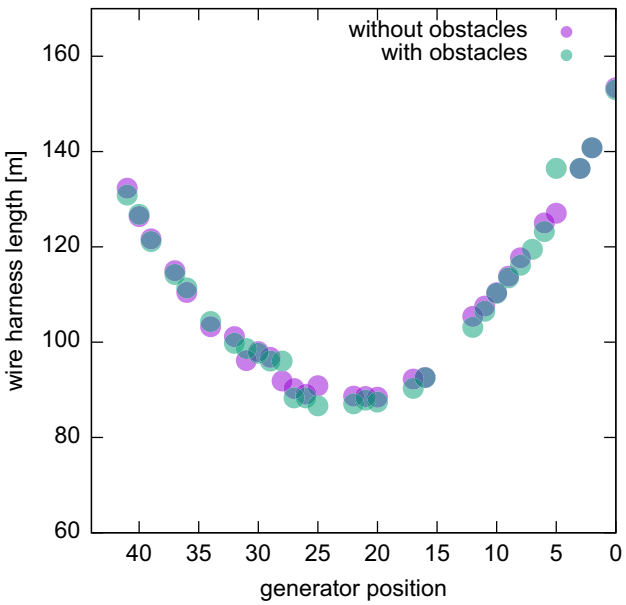

Fig. 18 Power harness leading edge

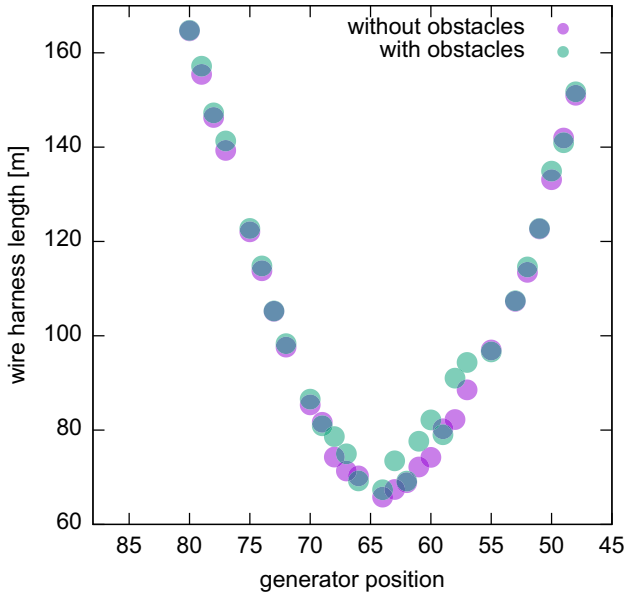

Fig. 19 Power harness trailing edge

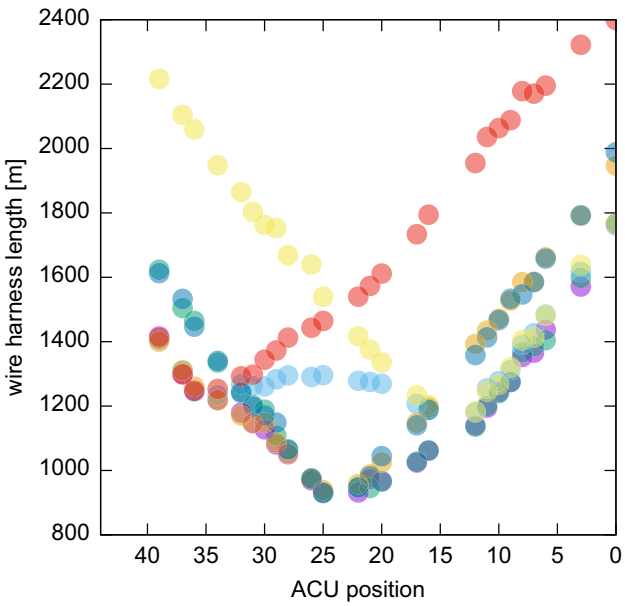

Fig. 20 Data harness leading edge

Data harness. To reduce computing time, the optimizations are performed without respecting the obstacle geometries. However, the passage configuration, shown in Fig. 17, is still adhered to by activating different routing paths.

The results for the placement of the ACU in the leading edge are shown in Fig. 20 and the results for a placement in the trailing edge are shown in Fig. 21. In both cases, the colors of the data points correspond to the applied passage configuration colors in Fig. 17.

The data indicate a high sensitivity of the wire harness length to the ACU position. However, the degree of the sensitivity, represented by the gradient of the connecting curve of the data points, differs for the passage configuration variants.

