Paper published as:
Rudolph, S., Beichter, J., Eheim, M., Hess, S., Motzer, M. and Weil, R.:
"On Multi-Disciplinary Architectural Synthesis and Analysis of Complex Systems with Graph-based Design Languages"
62. Deutscher Luft- und Raumfahrtkongress (DGLR 2013), Stuttgart, September 10-12, 2013.

On Multi-Disciplinary Architectural Synthesis and Analysis of Complex Systems with Graph-based Design Languages

S. Rudolph[a], J. Beichter[a], M. Eheim[a], S. Hess[a], M. Motzer[a], R. Weil[a]

[a] *Institut für Statik und Dynamik der Luft- und Raumfahrtkonstruktionen*
Universität Stuttgart,Pfaffenwaldring 31, 70569 Stuttgart, Deutschland

Abstract

Architectural synthesis of complex systems such as aircraft cabin design requires much engineering insight into the multi-disciplinary couplings of the sub-systems and their interconnecting networks (e.g. power, fluids, information, etc.). However, the potential of new architectural solutions in a complex systems environment is difficult to assess, since any assessment will require the elaboration of several architectural alternatives which is manually not feasible using conventional techniques. As a result, the potential of new architectures is never fully explored. Graph-based design languages have been conceived to solve the complex system investigation problem and are therefore able to overcome this problem. The machine-executed compilation of a graph-based design language in a design compiler allows very easily topological and parametrical design variations, relieves the design engineering teams from the tedious routine works of manual model generation and improves the re-use of design knowledge by re-use of design vocabulary and rules.

1. Introduction

The problem of multi-disciplinary architectural synthesis and analysis of complex systems, or, in a more philosophical formulation, the lacking capability of *"system integration is currently considered as the largest obstacle to an effective design of complex systems"* [1]. This *"is due primarily to the lack of a solid scientific (and methodological) foundation for the subject"* [1] and, as a direct consequence, a lack of appropriate and well adapted engineering design software tools.

The current *"lack of a solid understanding of a science of system integration is not due to (some active kind of ignorance or) neglect, but rather due to its (scientific) difficulty"* [1] and rigourness. Because of that many if not *"most of the large system builders*[1] *have therefore currently given up on any design science or engineering design methodology for system integration and treat it instead as a management problem"* [1].

While treating physics problems in management manner doesn't sound like the most perfect problem approach[2] anyway and by looking at both the engineering impact and financial scale of the problem, it seems quite evident that the problem of multi-disciplinary system integration is currently one of the most under-estimated and under-valued problems[3] in complex system design [1]. At the same time it is difficult to find another problem which has bigger impact on engineered systems [1].

Graph-based design languages [2, 3] provide this long sought-after solution approach to the problem of system integration with special emphasis on multi-disciplinary architectural synthesis and analysis of complex systems such as aircraft cabins.

[1] Large system builders use many software tools in their design flow which have been developed by a third party, i.e. embody a design theory and methodology conceived elsewhere. There exists therefore a latent danger that a third party tool doesn't match perfectly the real needs.

[2] It is evident that treating physics problems as a management problems will cause severe trouble, since the problem may only be solved if managment hits the physical solution. As a consequence, it is highly probable that design cycles with major rework will occur because the solution can only be found incrementally by iterations.

[3] If a company was able to make a fair and unbiased assessment of their financial losses due to ineffective processes, data and tool inconsistencies, rework, late deliveries, etc., it would be clear that even a serious investment into a system design methodology and associated software tools belongs to their most profitable investment opportunities.

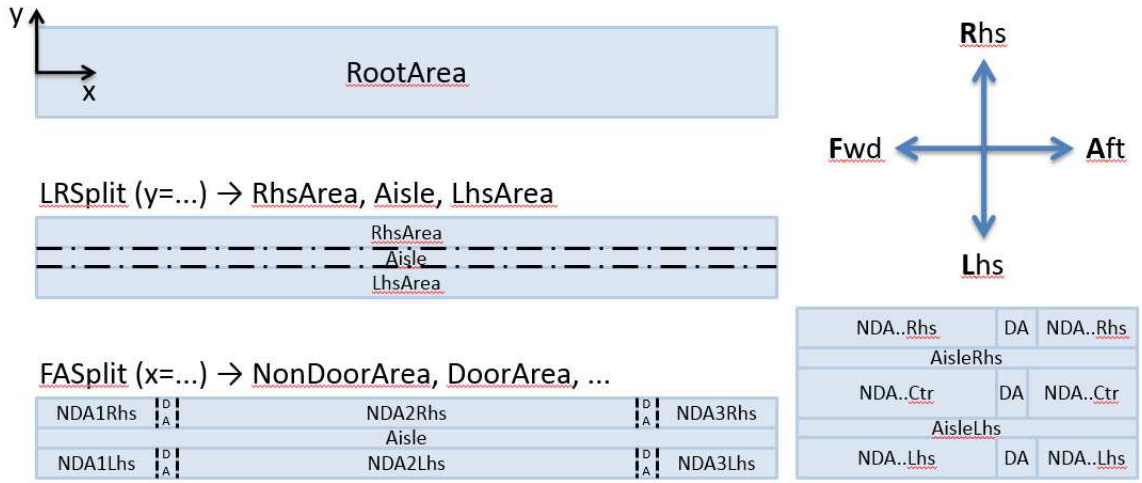

Figure 1: Cabin definitions

As stated above, these graph-based design languages may come only at the price of an underlying new design science and methodology, no more no less. In this respect, design languages resemble more a revolution than an evolution of design knowledge representation and processing. In order to completely benefit from the approach, the whole traditional industrial setting of *methods, processes and tools* must be sincerely reconsidered and newly balanced.

