Available online at www.sciencedirect.com
SCIENCE DIRECT®
Aerospace Science and Technology 9 (2005) 81–91
Aerospace Science and Technology
www.elsevier.com/locate/aescte

Satellite design by design grammars ☆

Satellitenentwurf mit Entwurfsgrammatiken

Jörg Schaefer *, Stephan Rudolph

Institute for Statics and Dynamics of Aerospace Structures, University of Stuttgart, Pfaffenwaldring 27, 70569 Stuttgart, Germany

Received 7 January 2004; received in revised form 11 August 2004; accepted 12 August 2004

Available online 25 September 2004

Abstract

The paper deals with the application of engineering design grammars for a satellite design process. The situation of today's satellite design facilities is addressed and the need for an automated design process support is explained.

The suggested method is to use graph grammars as a new representation of design. A design compiler acts as a 'front-end' for existing computation and visualisation tools. In this context the definition of the vocabulary and the rules of a design grammar and their relation to the syntax and the semantic of a satellite are presented. Particular sections of the design grammar are discussed and the implemented solutions are explained. The transfer of common design procedures to a rule-based design process is explained. The resulting satellite design process is exemplified through the conceptual design of a small satellite.

The applied method is analysed in respect to the required automation of the design process. Its advantages and disadvantages are discussed and compared to the current situation in satellite design.

Zusammenfassung

Die vorliegende Arbeit beschreibt die Verwendung einer Entwurfsgrammatik für den Satellitenentwurf. Einleitend wird die gegenwärtige Situation im Satellitenentwurf angesprochen und die Notwendigkeit eines automatisiert ablaufenden Entwurfsprozesses dargelegt.

Zur Verbesserung des Satellitenentwurfs wird die Verwendung von Entwurfsgrammatiken zur Darstellung des Entwurfsprozesses und der Entwurfskomponenten eines Satelliten vorgeschlagen. Dabei dient ein Entwurfscompiler als Front-End für die Abarbeitung der Satellitenentwurfssprache und zur automatischen Ansteuerung der verwendeten Analyseprogramme und Visualisierungswerkzeuge.

In diesem Zusammenhang wird die Definition von Vokabular und Regelwerk anhand einer Entwurfsgrammatik für Satelliten dargestellt. Dazu werden einige wichtige Kernpunkte der Grammatik und der verwendeten Entwurfsmethodik im Detail erläutert. Der Entwurfsprozess wird anhand eines einfachen Satellitenentwurfs dargestellt und die speziellen Vor- und Nachteile des Satellitenentwurfs mit einer graphenbasierten Entwurfssprache diskutiert.

Keywords: Design grammar; Satellite design; Design compiler; Rule-based design; Design process

Schlüsselwörter: Entwurfsgrammatik; Satellitenentwurf; Entwurfscompiler; Regelbasierter Entwurf; Entwurfsprozess

1. Introduction

One of the main difficulties of the general engineering design process lies in the complex relationship between almost all parts of the system and the increasingly hard constraints on cost and time during the design and production phases.

Therefore, specialised domain-dependent tools were developed in the past to support the specific design processes applied in various fields of engineering (e.g.: satellites, airplanes, automobiles, ...). However, it is desirable to introduce a more generic representation of the applied processes

☆ This article was presented at the German Aerospace Congress 2003.

* Corresponding author.
E-mail address: schaefer@isd.uni-stuttgart.de (J. Schaefer).

1270-9638/$ – see front matter
doi:10.1016/j.ast.2004.08.003

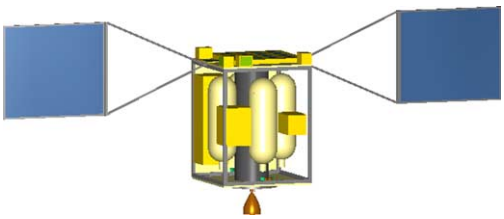

Fig. 1. Exemplary satellite design [27].

which contains all the required elements of the general system engineering design process. A promising methodology is presented in this paper with the specific application to satellite design.

Recent optimisations of the satellite design process led to the installation of centralised concurrent design environments (see [1,7,8,19,30]). Usually, the core component of these facilities is an integrated design model with fast and flexible access to all design parameters. Additional programs for computation, visualisation and simulation are separated from the design model and the required data is imported and exported manually.

The discussion of the generated evaluation results and the decision which modifications should be made for improvements still heavily depends on human interaction. That is to say that the feedback loop between the evaluation of a design and the necessary modifications is not closed in the processing chain of these common design tools. Additionally, not all of the possible options for design modifications can be analysed because the amount of design iterations is limited by design time constraints.

The situation can be improved if the processing chain is closed by a feedback loop of the evaluation results into the synthesis phase of the design process (see [24] and Fig. 7).

A completely closed future processing chain requires the inclusion of means for decision making which translate the evaluation results into specific actions for the synthesis of the design. Since most modification decisions are based on known design rules or experience that can be formulated in an 'if-then' construct, the application of design grammars as a framework for the rule-based design process is suggested.

The storage of the design components and the means for synthesis, analysis and decision making in one combined representation form allows a closed processing chain for the design. This is an unavoidable requirement for an automated exploration of the design space which allows the analysis of multiple modification options ('trade-offs') during the design in an acceptable time frame.

2. Design grammar approach

The design methods that are used in engineering today miss the capability for an unified representation of the design components, the rules for decision making and the methods for analysis (as explained in Section 1). The application of graph grammars as design language representation is a promising solution to this problem as shown in [4,25]. Design grammars for various engineering design domains are shown for space stations [16], aircrafts [18], power pylons [24], multi-body systems [13] and automotive structures [14]. Further utilisation of grammars are described in [6,15].

Design grammars are derived from formal languages as used in the field of computer science [29] (e.g. Ada, Basic, C++) or biology (e.g. L-Systems [22]). Generally, grammars comprise a set of 'vocabulary', 'rules' and 'axioms'. The vocabulary serves as the symbols of the grammar while the rules describe how the vocabulary can be combined in a 'sentence'. The axioms define a starting condition for the construction of a sentence which is build by the sequential execution of one axiom and several rules. A 'language' is defined by the set of all possible sentences of a grammar.