The data points of the diagrams 20 and 21 are combined in the box plots in Fig. 22 for legibility and an overview of the sensitivities of each passage configuration. The colors of the box plot correspond to the applied passage configuration colors in Fig. 17. Each box plot represents all optimization results of the respectively colored passage configuration. The box height comprises the values of the middle 50 percent, divided into the upper and lower quartile, with the median line in between. The whiskers comprise the whole range of values, excluding the outliers, which are depicted as points.

The best median values are achieved with passage configurations 5, 6 and 7, which have at least one additional passage in combination with the passage in the middle of the wing. Only a singular passage at the fuselage or the wing tip (configuration 2 and 3) lead to the longest harness lengths.

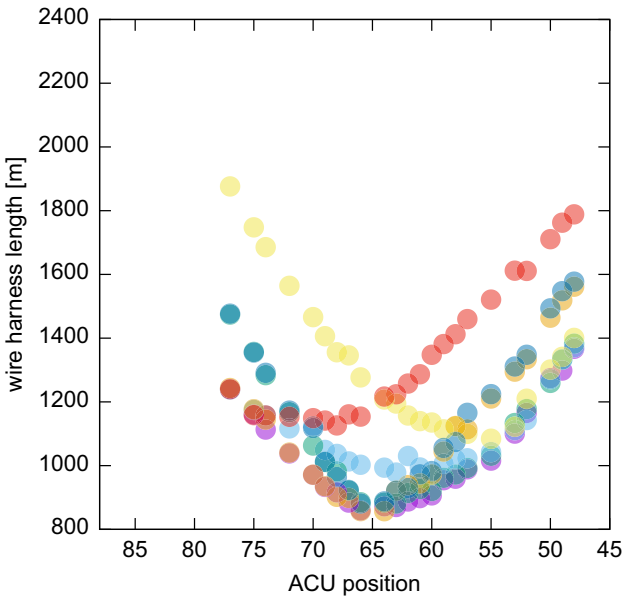

Fig. 21 Data harness trailing edge

This is likely caused by the actuator distribution closer to the fuselage. The divergence of the values decreases with increasing number of passages.

Possible weight reductions for the harness. From the three individual harness optimizations it is evident that choosing an optimal configuration can lead to a significant reduction in wire length, and therefore harness weight. For the power harness at the leading edge, the length reduction

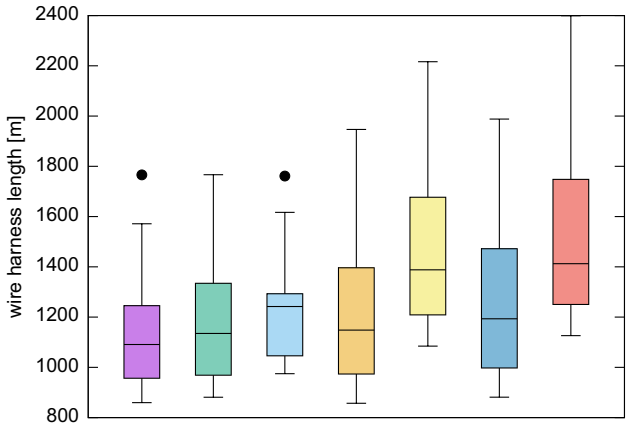

Fig. 22 Data harness box plot for passage configurations 7, 5, 4, 6, 3, 1, 2 from left to right

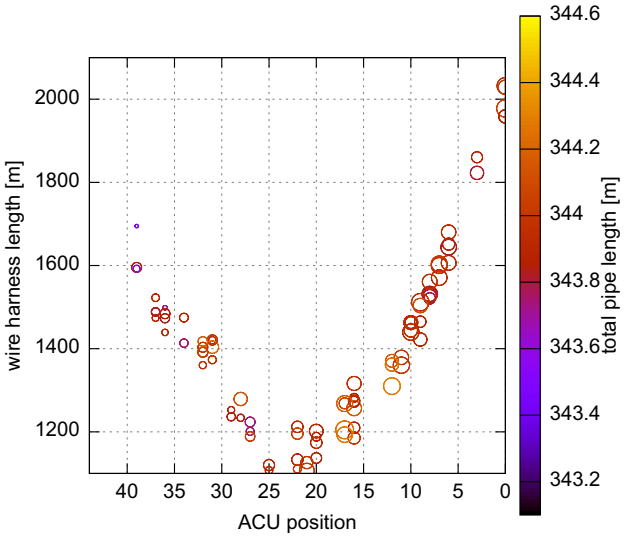

Fig. 23 Harness and piping for varied ACU position (leading edge)