Graph-based design languages are a new way of supporting the activity of engineering design [4]. They are inspired by natural human languages, in which the *vocabulary* (i.e. the building blocks) and the *rules* (i.e. the building laws) define a so-called language grammar. This means that any correct sentence in this language (i.e. a permissible vocabulary combination) represents a valid engineering product variant.

The compilation of graph-based design languages in a machine called *design compiler* relieves the design engineering teams by automatic model generation from tedious routine works, allows know-how re-use of design knowledge by re-use of design rules and eases topological and parametrical product variations [4]. Graph-based design languages on the basis of the *Unified Modeling Language (UML)* possess a well distinct information processing concept in comparison to other known current approaches.

A big part of the increase in productivity, higher model quality and shorter time-to-market stems from modeling and processing of the design knowledge on a higher level of abstraction then done previously using *model-to model transformations*. The mapping of this abstract level into a specific data format is provided by compiler plug-ins using *model-to-text transformations*. This avoids an intermixing of the per se pure design knowledge with vendor-specific representation dependencies.

2. Aircraft cabin design language

Inside an aircraft design process based on graph based design languages, a cabin layout design language has been developed in order to create different cabin layouts based on different input data and requirements. At first, the vocabulary stored in the class diagram and the basic principle of this cabin layout approach is introduced. This is followed by a description of the design process, which is modeled with rules inside an activity diagram.

The goal of the aircraft cabin layout design language is the automatic creation of different cabin layouts depending on changing requirements and input values such as e.g. the ratio passenger (pax) per lavatory or pax per trolley and many more.

The dimensions and boundaries of the cabin are defined by the preprocessed aircraft structure creation. From this structure a root area can be extracted which is the basis for the cabin layout. The main principle in the cabin layout design language is the division of the root area provided for the cabin layout and the allocation of the particular areas to a specific type like seat, trolley area, lavatory area, aisle and others. Figure 1 shows the principle and the first splitting steps of this approach.