In contrast to the common definition of formal languages which concatenate their vocabulary linearly in the sentences, engineering design grammars are able to model the topological ring structures and complex networks that are found in engineering objects (e.g. structural frames). In addition, they also provide means to map hierarchical information about functional affiliations or dependencies in the design (e.g. functional trees, requirement networks). Therefore, the 'sentences' formed by engineering design grammars are represented by a 'graph' data structure which is referred to later as the 'design graph'.

2.1. Vocabulary

The vocabulary of a design grammar defines the functional, geometrical and physical properties of a design component. Every vocabulary is identified by an unique name and can be instantiated in several 'nodes' in the design graph. An excerpt of a vocabulary definition is shown in Fig. 2.

The vocabulary definition can comprise parameters, variables and operations that describe their mutual dependency (e.g. equations, often also called constraints). Parameters are set in the vocabulary definition or by rules while variables

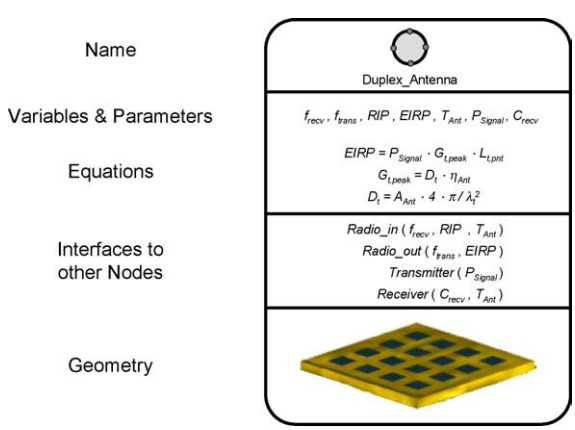

Fig. 2. Vocabulary definition [27].

are determined by the given operations during analysis of the design.

Both parameters and variables can be exchanged between nodes through ‘interfaces’ (sometimes also called ‘ports’). An interface specification in the vocabulary definition contains the variables or parameters that are to be transferred. They are marked with a public qualifier that is used to identify them in this interface. Two nodes in the design graph can be connected if they both provide interfaces with the same specification. During the analysis of a design the required operations are sorted in the necessary order, executed and the result is stored in the associated variables.

In addition, the vocabulary definition can contain a description of the geometrical properties of the component that can be parametrised by its defined parameters or variables. During the visualisation of a design those nodes that contain geometrical information are elaborated according to their parameters and computed variables and aligned against each other (see Section 2.7). The resulting geometrical representation is transferred to the visualisation program for further processing (e.g. visualisation, meshing).

2.2. Rules

The rules of a design grammar define the permitted or required connections between several nodes. The definition of a rule consists of two major parts:

1. The conditional part, which has to be found in the design graph before the rule can be executed.
2. The generative part, which defines the state after execution of the rule.

The conditional part consists of a required topology of nodes that has to be found in the design graph before the rule can be executed. Additionally, requirements for the values of parameters and variables of the nodes contained in the conditional part can be set.
The outcome of a rule execution is specified in its generative part. Just as in the conditional part additional specifications for the contained nodes can be set.

As shown in Fig. 3 a rule is defined by four fields. The quadrants denoted as ‘Q1’ and ‘Q2’ specify the conditional part of the rule while the quadrants denoted as ‘Q3’ and ‘Q4’ define the generative part.

During execution of the rule the following modifications can be performed on the contained nodes:

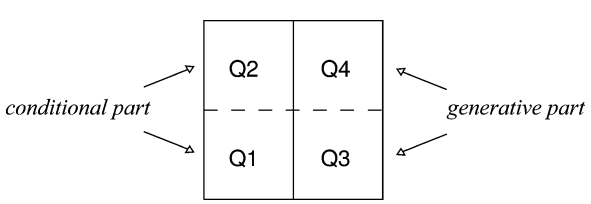

Fig. 3. Rule definition scheme [26].

- Deletion of nodes from the design graph (nodes located in quadrant ‘Q1’).
- Addition of nodes to the design graph (nodes located in quadrant ‘Q3’).
- Creation and removement of connections between nodes (in quadrants ‘Q3’ and ‘Q4’).
- Modification of nodes’ parameters (in quadrants ‘Q3’ and ‘Q4’).

Fig. 4 shows an exemplary rule definition. The rule can be executed when the nodes specified in the conditional part are found in the design graph (nodes ‘1’ to ‘6’). Furthermore, the nodes should be connected so that node ‘2’ is linked to the nodes ‘1’, ‘3’ and ‘4’ and the node ‘5’ is linked to node ‘6’.

During rule execution the following operations are performed.

1. Node ‘2’ and all connections to it are removed.
2. The connection between node ‘5’ and node ‘6’ is removed.
3. A new node ‘7’ is created.
4. Node ‘7’ is connected to nodes ‘4’, ‘5’ and ‘6’.
5. Node ‘1’ and node ‘3’ are connected.

By this definition the conditional refinement and extension of a design can be specified in a simple and generic way. A rule definition can be focused on a specialised design step and includes only those dependencies on other components in the design that are absolutely necessary for its execution. Therefore, it is independent of the overall design context (e.g. a rule adding a battery can be used for satellite design, space-station design and in other domains).

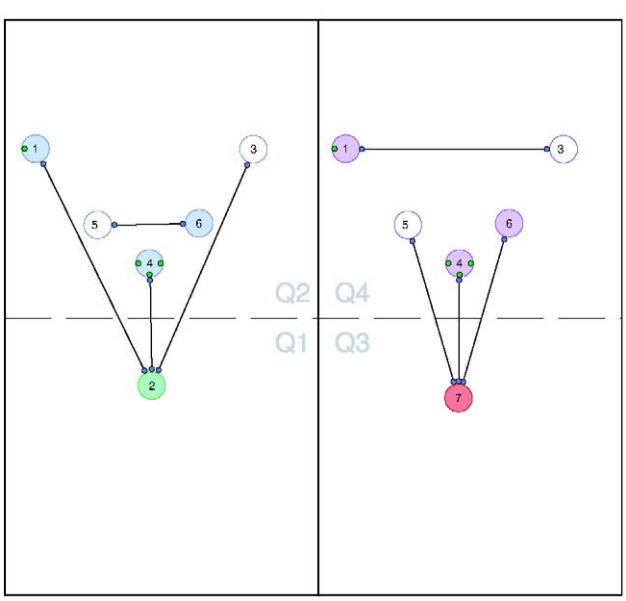

Fig. 4. Rule definition example [26].