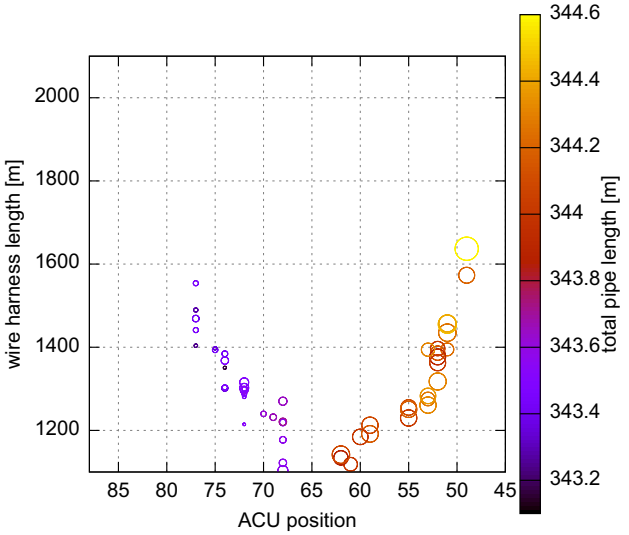

Fig. 24 Harness and piping for varied ACU position (trailing edge)

from the worst to the best generator position is around 40 % while the harness of the trailing edge offers the possibility for a 60 % reduction.

For the data harness, the possible savings including all passage configurations are around 65 % while the savings for a specific passage configuration vary between 40 and 60 %.

4.2 Harness and pipe optimization results

The overall optimization of piping and routing was done with all passages open (configuration 7). The run time of the average optimization run inside the PHAROS software stack is about 80 min on a desktop computer with an AMD Ryzen 7 3700x processor and 64 GB RAM. A total of 135 successful runs were done.

The results of all successful runs are shown in Figs. 23 and 25 at the leading edge as well as Figs. 24 and 26 at the trailing edge. The data are presented in a diagram of the varied ACU position over the wire harness length, while the color represents the total pipe length and the circle diameter represents the number of bends of the pipes.

The diagrams show that a trade off between the harness length, the total pipe length and the number of bends can be done. The farther the ACU box is moved to the wing tip, the shorter the pipes get and the fewer number of bends are necessary. On the other hand, if the box is placed near the wingtip the wire length increases.

In Figs. 25 and 26, the influence of the two generator box positions is shown. As there is no pattern visible in the diagrams, it can be concluded that the generator positions are no major design drivers. This is caused by the fact that when assessing the total harness length, the power harness has a smaller effect, as it is about an order of magnitude shorter than the data harness. A separate optimization of the power harness could show optimal generator positions, but would have little impact on total harness weight in this model application.

For Table 2, the results with the shortest total pipe length, the smallest number of bends and the shortest wire length are selected. For all three results, the ACU box is placed on the trailing edge in a range between the middle of the wing and the wing tip.

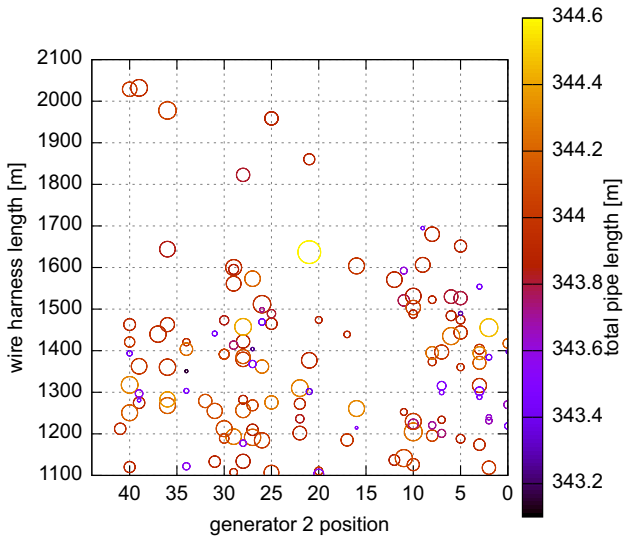

Fig. 25 Harness and piping for varied generator position (leading edge)

