Aerospace Science and Technology 49 (2016) 63–72

Contents lists available at ScienceDirect

Aerospace Science and Technology

www.elsevier.com/locate/aescte

Modeling graph-based satellite design languages

Johannes Gross *,[1], Stephan Rudolph

Institute for Statics and Dynamics of Aerospace Structures, University of Stuttgart, Pfaffenwaldring 27, 70596 Stuttgart, Germany

ARTICLE INFO

Article history:
Received 5 June 2014
Received in revised form 8 October 2015
Accepted 19 November 2015
Available online 27 November 2015

Keywords:
Design languages
Rule based design
Model-based engineering
Knowledge representation

ABSTRACT

Increasing complexity in spacecraft design requires new ways for comprehensive problem formulation. Graph-based design languages are an innovative response to this challenge. Using the Unified Modeling Language (UML), design languages are a formal, executable description of the design knowledge. The FireSat mission given in the textbook [1] from Wertz is used to demonstrate the analysis of more designs in shorter time.

For automating the design process, the domain knowledge is mapped onto a hierarchy of different modular design languages. Thereby the couplings between the components of the system are resolved and defined as generic interfaces. The variables and equations for the different subsystems are grouped in classes that describe the decomposition of the system. Rules are defined on the instance-level to recombine the different class-instances into a valid design description of the FireSat satellite. A sequence of these predefined rules specifies the integration of the components to a satellite system.

The paper is part of a series of three papers. The second paper [2] describes the analysis of equation systems for the whole satellite system. The third paper [3] shows the integration of detailed simulation models using the description mechanisms of the design language.

1. Introduction

This paper is the first in a series of three papers describing the automation potential in the satellite design process by the use of so-called graph-based design languages. Throughout the paper the fictious FireSat mission described by Wertz in [1] and used by Larson in [4] is used as reference example. Further FireSat examples to demonstrate language standards were given by Delp [5,6] and to demonstrate methodology by the authors of this publication in [7–11].

The current satellite design process as lived in various satellite design centers throughout the world (e.g. in ESA's CDF [12]) is based on concurrent engineering principles [13], but is still far from a complete process automation as implemented for the FireSat design process by the authors using a graph-based design language approach. The purpose of this paper focuses therefore uniquely on how such an automation of the satellite design process can be achieved, what kind of information representation is needed and how the multi-disciplinary design information has to be structured and processed to reach the desired automation.

2. Design languages

There exist many formal techniques for engineering design synthesis as described in the book by Antonsson and Cagan [14]. Qualitatively, the methods for design synthesis and automation may be classified according to their formalisms exhibiting either string-based, shape-based or graph-based representations. As one of the earliest precursors to the graph-based design language methodology as it is used later in this work so-called "L-Systems" have been conceived by Lindenmayer [15] and his co-workers [16]. L-Systems dispose of a rule-based synthesis mechanism working on a string-based representation. The rules are expressed in an "IF-THEN"-format and work similarly to the "search-and-replace" mode in a text editor. L-Systems are therefore known in the field of computer science as rewriting systems [17]. The resulting string representation after rule execution is translated into geometry using a so-called "turtle graphic" translation table [16]. L-Systems have been shown to be a mighty and performant modeling and automated synthesis tool for all kinds of biological structures such as plants, trees, sea shells, etc.; see [18] for details. One of the main deficiencies of L-Systems experienced by the authors is however their inbuilt tree-like product architecture which, while fitting perfectly well to tree-like biological "product" architectures such as plants and trees, limits its applicability in engineering to cycle-free product architectures only.

* Corresponding author.
E-mail address: johannes@jpl.nasa.gov (J. Gross).

[1] This work was conducted for the PhD Thesis of the corresponding author at the University of Stuttgart.

http://dx.doi.org/10.1016/j.ast.2015.11.026
1270-9638/

Shape grammars in contrary use a direct representation of geometry and a rule mechanism to directly manipulate shapes and create new (or combined) shapes from rules. Applications of shape grammars can be found most notably in architecture [19] and engineering [20] as described in [14]. The shape grammars in 2D from Stiny [19] were later expanded to applications in 3D as in [21,22] and have been focussing on rule-based geometry creation in a comparable manner as in other design synthesis methods.

The direct geometry representation of shape grammars however, makes it difficult to represent and manipulate the steadily growing portion of non-geometric design information. Using a rule-based modification mechanism, Refs. [23,24] show the representation and manipulation of functional and mathematical constraints. Furthermore the field of graph-grammars provides an automated synthesis of function structures as shown in [25,26]. These more abstract representations of design knowledge contribute to an all-encompassing approach on graph-based design representation.