2.3. Axiom

At the beginning of a design process an initial state of the design graph has to be set to allow the sub-sequent execution of rules, since those depend on an existing design graph for the evaluation of their conditional parts. Therefore, an initial state of the design graph can be defined in so called 'axioms'. One of these axioms has to be executed at the beginning of the design process.

From an engineer's point of view, an axiom describes the requirements or environment for the design object that are preset and cannot be changed during the design process (e.g. mission objectives, orbit parameters, payload components).

2.4. Programs

For an easy design process it is useful to collect rules and axioms to represent common design patterns. A sequence of several rules is called a 'program' or 'production system' which can be led by an axiom optionally.

In a design grammar programs are used for the set-up of known system concepts during the design process (e.g. an energy supply program collects the rules for addition of batteries, solar panels and control units).

2.5. Design graph

The expansion of components is described in the rules of a design grammar which are executed step by step during the design process. Thereby they extend or detail the current design state by adding or replacing nodes in the design graph. The design graph itself represents only the current state of design without storage of the previous design state history.

2.6. Syntax and semantics

A general syntax definition of a grammar is given by the specification of interfaces between the vocabulary. In general, any two nodes with the same interface specification can be connected. However, in some rare cases it is required to define a direction of an interface (e.g. electrical connectors, lego bricks) and to restrict the possible connections in that way that only interfaces with reverse direction to each other can be connected. Therefore, interfaces can be tagged as 'male', 'female' and 'neutral' implying that 'neutral' interfaces can be connected to interfaces of any type while 'female' interfaces can only be connected to 'male' interfaces and vice versa.

The amount of possible connections is furthermore reduced by the rules which specify the set of allowed connections. That is to say that the rules of a design grammar already contain statements about those combinations of design components that make sense for the desired design object. The knowledge about which combinations are permitted or desired has to be provided in the grammar definition before the design starts and will reflect in most cases well known design concepts. In combination with the vocabulary definition this represents the engineering semantics of a design grammar.

A more detailed discussion of engineering design grammars and the applied definition of syntax and semantics can be found in [2,4].

2.7. Component alignment

The mutual alignment of components can be defined in the design graph by the utilisation of special alignment vectors in the interfaces as shown in Fig. 5 for a box and a panel. Each component defines two vectors and the mounting position in its local reference frame and denotes them in the interface specification with standardised qualifiers as 'normal vector', 'plane vector' and 'position'.

An exemplary assembly is shown as an exploded view in Fig. 5. The alignment vectors are denoted with 'E' for the plane vector and 'N' for the normal vector. The locale reference frames are denoted with 'x', 'y' and 'z' for the corresponding axis.

During the alignment the components are aligned in the global reference frame so that:

- the positions are matched;
- the normal vectors are anti-parallel to each other;
- the plane vectors are parallel to each other.

The exemplary assembly after the alignment is shown in Fig. 6.

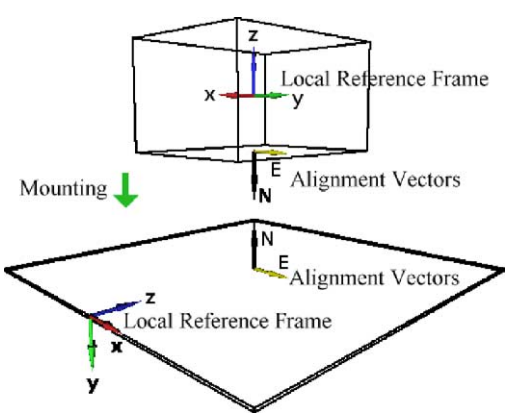

Fig. 5. Component alignment (exploded view).

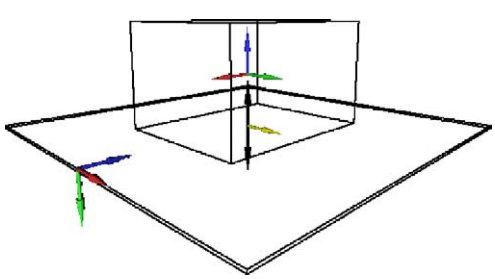

Fig. 6. Component alignment (assembled).

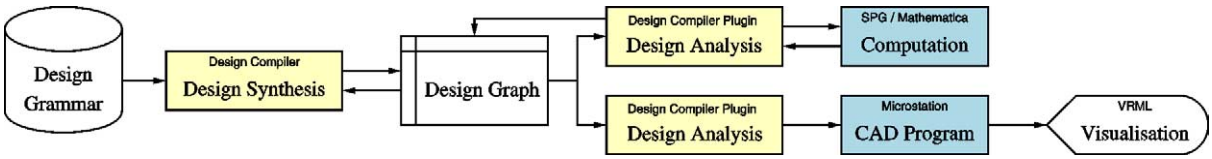

Fig. 7. Design processing chain.

The alignment vectors specified in a mounting interface can be parametrised by using variables of the vocabulary. Thus it is possible to place a mounting position or orientation depending on the overall dimension of a component or other dynamic properties. However, the amount of interfaces of a vocabulary is specified in the vocabulary definition. Therefore, the number of possible mounting interfaces is defined prior to the design execution and cannot be changed during the design process. This is a constraint for the geometrical configuration of components which can be solved by the introduction of functional configuration vocabulary (see Section 3.6).

