

# Generating Simulation Models from UML - A FireSat Example

Johannes Groß, Stephan Rudolph  
Institute for Statics and Dynamics of Aerospace Structures  
University of Stuttgart  
gross@isd.uni-stuttgart.de, rudolph@isd.uni-stuttgart.de

**Keywords:** complex system design, model-driven engineering, design languages, UML, model generation

## Abstract

The use of graph-based design languages in UML for complex system design is motivated. The FireSat mission from literature is modeled in different UML classes representing the systems, subsystems and parts of the satellite. A rule-based creation mechanism for the instances is shown along with an executable activity diagram for the definition of the design sequence. The generation of simulation models is demonstrated in the field of geometry (in OpenCascade), thermal simulation (in Esatan-TMS) and behavioral analysis (in Matlab-Simulink). The paper closes with an illustration of the flexibility in the generation design variants. based on design languages.

## 1. DESIGN PROBLEM

In modern product development a multitude of simulation models is involved in the design process. Due to the multidisciplinary nature of such a design process, the consistency between the models represents a major problem. Propagating the changes in the evolution of the product design by hand across the different domain specific models, is a time consuming and error prone task (see Fig. 1a).

### 1.1. Model Diversity

To reduce the work lost in these manual model transformations, interfaces between different engineering simulation tools have been developed. For engineering data exchange a broad standard (STEP) has been developed. Aiming in the same direction, the key players (Dassault, Siemens, ...) in the geometry modeling market create proprietary process chains. In these developments two points are crucial. Firstly, the number of interfaces grows in an all-with-all exchange situation (Fig. 1a) with the disadvantageous factor of  $n * (n - 1) / 2$  compared to  $n$  for the exchange via a central-model as shown in Fig.1b). Secondly, a bidirectional interchange between semantically flat application program data is sometimes even impossible due to the different dimensionality of these models [Rud06]. The simple and appropriate way out of this dilemma can be found in a semantically rich and more abstract central data model from which the application program data can be generated straightforward.

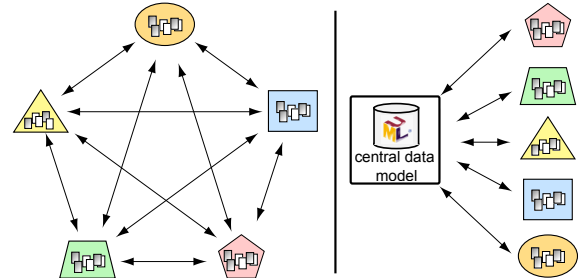


Figure 1. a) "All-With-All" vs. b) "Central-Model"

### 1.2. Design Languages

The approach of a design language as described in [ArR12], [Rud06] and [ScR03] avoids that such a central data model as in Fig. 1 which is typically large and complicated needs to be created manually. Alternatively, in a design language (Fig. 2), the engineering objects represents the vocabulary and the required model transformations represent the rules, i.e. the grammar of the design language. For this work, the definition of the vocabulary is expressed in the Unified Modeling Language (UML). In a production system, the rules are executed in order to instantiate the vocabulary classes. This compilation process builds up the central data model. From this high-level central data model, different interfaces

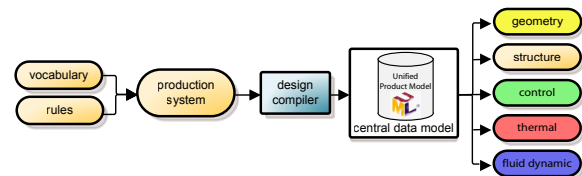


Figure 2. Process Chain to Create a Central Data Model

generate the simulation models. By this approach it is possible to address a multitude of simulation programs. In the presented paper, the compilation of the design language is done with the "DesignCompiler 43". This is an eclipse-based software tool for the compilation of design languages formulated in UML. The design compiler can process design languages on all abstraction levels to the full detail of the simulation models. This leads to a completely automated process within a unified framework. The implementation of the design language is the engineers skill, the generation of the models is mechanistic compiler work.