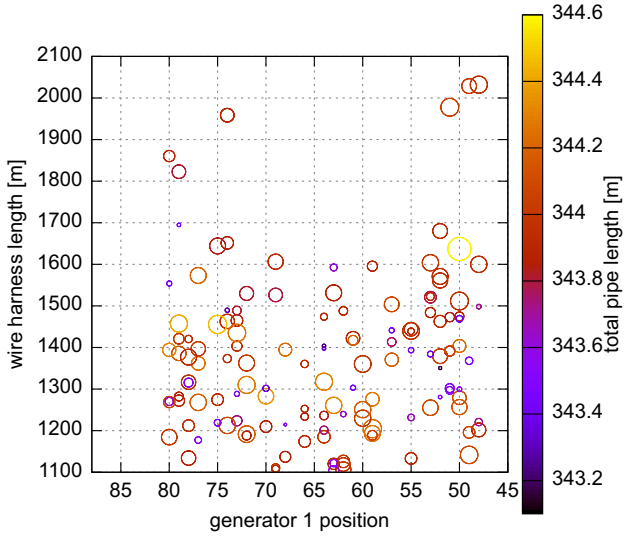

Fig. 26 Harness and piping for varied generator position (trailing edge)

Table 2 Selected results of the overall optimization

ACU position	Total pipe length (m)	Number of bends	Wire harness length (m)
74	***343.3***	158	1350.7
72	343.5	***157***	1214.6
68	343.6	176	***1103.3***

5 Discussion

The presented software stack formalizes and implements the linking of different engineering domains of an aircraft flight control system into one model. This ensures data consistency between the different domain models and allows for computational exploration and analysis of the integrated design space.

The initial formalization of specific design vocabulary and rules has to be done manually by domain experts and can be resource intensive, as it drives the quality of the model, and therefore the validity of the results. However, once general rules are formulated, they can be reused and built upon in future work. General tasks like the packing, piping and routing algorithms, are, therefore, encapsulated into independent plug-ins to be applied to various problems. Specific design languages are easily adapted to different design tasks, as long as they do not require a fundamental change of the problem type. This initial front loading of creating a design language that models the relevant design problem sufficiently well, can pay off quickly by the enormous potential of the following automated design, simulation and optimization.

The results of the software stack show great potential for optimization of a design, with possible wire harness weight reduction of up to 60 percent. In the posed problem, the total piping length cannot be reduced significantly, but the number of bends by up to 34 percent. Concrete optimization results highly depend on the specific problem.

The application of the presented model and optimizations are based on a simplified academic data set and problem definition. This may obfuscate difficulties, arising only in the application of realistic, more complex use cases, unknown to the authors. Such a use case would most likely include some of the following characteristics:

- Larger system model with more and more different components.
- More complex CAD geometry model.
- More constraints on the system, i.e. smaller installation space, consideration of the accessibility and maintainability.
- Consideration of component mounting and structural analysis as an additional domain.

In addition, the assumed problem definition, leaves room for improvement for the optimization of a design solution, since neither an exact target function is defined, nor a unique continuous space of input parameters is available. The definition of a target function that weighs the criteria according to a design target, would allow *OPTIMUS*® to find an

optimized solution, but is highly dependent on the specific design problem.

The application of heuristic algorithms in the multiple domains of packing, piping and harness routing means that the pure formalization of the problem does not provide a mathematically unambiguously defined solution for an optimal result. However, the integrated computational approach allows for a much faster computer-aided approximation of a satisfactory result and a much wider overview of the system design sensitivities, by computing valid design solutions as expressions of the design language. The formalization of the problem into a model- and rule-based design language simplify the extension of the design problem by any of the stated characteristics of a more practical use case. Adaption to larger and more complex input data is even possible without changes to the software stack, by simply changing the imported files of the 1-D system models and the corresponding 3-D CAD models and adjusting the design language parameters to them.

The achieved run time performance of on average 1 h 20 min on an AMD Ryzen 7 3700x processor and 64 GB RAM for one complete run, suggest that there is still a large margin to computational limits, when scaling up the process to industrial-sized problems.

6 Summary and outlook

Summary The developed software platform for an automated model-based system engineering design process was demonstrated on the optimal, integrated packing, piping and routing of a flight control system inside of an aircraft wing. Figure 1 shows the internal interactions and execution sequence among the algorithms which has been reproduced from the grant application [1]. The geometry results of a run of the software platform are shown in Figs. 14 and 15.

Overall the project results of the packing, piping and routing algorithms in its implementation inside the global optimization show great potential for generating and analyzing a far greater number of possible design variants compared to the time and cost of the manual design process. Through the connected algorithmic approach a conformity with the design specifications as well as a consistency between the different model domains is ensured within the model.