2.8. Design compiler

The definition and execution of design grammars can be performed in a design compiler which acts as a front-end to the required computation and visualisation programs. The associated processing chain is shown in Fig. 7. During the design process a design grammar is interpreted by the design compiler which constructs the design graph by the execution of the chosen programs of the grammar. The design analysis is started either by user command or automatically (e.g. when variables are queried in rule constraints) and is performed by the activation of application specific plugins. These plugins generate the required output files by exporting their associated data from the design graph and start the external analysis tools (e.g. Mathematica). After completion of the computation they import the result data back into the design graph. The design compiler '43' (see [3,9]) is used for the design grammar presented in this paper. A screenshot of the design compiler is shown in Fig. 8. The figure shows the design compiler environment with a visualisation of the design graph in its center and a geometric visualisation of the design in a separated window. In two sub-windows on the right hand side a list of the available programs (lower window) and the rules contained in the active program (upper window) are shown. The design process is performed by the controls shown on the left hand side of the design graph window. In addition to the execution of programs from the program list it is possible to launch specific analyses by hand (e.g. graphics generation, FE computation). As described in the processing chain (see Fig. 7) these analyses are performed by special plugins which are dynamically loaded by the design compiler (e.g. uStation [21], Ansys [5], VRML [17]).

The external programs used for symbolic and numerical computation of the variables include 'Mathematica' [20]

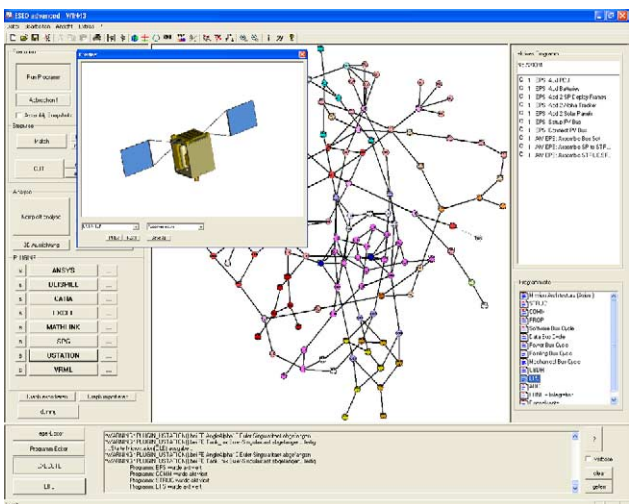

Fig. 8. Design compiler 43 [9,27].

and the 'Solution Path Generator' [23,24,32]. The graphical output is generated in a CAD environment and converted afterwards to 'VRML' for the sake of easy presentation. Additional programs for calculations in the field of computational fluid dynamics or structural dynamics are available and can be attached to the design compiler through a flexible plugin methodology.

An automated design process can be performed by the execution of a design program until no more modifications are applied to the graph. All necessary calculations during the design cycles are triggered automatically by the design compiler. This allows the design of complex objects with huge computational needs in an automated process without permanent user interaction for starting the computations (resp. for data import and export). The exemplary satellite design presented in this paper contains several thousands of equations which are solved symbolically by Mathematica.

In addition to the manual start of a design process '43' offers the option for a design variation by an evolutionary algorithm which allows an almost automated exploration of the design space [4]. During an evolutionary design process the design is modified within a preset set of parameters and rules. The fitness of one design process is determined by a fitness value which has to be defined by the vocabulary. Therefore, the desired fitness function can be defined dynamically to fit the needs of the specific design domain.

3. Satellite design grammar

An exemplary satellite design grammar was developed to examine the application of design grammars for satellite design (see [28]). The design drivers were the fundamental decisions to follow a 'top-down' strategy and the concentration on the design on system level. During the development several parts of the design grammar were identified to be of major importance for the design process:

- The utilisation of design requirements.
- The establishment of system budgets.
- The geometrical design configuration.
- The representation on system or sub-system level.

The capabilities of the developed design grammar are demonstrated by the semi-automated design of a micro-satellite (shown in Fig. 1).

3.1. Design process

The top-down strategy implies that the sub-system design process is based on the reaction on previously established requirements. This strategy was chosen to follow the design philosophy that is used in satellite design today: 'design as response on requirements'. It reflects the activities during the early design phases as described in [10–12,31]. In this context the common term of 'requirements' is refined to 'internal' requirements and 'external' requirements.

The external requirements are assumed to be set before the design process starts, for instance as a result from previous mission analysis, mission objectives or general preconditions that cannot be changed during the design process. These external requirements are defined in an axiom of the design grammar and set before the execution of any other design rule.

The internal requirements are those that emerge from the sub-system needs during the advancing system design (e.g. electrical power, data storage, ...). They are determined during the design process and stored in the system budgets. Therefore, the system budgets are of vital importance for the design on system level as they allow the feedback of needs between the several sub-systems. In general only one sub-system depends on the outcome of one specific system budget which is aggregated from the needs of all other sub-systems.

The typical design process performed in the satellite design grammar can be separated into the following stages:

1. Set-up of the external requirements.
2. Response on requirements by the sub-systems.
3. Determination of the internal requirements.
4. Design representation on system level.

The first step is performed by the axiom of the grammar that creates the necessary requirements nodes in the design graph. Steps two and three represent the actual design iterations and are repeated until all the requirements are fulfilled. The design process ends when no more changes are made to the design graph and the design proceeds to step four which summarises the developed design to a representation on system level. The corresponding loop is shown in Fig. 9.

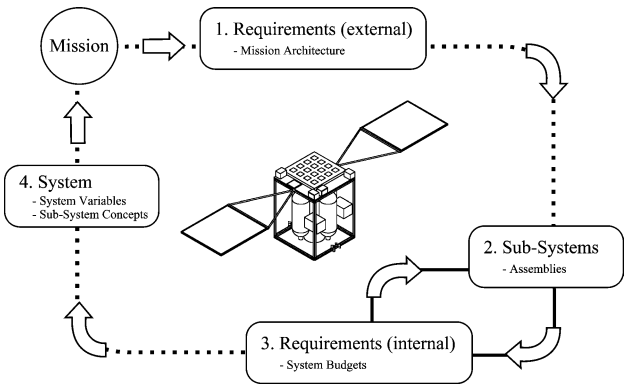

Fig. 9. Satellite design cycle [27].