### 1.3. Limitations of the state-of-the-art process

With the design language process chain from Fig. 2 several limitations of the state-of-the-art design process can be overcome. Currently, in engineering design the process chain runs most often from a geometry paradigm towards the subsequent simulation models. The different engineering domains are interfaced alike Fig. 1a) directly and often the data exchange process requires manual rework. Within a design language, the number of interfaces can be reduced according to Fig. 1b) and the information flow can be organized in a hierarchical manner. Thus more abstract design decisions (e.g. topological ones) can be automatically compiled into their implementations in the different engineering domains. For instance, the decision on which plate a battery of a satellite is mounted on, is a topological decision which has to be propagated downstream into many subsequent and detailed simulation models (CAD, FEM, Thermal, etc.).

## 2. SATELLITE DESIGN LANGUAGE

In this paper, the FireSat example from [LaW99] is modeled in a UML model. The FireSat mission describes a satellite flying in low earth orbit for earth observation in the infrared band. The payload of the satellite is supposed to observe forest fires in the US. The model of the satellite is supposed to enable the layout of most of the subsystems on a conceptual level. In the following sections, a selection of some class diagrams from the model are presented along with short descriptions of their purpose.

### 2.1. FireSat Payload

For the payload an optical instrument is laid-out. The class diagram (Fig. 3) for the payload consists of 5 classes.



Figure 3. Classes for the FireSat Payload

### 2.1.1. FireSat Class

The "FireSatPayload" class calculates the requirements from the payload according to given approximations from [LaW99 p. 296]. These approximations are based on the aperture size of the instrument. This size is calculated in the other classes of the payload diagram.

### 2.1.2. Optical Instrument

The three classes "OpticalInstrument", "SensorOptics" and "SensorRadiometry" contain a calculation of all the relevant parameters of an optical instrument in this stage of design. The equations contained in these classes are given in [LaW99] in table 9-15 on page 287. From these calculations, the focal length and the aperture diameter are obtained as first approximations for the size of the instrument. Furthermore the noise equivalent temperature difference is an important value to meet mission requirements. The integration time combined with the pixel size serves as input for the attitude control system design.

### 2.2. Orbit Classes

For the definition of an orbit, the "OrbitParameters" class calculates the orbital period and the number of eclipses over the satellite life time. The orbit is given in Keplerian orbit elements. If the orbit is circular, the period is calculated by the "CircularOrbit" class. The "OrbitEclipse" class calculates the night and daytime for the satellite according to [Gil02]. If different ground stations shall be used for the mission, the class "GroundStation" can be instantiated multiple times.

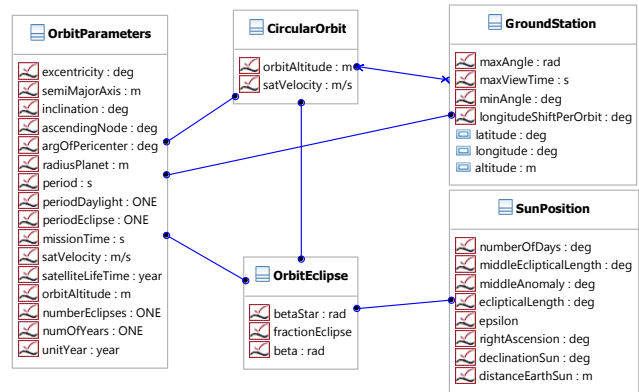


Figure 4. Classes for the orbit definition

## 3. DESIGN LANGUAGE RULES

The classes shown in the last section are instantiated by graphical rules. An example rule for the creation of the basic orbit parameters is shown in Fig. 5. The rule consists of two sides, a left hand side (LHS) with a search pattern and a right hand side (RHS) with a pattern of the partial graph after

execution of the rule. If an instance is mapped from the left hand side (LHS) to the right hand side (RHS) it is kept over the operation and marked with equal colors on both sides of the rule. If the instance is only on the left hand side, it will be deleted. Instances occurring only on the right hand side are created by the rule. In the following, a few different rules from the FireSat design language shall be explained.

### 3.1. ‘Axiomatic’ Rules

Axiomatic rules do have an empty left hand side. These rules will be executed without any (pre-)conditions. The right hand side of the rule will be inserted in the model. As shown in Fig. 5, the orbit parameters for the satellite layout are created by this rule. The parameters consist of the instance of the class "OrbitParameters" with given values for the inclination, ascending node and the argument of the pericenter.