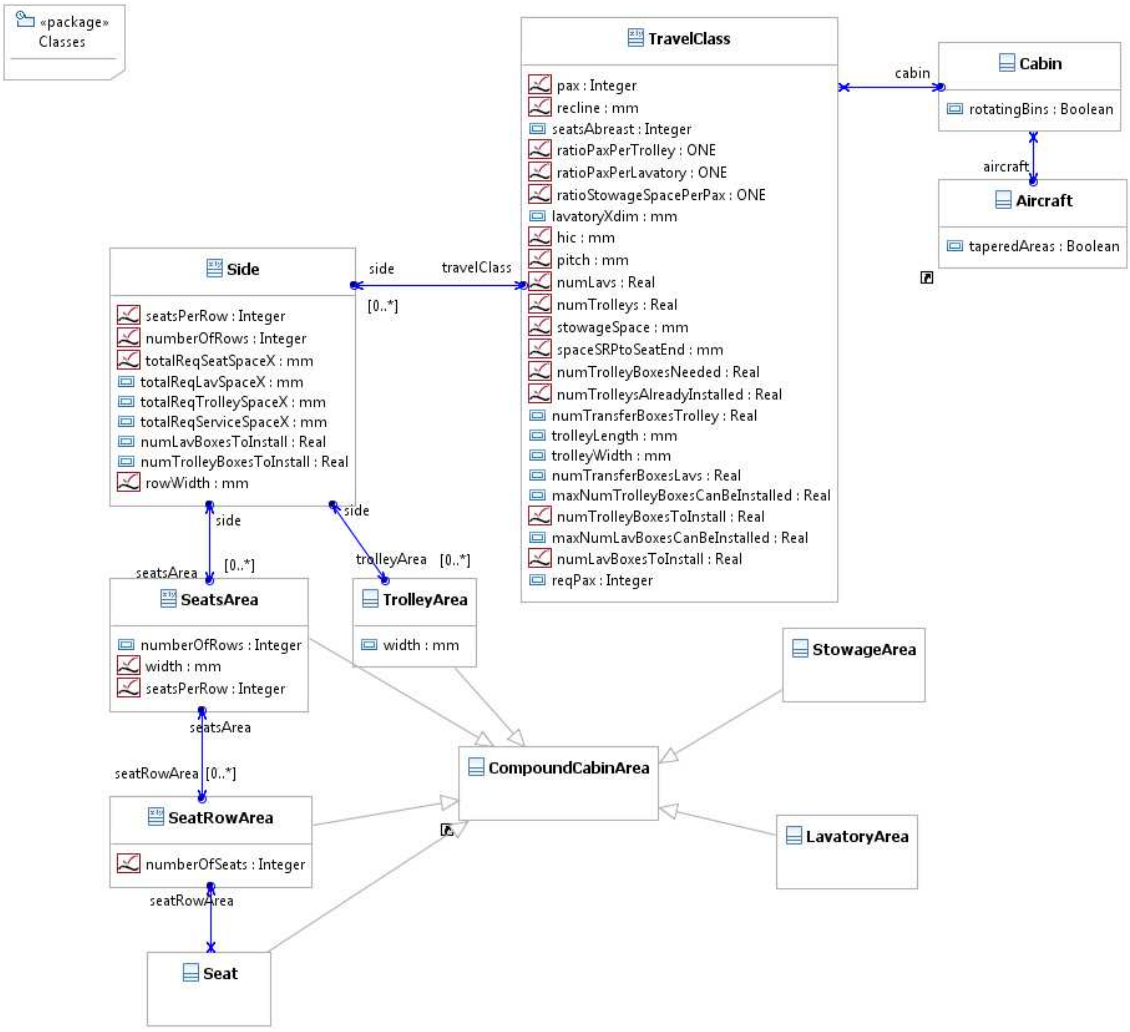

Figure 2: Cabin class diagram

This principle can again be found in the class diagram in which the vocabulary of the cabin design language is represented, see figure 2. The classes inherit from the linked class `CompoundCabinArea`, which stands for the particular areas of the layout. This class is from the abstract upstream floor plan definition. The arrow in the lower left corner shows that the class element is a shortcut to a class in another project or class diagram. The classes of the different areas in figure 2 inherit from this class. These inheritances are the linkage to the floor plan design language. This abstract language provides the described functionality of dividing the areas into smaller areas.

2.1. Vocabulary

The various classes `Cabin`, `TravelClass`, `Side`, `SeatsArea`, `SeatRowArea`, `Seat`, `TrolleyArea`, `StowageArea` and `LavatoryArea` are the vocables of the cabin design language. This vocabulary can be instantiated e.g. writing `BusinessClass : TravelClass` means that `BusinessClass` is an instance of `TravelClass`. A description of the vocabulary follows, defining the vocabulary of the cabin layout design language:

- `TravelClass`: The class `TravelClass` can be instantiated as different travel classes in an aircraft e.g. first class, business class or economy class. This class contains the most important properties for the layout creation. The requirements like the ratios, the desired `pitch` or the

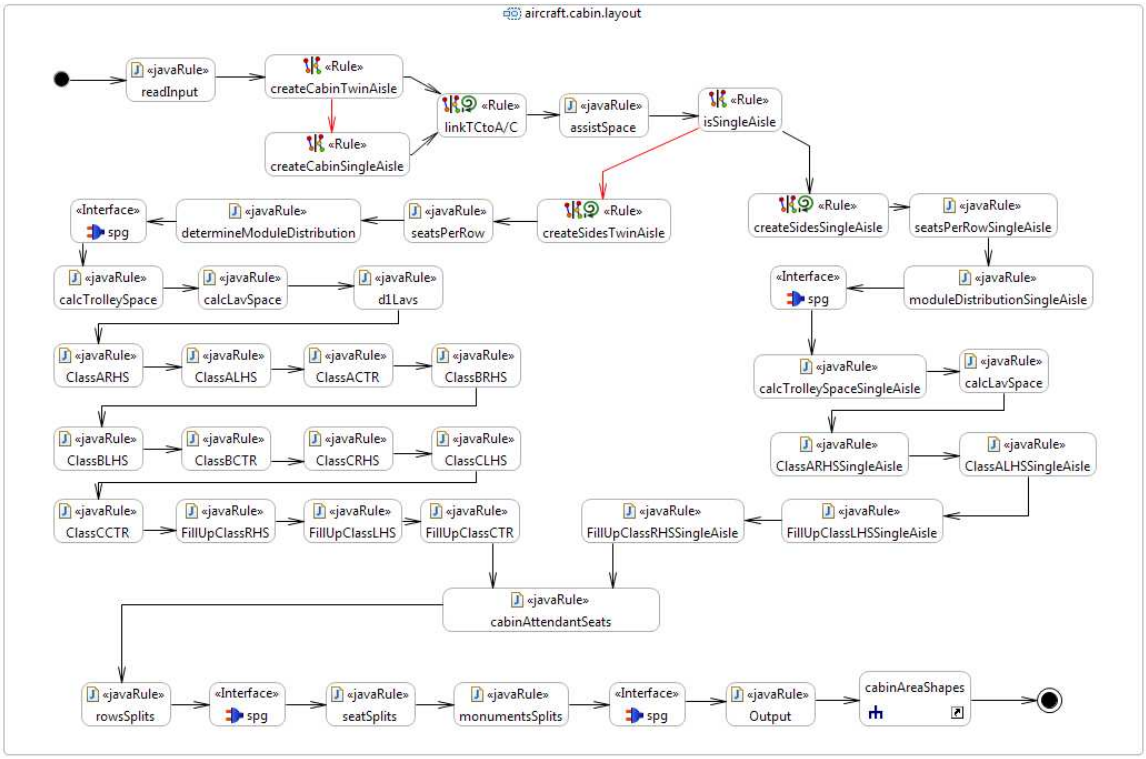

Figure 3: Cabin layout activity diagram

given `trolleyLength` are defined here and determine the layout.