3.2. Sub-system concepts and components

Inside the satellite design grammar the available satellite components are modeled in the vocabulary definition. Every component is defined by one vocabulary that contains the necessary parameters, their functional dependencies and the interfaces to other components. Fig. 2 shows an excerpt of the vocabulary definition of a duplex antenna while Figs. 10–13 show the geometries of some components which are used in satellites.

The different components are connected by standard interfaces which allow the easy replacement of one component by another component of the same type (e.g. exchange of patch antenna with cassegrain antenna). The interface specifications themselves are a major design subject as they determine the generality of the design grammar.

Sub-systems are created and modified by their specific rule-set which is assembled as a 'program' in the design

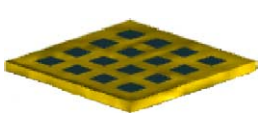

Fig. 10. Patch antenna.

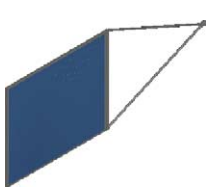

Fig. 11. Solar generator wing.

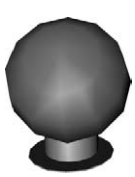

Fig. 12. Spherical tank.

Fig. 13. Cassegrain antenna.

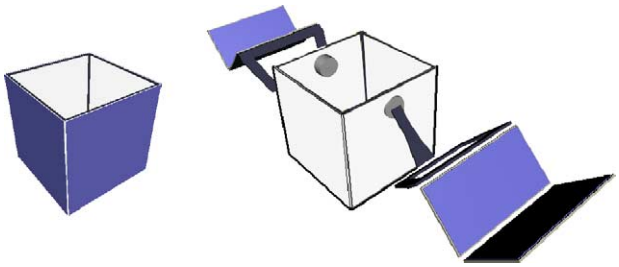

Fig. 14. Different energy supply concepts.

grammar. Each program is designed for an unique sub-system concept and its activation depends on an initial condition that is given by the existence of the specific requirement for a sub-system of that type (e.g. electrical power need triggers creation of an electrical power supply sub-system). After creation of a sub-system its modification is triggered by changes in the parameters of the associated requirement. The sub-system programs can add and remove components or modify their parameters to fit the requirements. As an outcome of this the created components can set additional requirements to other subsystems by specifying system budget items in their interfaces (e.g. electrical power requirement).

As an example, Fig. 14 shows two different concepts for the placement of solar cells (e.g. body mounted or deployed) on the satellite structure. The required components are added by a program that elaborates the chosen concept when it is activated. In this way sub-system concepts can be traded against each other without impact on the implementation of other sub-systems. To analyse the effect of a conceptual change it is only required to exchange the corresponding program in the program sequence of the design compiler.

3.3. Level of design detail

During the development of the satellite design grammar it turned out to be more efficient to model the components at most on the level of sub-system components instead on assembly level or part level. That is because a more detailed modeling of the components results in an increased complexity of the sub-system topologies which are hard to handle as a whole by simple and generic rules.

Smaller portions of the sub-systems can still be found and modified by simple rules. However, the complete network of all nodes forming the higher-level sub-system usually is very complex and not fixed with respect to the various configurations of its components. Therefore, it would be necessary to define the desired rules for all possible combinations of components in a sub-system if it is to be manipulated as a whole. By this, the required amount of rules in a grammar is significantly increased. In addition, the rule definitions would be very specialised and complex and therefore result in a very specific and fixed design grammar. This is in contradiction to the suggested approach to formulate the rules of a design grammar as general and simple as possible to allow a vast variety of different designs with the same rule-set.

Therefore, the decision at which stage in the design process the design is refined into a deeper system level has to be chosen carefully in accordance with the desired design phase. In the present satellite design grammar, which is intended for a preliminary system design, the defined components are among others: generic boxes, antennas, solar panels, thrusters, tanks, panels and beams. It is however straightforward to increase the achievable design detail by additional refinement rules.

3.4. System budgets

During the design process various values have to be summarised in the so-called ‘system budgets’ as a system level representation of the components‘ values (e.g. energy consumption, mass, moments of inertia). Several of those system budgets have to be established and maintained during the design process for the evaluation of the satellite design (e.g. electrical power, data handling, mechanical properties).

Therefore, a simple and generic procedure for the automated collection of the budget items and their aggregation to the required budget was developed.

The fundamental problem while establishing the budgets is that the number of budget items is not fixed during the design process. Their amount can increase and decrease during the design iterations when new components are added to or removed from the design. Therefore, it is impossible to set up a static topology inside the design graph to aggregate the budget items. The implemented solution to this problem is the set-up and maintenance of linked lists in the design graph. Each list represents one type of budget and consists of nodes that are instantiated from a small set of special vocabulary (see Fig. 15).

The budget list is terminated by two special nodes. The first node is the ‘terminator’ node (denoted with ‘0’ in Fig. 15) which has to provide safe default values for the parameters contained in the budget to avoid an empty budget and to ensure that the budget aggregation method can operate. The last node is the ‘budget’ node that contains the overall budget obtained from the budget items (denoted with ‘=’ in Fig. 15).

New budget items are connected to the budget by inserting a new node in the list that is connected to its predecessor

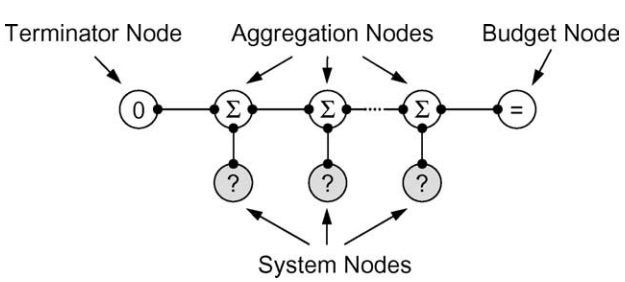

Fig. 15. Budget list [27].

and successor in the list as well as to the node representing the budget item. These interconnecting nodes are called 'aggregation' nodes and denoted with 'Σ' in Fig. 15. In addition to the establishment of the list structure these nodes contain the budget specific method of parameter aggregation that is needed to obtain the overall budget. Required transformations between the system node parameter set and the budget parameter set (e.g. reference frame transformations) are performed in the aggregation node before inclusion in the budget.