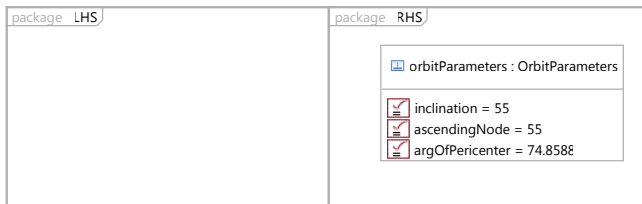


Figure 5. Rule for the initial orbit creation

### 3.2. ‘Insertion’ Rules

More often the rules have a condition on the left hand side. Either because the created instance has to be connected with some existing instance or because it makes only sense to execute the rule if some other system element already exists. In Fig. 6 an instance of the class "SensorOptics" is created under the condition of the existence of instances of the classes "FireSatPayload" and "OpticalInstrument". The name string of the instance can be given in such a rule by a regular expression. If any instance of the class shall be matched, the expression ‘.\*’ can be used.

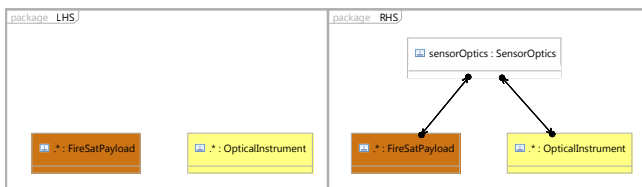


Figure 6. Rule to connect a sensor optics instance to the payload

In Fig. 7 an example for a bigger pattern in the RHS of a rule is given. Due to the explicit modeling of the geometry in this design language, some rules become quite large. In this example a basic geometry for the helix antenna is created.

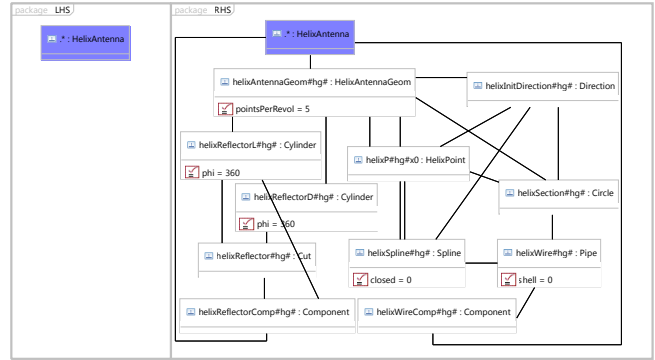


Figure 7. Rule to create geometrical definition of a helix antenna

### 3.3. ‘Architectural’ Rules

In graph-based design languages, architectural definitions can be described by the links between the different instances in an elegant manner. In Fig. 8 the architectural definition for different instances is shown. The instances are all searched on the left hand side of the rule and mapped to the right hand side. On the right hand side, only the links between the instances are created.

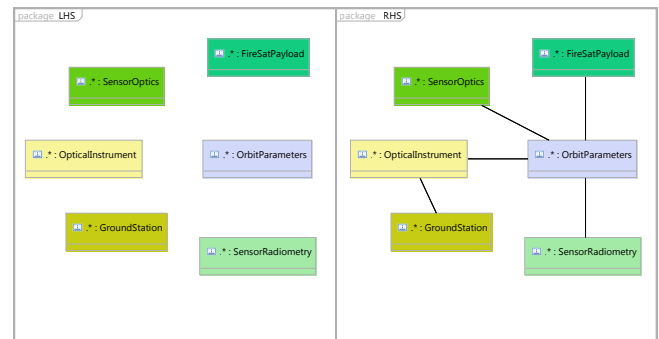


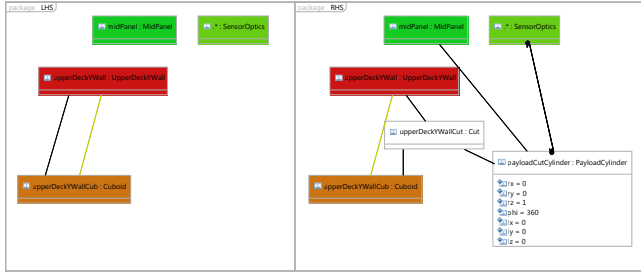
Figure 8. Rule to connect orbit instances with the payload

### 3.4. ‘Modification’ Rules

With the rule mechanism, not only links and instances can be added, but also existing links and instances can be modified. In the rule shown in Fig. 9 the rule for the cut-out in the upper deck wall of the satellite is shown. In this case, the link from the instance representing the upper deck wall to its cuboid has to be deleted and the newly created "Cut" instance takes this role. Under the cut, the cuboid and the cutting part, in this case a cylinder, are placed.