- `Side`: Each `TravelClass` can have different sides like right hand side, left hand side or in case of a twin aisle approach a center side. Each travel class is divided in the needed number of sides.
- `SeatsArea`, `TrolleyArea`, `StowageArea`, `LavatoryArea`: These areas evolve from the division of the sides in x-direction.
- `SeatRowArea`: Each `SeatArea` can be divided into a number of `seatRowAreas`, which represent the space of the seat rows.
- `Seat`: The `SeatRowAreas` are divided into particular seats.

The classes in figure 2 are interconnected and have several properties. This enables (among some other things) the linkage of classes through equations. The associations show the possible data flow. For example, the required seats space of a side of a travel class is represented through the equation:

*totalReqSeatSpaceX = travelClass.hic + travelClass.spaceSRPtoSeatEnd + travelClass.recline + (numberOfRows - 1) * travelClass.pitch.*

The properties are instantiated with the classes. The equations defined in the classes form a system of equations relating the instances in the final model.

2.2. Cabin layout design process

The cabin layout design language is based on the vocabulary modeled in a class diagram as described before. Out of this available vocabulary in the class diagram, the design rules

are created to build up the cabin layout design incrementally by rule-based model-transformations. These rules which are *model-to-model* (M2M) transformations are arranged in the *activity diagram* and the work flow of the cabin layout design language can be processed by the execution of the transformations. Figure 3 shows the activity diagram of the developed design language.

At first, the execution of the activity diagram in figure 3 begins with the requirements and some start values which are defined in an Excel-file. These values serve for the characterization of

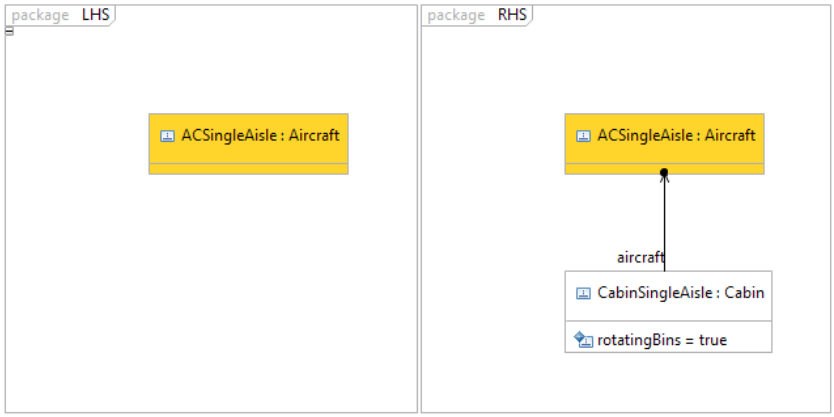

Figure 4: Rule `createCabinSingleAisle`

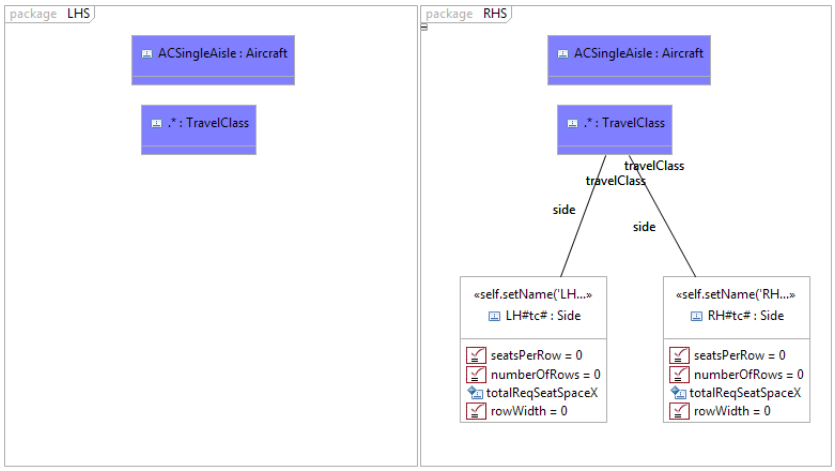

Figure 5: Rule `createSidesSingleAisle`

the different travel classes and their properties. The first rule `readInput` in figure 3 imports the data contained in this Excel-file, so that the data needed can be accessed inside the design language. The input parameters of each travel class are in detail: `numberOfSeats`, `recline`, `hic`, `pitch`, `spaceSRPtoSeatEnd`, `seatsAbreast`, `ratioPaxPerTrolley`, `ratioPaxPerLavatory`, `ratioStowageSpacePerPax`, `lavatoryXdim`, `numTrolleysAlreadyInstalled`, `maxNumTrolleyBoxesCanBeInstalled`, `maxNumLavBoxesCanBeInstalled`, `trolleyLength` and `trolleyWidth`.