The budget item nodes are connected to the aggregation nodes by a bus specific standard interface which is used to identify those nodes that are to be added to the budget. Every node in the design graph that offers this specific interface, except the aggregation nodes themselves, are inserted in the budget list. Thereby it is possible to specify a desired connection of a node to a specific bus only by defining the corresponding interface in the node's vocabulary. This allows a better modularisation of the vocabulary as the design components don't need knowledge about the budget aggregation method and can specify the required parameters in their local reference system.

The establishment and maintenance of the budget list is controlled by a set of rules that have to be executed sequentially after every change of the system design. This rule sequence removes eventually unconnected aggregation nodes, inserts new budget items in the list and generally removes or creates the budget list's terminating node.

The vocabulary and rules for a budget can be formulated as a fixed template. For every system budget this template has to be implemented by adding the budget specific set of parameters and the aggregation method in the vocabulary and by using this budget specific vocabulary in the implementation of the rule-set.

Therefore, the vocabulary and rules of the budgets are encapsulated in a sub-grammar inside the design grammar. As a consequence thereof, the implementations of the system budgets and the system components are functionally separated from each other. They are only connected through the common interface specification between the budget item nodes and the aggregation nodes. For most parts of the design process this budget sub-grammar is not even linked to a specific project or engineering domain and can be re-used in other design grammar projects.

3.5. Satellite configuration

The creation process for a geometrical representation of a design can be split up into two stages:

1. The creation of the structural parts, the so-called '*enumeration problem*'.
2. The part assembly into a satellite configuration, the so-called '*configuration problem*'.

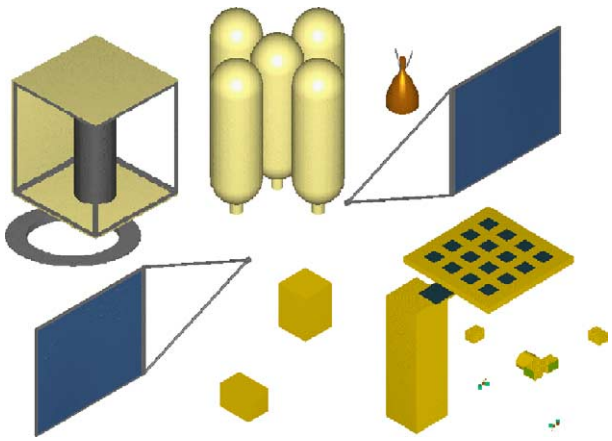

Fig. 16. Enumeration of components [27].

During the enumeration process, the structural parts are created by the sub-systems as components of a specific subsystem concept. New parts can be added and existing parts can be modified and deleted during the design process. This process is called the 'enumeration' since all the required structural parts are added to the design graph, independent of their relative position in the satellite configuration. The list of structural parts used in the exemplary satellite design is shown in Fig. 16.

The configuration process begins after the enumeration problem is solved as the parts have to be assembled in a satellite configuration. This procedure is part of the 'configuration problem' and is often also called 'packaging'. The three dimensional configuration of a satellite involves complex analysis of body intersections and comprises a large quantity of possible combinations. The complexity of the configuration problem scales up with the 'density' of a satellite, i.e. with the ratio of the satellite size to the number and the size of the parts. Therefore, the configuration problem is omitted in the present satellite design grammar and remains subject of future extension of either the specific design grammar or the generic design compiler.

3.6. Configuration layers

To allow an assembly of the satellite without the automated configuration of the parts inside the design grammar so-called 'configuration layers' were introduced. They are instantiated from an unique vocabulary into the design graph and serve as geometrical interfaces between structural parts. In the special case of the satellite design grammar they link the components of the secondary structure (parts created from sub-systems) to the parts of the primary structure (parts created by the main structure of the satellite). By this means it is possible to mount several secondary structural parts on one primary structural part in a predetermined way as shown in Fig. 17.

Configuration layers have no geometrical representation and the number and position of the various mounting ports is given in their vocabulary definition. However, it is con-

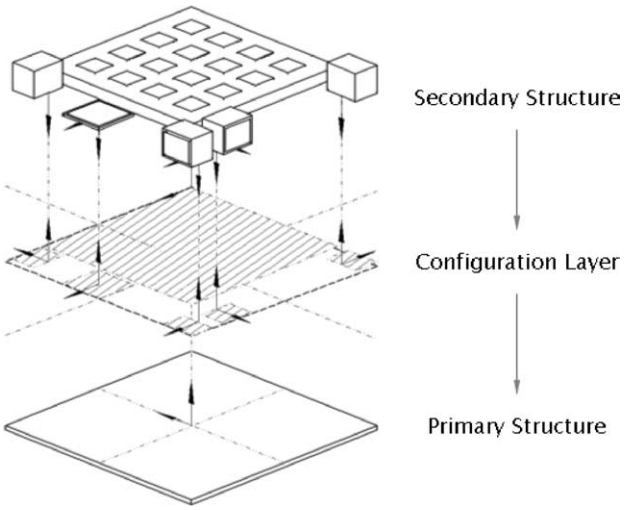

Fig. 17. Configuration layer [27].

ceivable that in future extensions their definition is imported from a packaging algorithm external to the design grammar.

3.7. Exemplary satellite design

An exemplary design and integration sequence is shown in Figs. 18–20 and in Figs. 21–22. The design begins by setting of the external requirements that demand:

- a cubical shape made of beams and panels as primary structure;
- two antennas for the communication sub-system;
- a propulsion sub-system for an orbit manoeuvre;
- a micro-satellite launch on the ASAP5 platform.

The conceptual settings were to use a central thruster with cylindrical tanks in the propulsion sub-system, deployable solar arrays for the electrical power supply and a combination of a single-patch antenna with a multiple-patch antenna in the communication sub-system.