The upfront effort of formalizing the design specifications and steps into a design language paid off, by speeding up the design process by several orders of magnitude to a run time performance of hours compared to weeks for the manual process.

Through the utilization of such automation solutions, like the presented software stack, the design process is lifted to a higher level. This means that the costly manual engineering work can be focused on top level design choices, while laborious routine tasks are performed automatically. This enables the engineers to have significant simulation-based knowledge about the product in early design phases, which can lead to improvements to the design. This was demonstrated here with a optimal physical wing architecture for significant piping and harness weight reductions.

Outlook Since industrial CAD models typically come along with publication restrictions in the form of a non-disclosure agreement (NDA), the PHAROS software stack for the reason of confidentiality uses self-made academic data of an aircraft wing section. Naturally, the degree of complexity is much less than for a real industrial model, for both CAD geometry and Modelica system data. However, the functionality and effectiveness of the algorithms can already be very well tested. Switching from the academic to the industrial CAD-model as the design space raises the question of the *scalability* of the algorithms to industrial problem size.

The project partner AIRBUS also provided a more complex model of an aircraft main landing gear bay for this reason. The piping algorithm is applied to the piping of the enclosed hydraulics and other piping problems. Although this problem is currently confined to a part aspect of the presented software platform, these data are used to mature the algorithm in order to deal with the arising problem of scaling in both size and detail.

It represents currently the next step towards the demonstration of the industrial applicability of the shown information processing of graph-based design languages for automated system installation. The publication of this industrial project result appeared in [24].

Acknowledgements This project has received funding from the Clean Sky 2 Joint Undertaking (JU) under grant agreement No 865044. The JU receives support from the European Union's Horizon 2020 research and innovation program and the Clean Sky 2 JU members other than the Union.

The PHAROS consortium members also want to express their gratitude towards the topic manager Ana Garcia Garriga from UTRC, Ireland, and the JU CS2 project administration officer Jaime Perez-de-Diego, Brussels, for their smooth management and continuous support throughout our project. The authors thank our ERASMUS exchange students Fergus Hall, Scott Bell, Gregor Cunningham and Ross O'Hare from Strathclyde University, UK, for their joint effort and contributions to all the parts and the assembly of the academic CAD-model in the PHAROS project.

Funding Open Access funding enabled and organized by Projekt DEAL.