After importing the required start values, an abstract instance of the class `Cabin` is created in the rule `createCabinTwinAisle` shown in figure 4 or in an analogous rule `createCabinSingleAisle`. According to the upstream configuration, the branch of a single aisle layout or the branch of a twin aisle layout is chosen. This current choice is modeled with a red arrow. It describes an alternative way, if the rule could not be applied. Elsewise the work flow follows the ordinary black arrow. The illustration in this report describes the work flow of the branch of a single aisle configuration.

The following rule `linkTCtoA/C` in figure 5 creates a connection between the cabin (the instance `Cabin`) and the travel classes (the instances of `TravelClass`). After that, the split of the door area to get the assist space next to the doors is performed in the rule `assistSpace`. The next rule `isSingleAisle` defines the branch, which will be followed, depending on the desired configuration.

The rule `createSidesSingleAisle` in figure5 creates for each travel class a left hand side and a right hand side. In the rule `seatsPerRowSingle-`

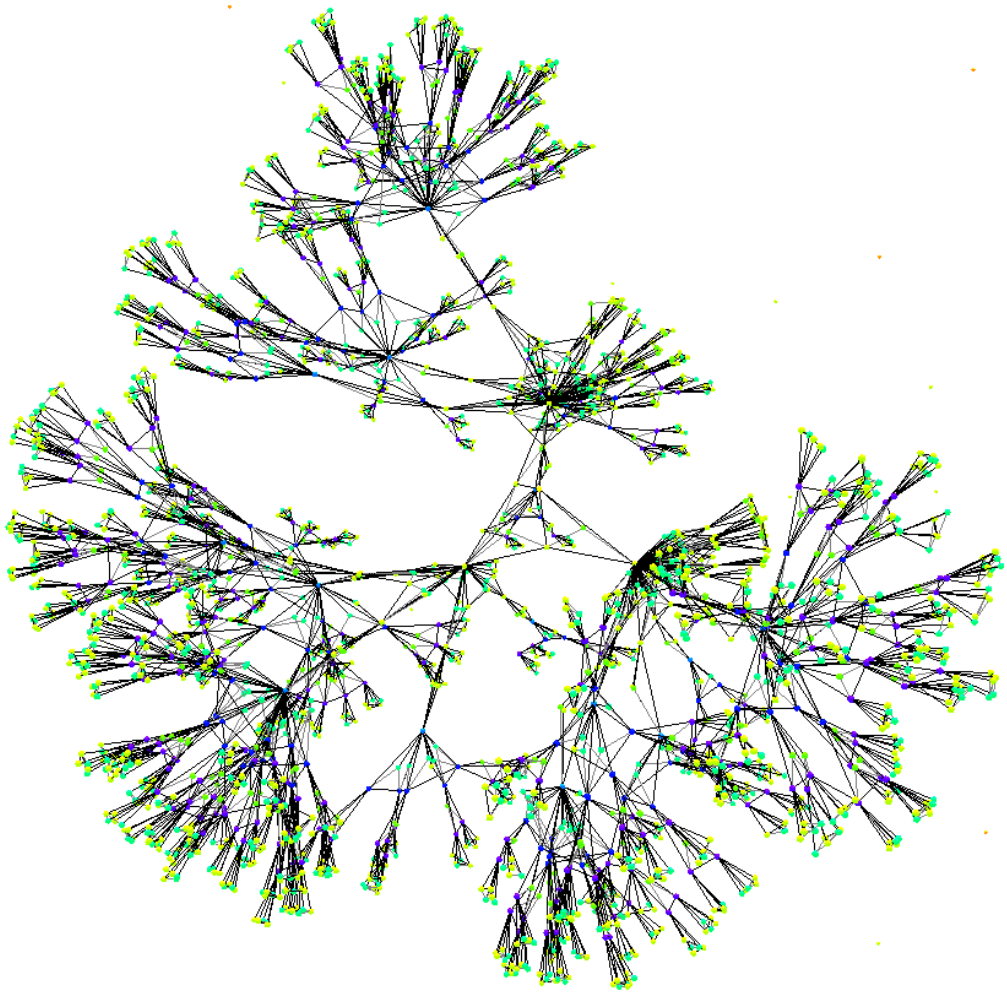

Figure 6: Design graph visualization (topological layout)

`Aisle`, the number of seats per row is indicated according to predefined values. For example, if there are 6 seats abreast in a single aisle configuration the distribution is 3 on the left and 3 on the right. Then the distribution of the modules is set in `moduleDistributionSingleAisle`. This means, if there are for example 18 seats in a travel class with a 3left/3right configuration, there will be 3 rows on the left and 3 rows on the right with 3 seats each. If the number of desired seats does not fit to the seat and module distribution, it is treated accordingly and an output is given if the desired number of seats cannot be installed.

Afterwards the interface `spg` is called to perform mathematical calculation respectively to solve the system of equations, which is set up according to the instances respectively their classes in the class diagram (vocabulary) as described before. The rule `calcTrolleySpaceSingleAisle` determines hereby the needed space for the trolleys, distributes them and gives an output, if there are too many trolleys, which cannot be installed. In a similar way, the needed space for the lavatories is calculated in `calcLavSpace`.