The working group of the authors consequently focused on the development of an abstract, graph-based design representation where geometry as well as functional and mathematical design information can be treated as first class citizens. Not much other research groups except Schmidt [27,28] seemed to share at the time this view. Using a completely abstract, graph-based design representation and a graphical rule-based synthesis mechanism, an early version of “graph-based design languages” was capable to automatically design and generate space stations [29,30], satellite designs [31,32], aircraft families [33], airships [34] and automotive structures [35,36]. Using a home-grown, self-developed graph representation was however soon found to hinder a clear and concise knowledge representation which is easily accessible and readable by others.

As a consequence, the use of the internationally standardized Unified Modeling Language (UML) as formal representation syntax for graph-based design languages was investigated by Reichwein [37]. Generating automatically consistent CAD-, MBS- and controls models from graph-based design languages in UML this work showed the integration capabilities of graph-based design languages on the basis of UML across various engineering disciplines. Similar works showing other abstraction and integration aspects of the methodology are from Arnold [38] on the digital factory integration and generation, by Beilstein [39] on structural joining technologies, by Landes [40,41] on the automated tolerancing of aircraft structures and by Vogel [42,43] on the automated design, simulation, analysis and evaluation of off-highway exhaust systems. The work described here extends on the earlier works by Schaefer [31,32] on satellite design and is the first to show the interaction of different design languages based on UML in the broader context of consistent multi-disciplinary satellite design. The design language example in this paper series shows mathematical, functional, geometrical and behavioral models integrated in one generative design model. It ranges from component-level over subsystem-level to system-level evaluations.

Any multi-disciplinary design activity (of satellites) is also closely related to the methodologies developed in systems engineering. A related top down approach to integrate engineering disciplines is represented by the development and application of the Systems Modeling Language (SysML) [44–46].

In a spacecraft design process, a multitude of tools is used to manually create models of the design artifact. The design process using a design language differs from this approach. The graph-based design language serves as an abstract problem description which can be translated to the various engineering domains. In Fig. 1 the resulting design process is shown.

The design language describes the names, properties and relations of the vocabulary. To define the possible combinations of the vocabulary, rules are formulated. The execution sequence of the rules to build up a design is described by means of programs. The programs and rules are compiled by a design compiler to the design graph. From the design graph, models in different engineering domains can be generated. A feedback loop enables the control of the design process from a given design graph.

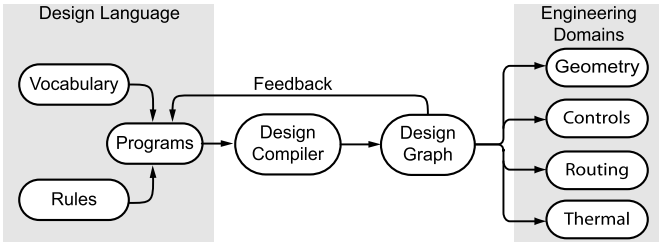

Fig. 1. Process when using design languages [47].

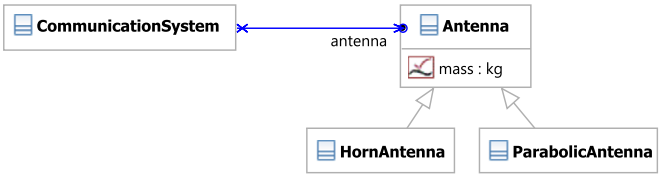

Fig. 2. Example of a Class Diagram.

Fig. 3. Simple rule with left (LHS) and right hand side (RHS).

2.1. Vocabulary

The vocabulary of the design languages is modeled in the widespread standard of the Unified Modeling Language (UML). The single words of the vocabulary are modeled as UML-Classes. The name of the Class refers to the engineering artifact. The properties of the Class describe the attributes of the artifact. In Fig. 2 an exemplary Class Diagram is shown.

In the diagram the relation between the *CommunicationSystem* and its *Antennas* is shown as a link. The *Antenna* can either be a *HornAntenna* or a *ParabolicAntenna*, which both inherit the property “mass” from the upper class *Antenna*.

2.2. Rules

The rules to create instances of the classes are expressed in a graphical left-hand side (LHS)/right-hand side (RHS) schema. The LHS is the conditional part which is used to search the design graph. The RHS is the execution part which is used to define a change on the design graph. In Fig. 3 an exemplary rule is shown. This rule searches on the LHS an instance of a *CommunicationSystem* and appends on the RHS an instance of an *Antenna* to it.

2.3. Programs

The programs are encoded using UML Activity Diagrams. In Fig. 4 an example activity is shown. The execution of the program starts at the initial node of the activity. Then the first rule *Rule1* is executed. The second block “Interface” “calc” represents a call to an external software that solves the equation system. Then an UML Decision Node is used to make a query on the design graph about a specific model property. The condition formulated in the Object