The design steps are executed so that new structural parts are mounted on the satellite after every design step. Furthermore, the satellite configuration was derived from a configuration proposal for the 'European Student Earth Orbiter' (ESEO). A list of the used components is shown in Fig. 16 and the top-panel configuration layer is shown in Fig. 17.

In the first design step the requirement for a cubical shape was detected and elaborated by the set-up of the primary structure (see Fig. 18). The launch adapter is already mounted at the bottom panel of the primary structure (the side panels are not shown for a better visibility of the interior space). The next figure (Fig. 19) shows the state after elaboration and integration of the propulsion sub-system. Five cylindrical tanks are mounted inside the satellite and one central thruster is mounted below the bottom panel. The external requirements for the communication sub-system are fulfilled in the third design step. Fig. 20 shows the satellite after integration of two antennas on the top-panel and two transceiver boxes inside the satellite.

At this state of the design all external requirements have been fulfilled. However, the installed sub-systems need electrical power as well as control lines and control software. These needs have been aggregated in the electrical power budget, the data budget and the software budget which serve as internal requirements for the on-board data handling sub-system and the electrical power supply sub-system that are added during the next design steps.

Fig. 21 shows the satellite after integration of a computer box as a response on the requirements for the storage and processing of data by software. In the last design step the requirement for electrical power is fulfilled by the addition of two solar panel wings and a box that contains the necessary batteries and power control unit (see Fig. 22). The satellite is closed with a panel on the front side to show the planned lookout of the primary structure.

In this state the design process is stopped as the basic principles have been shown. The next steps, which are left out here, would show the creation of the attitude determination and control sub-system as a response on the pointing requirements of the antennas, which would add the thrusters, a reaction wheel and the sensors shown in Figs. 16 and 17. After that the existing sub-systems probably have to be scaled up or down until the changed mass properties, electri-

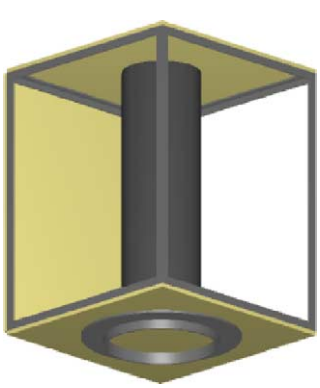

Fig. 18. Design step 1 [27].

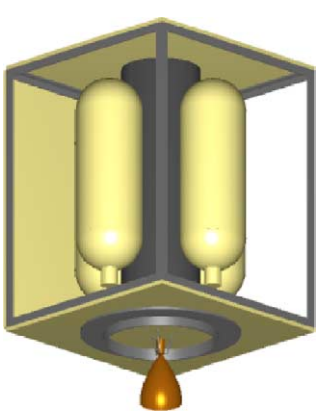

Fig. 19. Design step 2 [27].

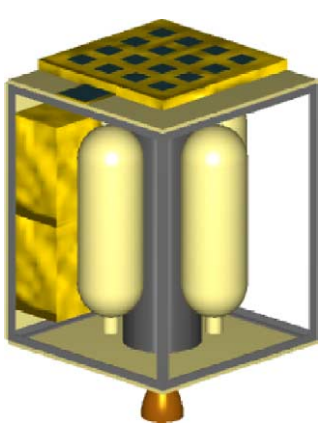

Fig. 20. Design step 3 [27].

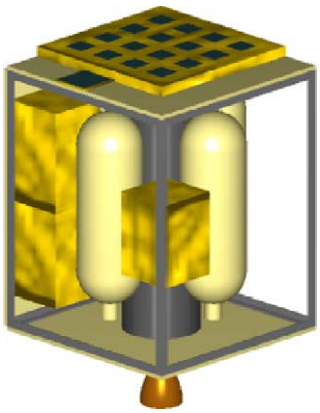

Fig. 21. Design step 4 [27].

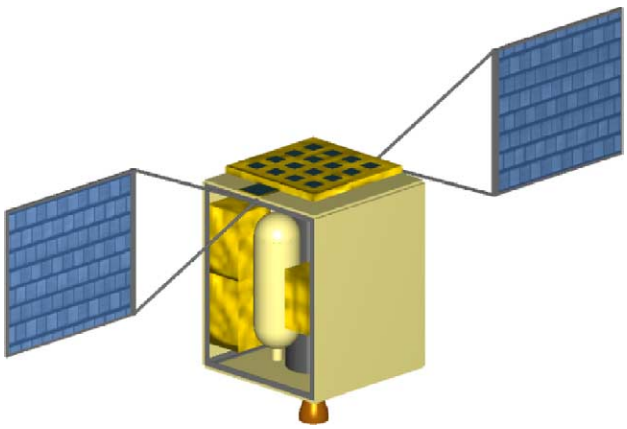

Fig. 22. Design step 5 [27].

cal power requirements or control requirements are fulfilled. The design process stops when no more changes are made to the design.

4. Conclusions

A satellite design grammar was developed and a simple satellite is designed in an automated process (except for the packaging). The automated exploration of a given design space is possible by the implementation of various competitive sub-system concepts and the application of evolutionary algorithms [4].

The design space is limited by the provided components and sub-system concepts as the design grammar can only use the knowledge that is previously included in its vocabulary and rule definitions. On the other hand, this is an advantage when specific design concepts should be used and evaluated with respect to their implementation by given 'off the shelf' components. In this case only the component vocabulary has to be replaced by more specific vocabulary without effect on the general design procedure.

The introduction of 'configuration layers' allows the flexible and modular geometrical configuration of satellites. Therefore, it is possible to execute the design grammar in completely automated mode. Configuration layers can either be defined statically by hand in the design grammar vocabulary or could be generated dynamically by an external configuration program. Therefore, the common functional design space is enlarged by a variety of geometrical configurations which allows a more general exploration of the design space.

The application of the design grammar allows a flexible variation of multiple satellite designs in the conceptual design phase of a satellite project. Because of the open front-end philosophy of the design compiler a found design can be transferred to external design environments or simulation architectures. The rule based design approach allows the documentation of the decision tree during the design process and the repetition of the design process at any time.