The following four rules (`ClassARHSSingleAisle`, `ClassALHSSingleAisle`, `FillUpClassRHSSingleAisle`, `FillUpClassLHSSingleAisle`) assemble the monuments and the seats into the cabin. Following an installation sequence, the monuments and seats are placed into the cabin. To get the number of seats for the so-called `FillUpClass` an algorithm has been developed. It checks the calcu-

lated seat and monument distribution and adapts the number of elements in the cabin, until the desired ratios for the `FillUpClass` are reached. Afterwards the cabin attendant seats are added in the rule `cabinAttendantSeats`.

With the rule `rowsSplits` the `SeatsArea` in the floor plan are split into particular rows and the boundaries are calculated by calling the mathematical interface `spg`. The rule `seatsSplits` splits the rows into seats with space for legroom and backrest. After another call of the mathematical interface `spg` in the rule `monumentsSplits` the dimensions of the monuments in y-direction are adapted. With the final calculation and the rule Output printing output to the console, the cabin layout activity is finished.

2.3. Visualization

The cabin layout configuration created with the described cabin design language can be viewed in different ways to get out the information needed for further improvement and development of the cabin layout. Figure 6 shows the abstract graph of the created cabin layout design. Every node of the graph represents an instance in UML. Each UML instance in the design graph is hereby shown as a small circle, also the association and inheritance links are shown. The principle of dividing the root area iteratively into smaller areas can be retrieved in the structure of the design graph.

Since the cabin layout has not only been build up logically, but has also been calculated and assembled in physical 3-D dimensions, the graph nodes, respectively the instances of our model, have positioning information. By placing the nodes using this information of their true position later in 3-D, the following arrangement shown in figure 7 can be computationally achieved and displayed.

To examine and improve the cabin layout design language, a special cabin layout view has been developed. Based on the cabin layout data, which is the same than in the two views above, the view in figure 8 shows the individual areas of the cabin layout. The colored elements comply with the areas in the design language. *Yellow* stands for `aisle`, *turquoise* for the seats, *green* for the `lavatories` and *pink* for the `trolleys` and the `galleys` respectively.

2.4. Discussion

The design language for cabin layout of single aisle and twin aisle configuration addresses the following requirements and needs:

- Layout creation depending on ratios as input
- Considering ratios of each travel class
- Ratio PAX per lavatory
- Ratio PAX per trolley
- Ratio PAX per stowage
- Galley distribution
- Lavatory distribution
- Partition definition
- Stowage installation
- Doortype
- Evacuation spaces such as passageway, assist space, cross aisle areas
- Seats parameters: pitch, recline, HIC (Head Injury Criterion), seats abreast

Besides the aforementioned points several other points need also to be considered. There are some rules i.e. aspects in the cabin layout that still have been neglected. Since any expression in a formal language is interpreted and translated by a machine, any formalized rules must represent what is really wanted. In this respect the verbal formulation of a design rule expressed in the words of a natural language must be very clear in order to allow a correct implementation in a design language.

Besides the current cabin layout design sequence implemented, an even more variable way of placing the monuments and seats could be developed to encode more "unconventional" designs.

As two main advantages of graphical modeling, the possibility of graphical rules and coded JAVA rules at the same time allows that the rules written can be as complex as a program written in a JAVA, if need be. This means that the graphical language is extensible be conventional programming techniques. Furthermore, the encapsulation of design knowledge in form of design rules gives an clear overview of the various steps during the design process and allows therefore not to get lost in a big confusing code accumulation.

2.5. Overall aircraft cabin generation process

The successful overall design of an aircraft cabin involves many key disciplines. As the current state-of-art it can be mentioned that each discipline (industrial design, continuum mechanics, fluid mechanics, electronics and so on) understands its own (disciplinary) view on the system. As described in the introduction, the multi-disciplinary integration

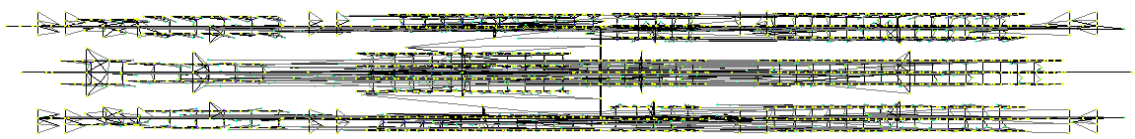

Figure 7: Design graph visualization (geometrical layout)

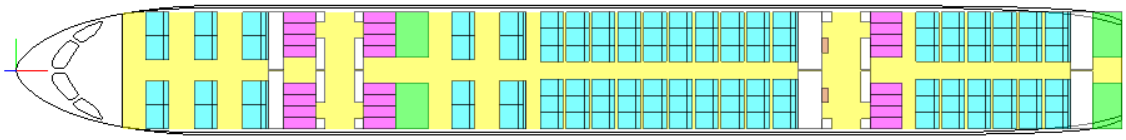

Figure 8: cabin layout of single aisle configuration

of all these different models into one single and consistent "master model" is currently in industry an unsolved open problem.