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Rudolph, S.: System, physical architecture optimization. (PHAROS). Submitted,: H2020 CS2 Grant Proposal. I (Confidential, Unpublished), Part B (2019)
2. CleanSky2 Website. Physical Architecture Optimi-zation System (PHAROS). Project Description, https://cordis.europa.eu/project/id/865044, Last Access February 27, (2022)
3. CleanAviation Media. Physical Architecture Optimization System (PHAROS). PHAROS Project Article, https://www.clean-aviation.eu/media/news/simplifying-the-3d-packaging-3d-piping-and-3d-routing-sequence-for-physical-system-architecture, Last Access February 27, (2022)
4. Groß, J., Rudolph, S.: Modeling graph-based satellite design languages. Aerosp. Sci. Technol. **49**, 11 (2015). https://doi.org/10.1016/j.ast.2015.11.026
5. Walter, B., Kaiser, D., Rudolph, S.: From manual to machine-executable model-based systems engineering via graph-based design languages. In *MODELSWARD*, pages 201–208, (2019)
6. Riestenpatt, M., Richter, G., Rudolph, S.: A scientific discourse on creativity and innovation in the formal context of graph-based design languages. In *Proceedings of the 13th Anniversary Heron Island Conference Workshop on Computational and Cognitive Models of Creative Design (HI '19), Heron Island, QLD, Australia*, pages 15–18, (2019)
7. Kroo, I., Altus, S., Braun, R., Gage, P., Sobieski, I.: Multidisciplinary optimization methods for aircraft preliminary design. In *5th symposium on multidisciplinary analysis and optimization*, page 4325, (1994)
8. Perez, R., Liu, H., Behdinan, K.: Evaluation of multidisciplinary optimization approaches for aircraft conceptual design. In *10th AIAA/ISSMO multidisciplinary analysis and optimization conference*, page 4537, (2004)
9. Raymer, D.: *Enhancing aircraft conceptual design using multidisciplinary optimization*. PhD thesis, Institutionen för flygteknik, (2002)
10. Tfaily, A., Liscouet-Hanke, S., Esdras, G.: Parametric 3d modeling for integration of aircraft systems in conceptual design. In *Canadian Aeronautics and Space Institute Conference*, pages 1–10, (2015)
11. Coffman, E.G., Csirik, János, J., David, S., Woeginger, G.J.: An introduction to bin packing. *Bibliographie. Siehe www. inf. u-szeged. hu/' csirik*, (2004)
12. Venter, G., Sobieszczanski-Sobieski, J.: Multidisciplinary optimization of a transport aircraft wing using particle swarm optimization. Struct. Multidiscip. Optim. **26**(1), 121–131 (2004)
13. Peddada, S.R.T., James, K.A., Allison, J.T.: A novel two-stage design framework for two-dimensional spatial packing of interconnected components. J Mech Des **143**, 3 (2021)
14. Jessee, A., Peddada, S.R.T., Lohan, D.J., Allison, J.T., James, K.A.: Simultaneous packing and routing optimization using geometric projection. J. Mech. Des. **142**(11), 111702 (2020)
15. Wang, H., Zhao, C., Yan, W., Feng, X.: Three-dimensional multi-pipe route optimization based on genetic algorithms. In *International Conference on Programming Languages for Manufacturing*, pages 177–183. Springer, (2006)
16. Liu, Q., Tang, Z., Liu, H., Yu, J., Ma, H., Yang, Y.: Integrated optimization of pipe routing and clamp layout for aeroengine using improved moalo. Int. J. Aerosp. Eng. **2**, 2 (2021)
17. Alex Jessee, P., Satya, R.T., Lohan, D.J., Allison, J.T., James, K.A.: Simultaneous packing and routing optimization using geometric projection. J. Mech. Des. **142**(11), 111702 (2020)
18. Thantulage, G.I.F.: Ant colony optimization based simulation of 3d automatic hose/pipe routing. Phd thesis, School of Engineering and Design, Brunel University, UK, March (2009)
19. Van der Velden, C., Bil, C.Y., Xinghuo, S.A.: An intelligent system for routing automation. In *Innovative Production Machines and Systems, Virtual Conference*, pages 2–13, (2008)
20. Conru, A.B.: A genetic approach to the cable harness routing problem. In *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, pages 200–205 vol.1, Jun (1994)
21. Park, H., Lee, S.H., Cutkosky, M.R.: Computational support for concurrent engineering of cable harnesses. Technical report, Center for Design Research CDR Technical Report No. 19920219, San Francisco, CA, February . Submitted in Computers in Engineering Conference (1992)
22. Ng, F.M., Ritchie, J.M., Simmons, J.E.L., Dewar, R.G.: Designing cable harness assemblies in virtual environments. J. Mater. Process. Technol. **107**(1), 37–43 (2000)
23. Zhu, Z.: *Automatic 3D Routing for the Physical Design of Electrical Wiring Interconnection Systems for Aircraft*. doctoral thesis, TU Delft, (2016)
24. Neumaier, M., Kranemann, S., Kazmeier, B., Rudolph, S.: Fully automated piping in an Airbus A320 landing gear bay using graph-based design languages. In *IOP Conference Series: Materials Science and Engineering, Vol. 1226, International Conference on Innovation in Aviation & Space to the Satisfaction of the European Citizens (11th EASN 2021) 01/09/2021 - 03/09/2021 Online*, (2022)
25. Weil, R.: Automatisierte Verkabelung des Kleinsatelliten Flying Laptop. Diplomarbeit, Institut für Raumfahrtsysteme, Universität Stuttgart, Juni (2012)
26. Rudolph, S., Hess, S., Beichter, J., Motzer, M., Eheim, M.: Architectural analysis of complex systems with graph-based design languages. In *4th International Workshop on Aircraft System Technologies (AST 2013), Hamburg, April*, pages 23–24, (2013)
27. Eheim, M., Kaiser, D., Weil, R.: On automation along the automotive wire harness value chain. *Stuttgart Conference on Automotive Production 2020*, November (2020)
28. Solutions, N.: Optimus 2021.1—users's manual. Leuven, Belgium, April (2021)
29. Lefebvre, T., Bartoli, N., Dubreuil, S., Panzeri, M., Lombardi, R., Della Vecchia, P., Stingo, L., Nicolosi, F., De Marco, A., Ciampa, P.D., et al.: Enhancing optimization capabilities using the agile collaborative mdo framework with application to wing and nacelle design. Progr. Aerosp. Sci. **119**, 100649 (2020)