The level of detail that is provided by the present satellite design grammar is sufficient for a feasibility study and offers the option to continue the design process with a deeper level of detail in later design phases.

Due to the generality of the approach it is possible to easily exchange vocabulary and rules between different design projects. Therefore, existing design knowledge from parts of one engineering design can be transferred and be re-used in the context of another engineering domain (e.g. environmental control systems of trains, cars and space stations). This mechanism does not only allow the spin-off of available technology but also its merger and thus eventually extends the available design space to previously unreachable areas. Because of the possible separation of function and geometry this powerful option can be used on both the conceptual and the product level.

In this respect the design grammars have also shown their capabilities to store, retrieve, reuse and adapt the coded design knowledge in an efficient way and qualifies therefore as the basic core technology of a successful future engineering design knowledge management methodology.

References

[1] J.A. Aguilar, The aerospace corporation's concept design center, 8th INCOSE Symposium, Vancouver, 1998.
[2] R. Alber, Formale Sprachen zur rechnergestützten Entwurfssynthese, DGLR Jahrbuch 2002, Deutsche Gesellschaft für Luft- und Raumfahrt, Bonn, 2002.
[3] R. Alber, S. Rudolph, "43" – a generic approach for engineering design grammars, in: Proceedings AAAI Spring Symposium 'Computational Synthesis', Stanford, 2003.
[4] R. Alber, S. Rudolph, B. Kröplin, On formal languages in design generation and evolution, WCCM V – Fifth World Congress on Computational Mechanics, Vienna, 2002.
[5] Ansys, Ansys Inc., http://www.ansys.com.
[6] E.K. Antonsson, J. Cagan, Formal Engineering Design Synthesis, Cambridge University Press, Cambridge, 2001.
[7] M. Bandecchi, et al., Concurrent Engineering Applied to Space Mission Assessment and Design, ESA Bulletin 99, ESA Publications Division, Noordwijk, 1999.
[8] M. Bandecchi, et al., The ESA/ESTEC concurrent design facility, in: Proceedings of EuSEC 2000, Munich, 2000.

[9] Designcompiler 43, IILS mbH, http://www.iils.de.
[10] ECSS-M-30A, Space project management – project phasing and planning, European Cooperation for Space Standardisation, 1996.
[11] ECSS-E-10A, Space engineering – system engineering, European Cooperation for Space Standardisation, 1996.
[12] P. Fortescue, et al., Spacecraft Systems Engineering, third ed., Wiley, Chichester, 2003.
[13] S. Gehrig, Anbindung einer Mehrkörpersimulation an den Entwurfscompiler 43, Studienarbeit, Institute for Statics and Dynamics of Aerospace Structures, University of Stuttgart, Stuttgart, 2004.
[14] M. Haq, S. Rudolph, EWS-Car – Eine Entwurfssprache für den Fahrzeugentwurf, VDI Tagung Berechnung und Simulation im Fahrzeugbau, Würzburg, 2004.
[15] J. Heisserman, et al., A design representation to support automated design generation, in: J.S. Gero (Ed.), Artificial Intelligence in Design '00, Kluwer Academic, Dordrecht, 2000, pp. 545–566.
[16] M.R. Irani, S. Rudolph, Design grammars for conceptual designs of space stations, IAC-03-T.P.02, 54th International Astronautical Congress, Bremen, 2003.
[17] ISO/IEC 14772: the virtual reality modeling language, International Organization for Standardization, 1997/2002.
[18] T. Kormeier, et al., Topological and parametrical design of aircraft surfaces using design grammars, DGLR Kongress 2003, Deutsche Gesellschaft für Luft- und Raumfahrt, Bonn, 2003.
[19] R. Mager, R. Hartmann, The satellite design office at astrium – a success story of an industrial design center application, in: Proceedings of EuSEC 2000, Munich, 2000.
[20] Mathematica, Wolfram Research Inc., http://www.wolfram.com.
[21] MicroStation, Bentley Systems Inc., http://www.bentley.com.
[22] P. Prusinkiewicz, A. Lindenmayer, The Algorithmic Beauty of Plants, Springer, New York, 1990.
[23] S. Rudolph, M. Bölling, Constraint-based conceptual design and automated sensitivity analysis for airship concept studies, Aerospace Science and Technology 8 (2004).
[24] S. Rudolph, Übertragung von Ähnlichkeitsbegriffen, Habilitation Thesis, Faculty of Aerospace Engineering and Geodesy, University of Stuttgart, Stuttgart, 2002.
[25] S. Rudolph, Aufbau und Einsatz von Entwurfssprachen für den wissensbasierten Ingenieurentwurf, 3. Forum KBE, Messe CAT. PRO, Stuttgart, 2003.
[26] J. Schaefer, Eine Entwurfsgrammatik zur Flugbahnentwicklung und Bewertung für Raumfahrzeuge, Diplomarbeit, Institut für Raumfahrtsysteme/Institut für Statik und Dynamik der Luft- und Raumfahrtkonstruktionen, Universität Stuttgart, Stuttgart, 2003.
[27] J. Schaefer, et al., Satellite Design by Design Grammars, DGLR Kongress 2003, Deutsche Gesellschaft für Luft- und Raumfahrt, Bonn, 2003 (short conference version of present journal article).
[28] J. Schaefer, Satellite Design Grammar, Studienarbeit, Institute for Statics and Dynamics of Aerospace Structures, University of Stuttgart, Stuttgart, 2002.
[29] U. Schöning, Theoretische Informatik kurz gefasst, BI Wissenschaftsverlag, Mannheim, 1993.
[30] R. Shishko, The proliferation of PDC-type environments in industry and universities, in: Proceedings of EuSEC 2000, Munich, 2000.
[31] J.R. Wertz, J.L. Wiley, Space Mission Analysis and Design, third ed., Kluwer Academic, Dordrecht, 1999.
[32] H. Yusan, S. Rudolph, A study in constraint management integration into the conceptual design phase, in: Proceedings of DETC'99, ASME Design Engineering Technical Conferences, Las Vegas, 1999.