Graph-based design languages handle the consistency problem of an integrated model quite differently: Instead of trying to solve the integration problem by merging different models into one single master model, all different disciplinary models are automatically generated from a central model [5] and are for this reason consistent by definition.

The price for this achievement is the formulation of aircraft design knowledge in form of a design language in UML format. As far as the engineering is concerned, the design of an aircraft cabin involves (among many others which have to be omitted here for the reason of space) the following important steps [3]:

- definition of the cabin design variants and their boundary conditions
  (plane dimensions, number of frames, definition of crown area and so on)
- definition of cabin architecture and layout
  (number of classes, number of seats, definition of various ratios)
- definition of networks and routes
  (the network is generated through an algorithm, while routes which are intentionally predetermined are read in from a database)
- definition of network components and equipment
  (currently the equipment is read in from a database)
- definition of the network
  (an algorithm is used to connect equipments appropriately. The corresponding connection information is stored in a dedicated UML class diagram)
- routing of electrical cables in 3-D
  (routing includes arbitrary routing space boundaries and collision testing in 3-D)
- integration of routing result into 3-D geometry model
  (through the 3-D model can be navigated in a virtual environment)
- results computation and architecture trades
  (for comparison of design variants, the weight of the cables or the diameter of the routes in the cross sections at the frame positions is calculated and exported to MS-EXCEL or CATIA V5 for further processing)
- design modifications
  (changing the input and re-executing the above processes allows the exploration of the design space through a comparison of two or more variants)

In an industrial research project ongoing for several years, several smaller design languages for different purposes have been combined together into one large design language to generate the aircraft cabin models. This combined design language contains all the necessary design information in an (re-)executable form.

The hierarchical and modular decomposition of the overall aircraft cabin design task allows the encapsulation of individual design tasks into specialized design languages. By combining the cabin layout language with the geometry generation language, a 3-D model of the aircraft cabin can be generated.

Generating a class layout for an aircraft cabin is a task with involves the concurrent satisfaction of many constraints. For the multidisciplinary integration problem of the necessary networks for air, power and data into the crown area of the aircraft the cabin layout represents a significant part of the boundary conditions definition.

Adding the routing language to the network generation algorithm allows the integration of the electrical networks into the crown area of the aircraft. This will be shown in the following sequence of figures 9 to 10 which show the intermediate results of the different steps during the routing.

Together with the previous information from the cabin layout, from which a complete 3-D geometry model can be generated and the placement information of the network equipment including the harness information for the power and information networks can be started. In a first step, the generation of the 3-D geometry model is shown on the left hand side in figure 9. From this 3-D geometry model, the routing space on the right hand side in figure 9 is automatically constructed.

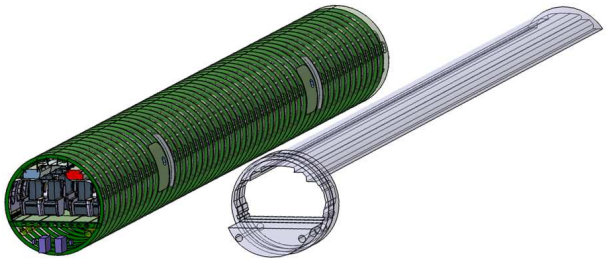

Figure 9: Extraction of routing space (right) from aircraft cabin CAD model (left) [3]

In the second step the network boxes are placed in the routing space, see figure 10 left. A routing algorithm is then used to create the cable connections according to a harness plan loaded from a database. The routing algorithm guarantees a shortest path search and includes collision testing. The routing results are shown in figure 10 right.

Since the 3-D model in figure 10 is computationally accessible any information can be extracted. One finds that the length of network 1 adds up to $558,83$ meters while network 2 has a length of $582,50$ meters. The two equipment weights sum up to $404,22$ kg (for version 1) versus $364,22$ kg (for version 2) and the total weight of equipment and cables sums up to $963,05$ kg versus $946,72$ kg respectively.

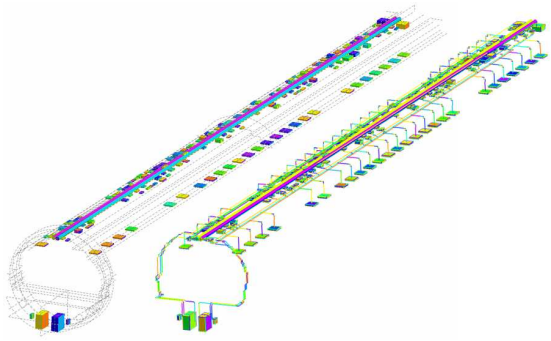

Figure 10: Network components (left) routing algorithm result (right) [3]

From figure 10 the two network quantities total *weight* and total *network cable length* can be easily computed and may be used as performance measures for the comparison of the two generated aircraft designs. The two functionally equivalent networks do however differ in the different internal distribution infrastructure and are therefore considered as design trades. While network version 1 has 8 internal nodes, network version 2 has only 4 internal nodes. As a result, the internal weight distribution can be computed [3].

2.6. Design trades

Two alternative networks with a different infrastructure for power and information distribution were automatically constructed through compilation of the design language in a design compiler. As a result, the two network quantities total weight and total network cable length were selected as performance measures for the comparison of the two generated aircraft designs. By extending this approach in the future, the *Pareto surfaces* of competing aircraft cabin architectures can be automatically constructed, thus serving the cabin architect as a solid guideline for a justified decision making in complex future multi-disciplinary aircraft cabin design scenarios.

3. Summary and outlook

A consistent central model which covers several distinct engineering disciplines such as a cabin layout model (for/from operations), a CAD geometry model (for/from construction in 3-D) and an electrical systems model (from/for network analysis) was built using graph-based design languages.

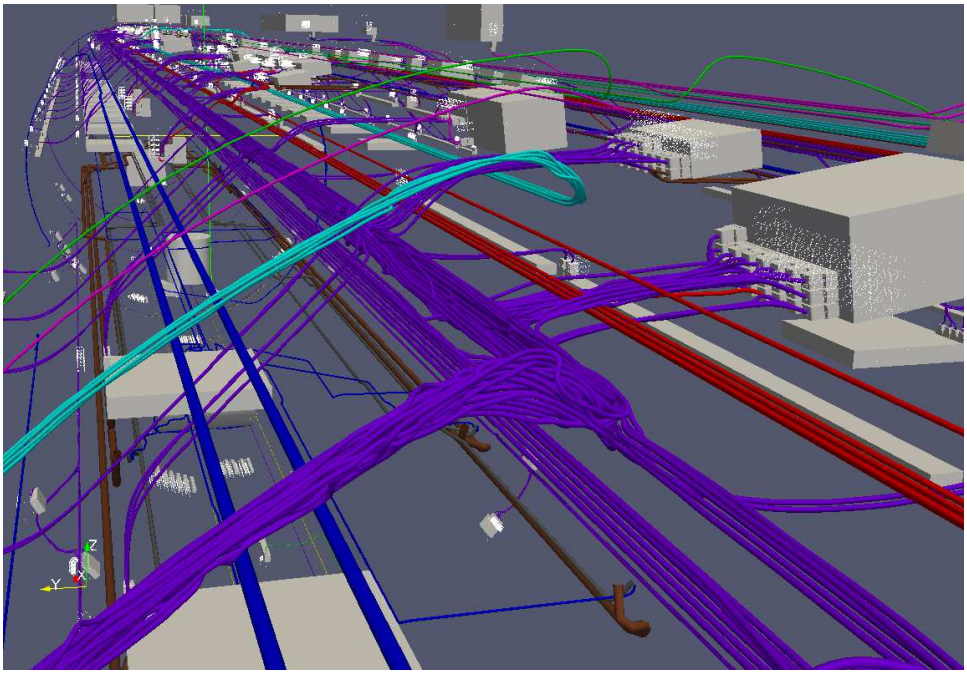

Figure 11: Detailed view of the routed network connections (including collision testing)

The analysis included the generation of the class layout (seats, lavatories and so on) and showed the integration of several routes of a power and an information network. The routing algorithm guarantees a shortest path search and includes collision testing.

As described, many components of electrical systems have been considered and installed in the model using graph-based design languages. In this project phase, the focus was on the integration of several systems and to show the possibility of representing large models inside the graph based design languages, as well as on the interdependencies between different domains of an aircraft design.

On basis of data from a twin aisle configuration, a very good demonstration of the capability of system integration including electrical boxes and connections inside the graph-based design language has been achieved. Figure 11 shows a screenshot of the described system integration with the boxes and the interconnecting cables, routed by a routing algorithm. To give a taste of numbers, 745 equipment boxes have been installed in total. They are connected with 1357 cables. The boxes divide into 205 power and information distribution equipment boxes, 57 antenna boxes and related boxes, as well as 23 boxes, 8 switches and 452 small equipment boxes for integrated modular avionics.

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 achieve automatic placement (e.g. packaging) by suitable algorithms.

References

[1] J. Sztipanovits, X. Koutsoukos, G. Karsai, N. Kottenstette, P. Antsaklis, V. Gupta, B. Goodwine, S. Baras, J. and Wang, Cyber-physical system integration, Proceedings of the IEEE 100 (January) (2012) 29–44.

[2] S. Rudolph, B. Kröplin, Entwurfsgrammatiken - Ein Paradigmenwechsel ?, Der Prüfingenieur 1 (April) (2005) 34–43.

[3] S. Rudolph, S. Hess, J. Beichter, M. Motzer, M. Eheim, Architectural analysis of complex systems with graph-based design languages, 4th International Workshop on Aircraft System Technologies (AST 2013), Hamburg, April 23-24.

[4] S. Rudolph, Übertragung von Ähnlichkeitsbegriffen, Habilitation, Universität Stuttgart, Stuttgart (2002).

[5] IILS Ingenieurgesellschaft für intelligente Lösungen und Systeme mbH, The design compiler 43, `http://www.iils.de/43.htm`.