### 3.5. ‘Non-visual’ Rules

For all operations on the model, which can not be expressed graphically in an intuitive and straight forward manner, rules can also be written in Java. This allows arbitrary



**Figure 9.** Rule to cut a hole for the telescope in a satellite wall

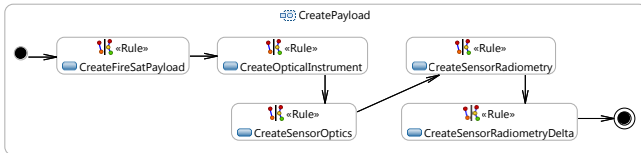
direct modifications of the UML instances.

## 4. DESIGN LANGUAGE EXECUTION

The creation of the satellite model starts with the creation of the payload. The requirements of the payload for power, attitude control, data down link and a mechanical structure are then satisfied by the different subsystems. The subsystems can themselves invoke further parts or even other subsystems to be created as described in detail in [ScR05].

### 4.1. Creation of the Payload

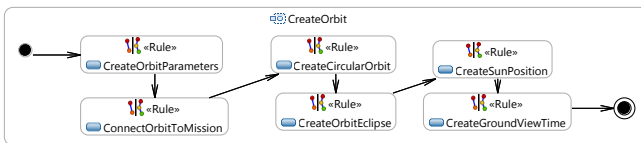
The instantiation of the payload classes is done within five rules shown in Fig. 10. Firstly the common "FireSatPayload" class is created and subsequently the classes for the calculation of the optical instrument are instantiated and linked with the existing instances.



**Figure 10.** Activity for the payload creation

### 4.2. Creation of the Orbit

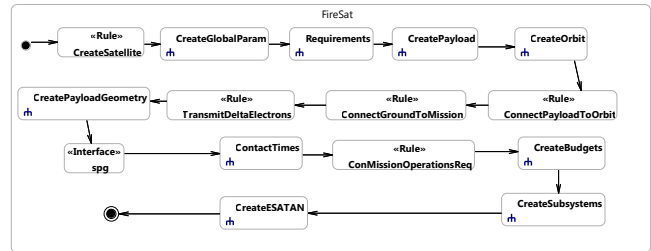
The creation of the orbit is done in a separate activity (Fig. 11) independent of the FireSat mission. Only the orbit parameters have to be adjusted to comply the missions needs. After the generation of the orbit, the orbit classes are connected to the mission classes as shown in the rule in Fig. 8.



**Figure 11.** Activity for the orbit creation

### 4.3. Creation of the Satellite

In Fig. 12 the activity for the whole satellite creation process is shown. The different actions in the flow are either rules, subprograms or interface calls. Rules have already been described above. Subprograms are just another activity like this one but they may be called from another UML model like the subprogram "CreateOrbit" in Fig. 11. An interface call can be used to trigger e.g. the solution of the current equation system of the model.



**Figure 12.** Main Activity of the FireSat Design Language

In Fig. 13 the model of the created instances after the first line in the activity above (Fig. 12) is shown. The blue lines represent links between the instances. Due to several roles an instance can play in respect to another one, multiple links can exist between the instances. The values within the instances result from the evaluation of the analytical equations given in the classes. The results obtained here can be compared with table 9-15 in [LaW99].

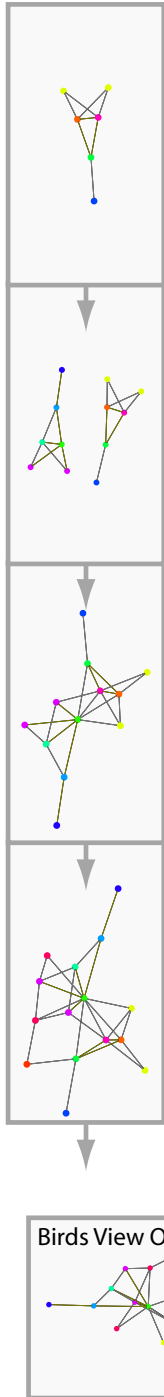
In the lower right corner of Fig. 13, an alternative view on the model is shown. In this view each instance is a colored circle and the links are gray lines. This birds' eye view is introduced to be able to give an impression of the complexity of the model at the end of the design process on the lower left corner of Fig. 13.

### 4.4. Design Driver Analysis

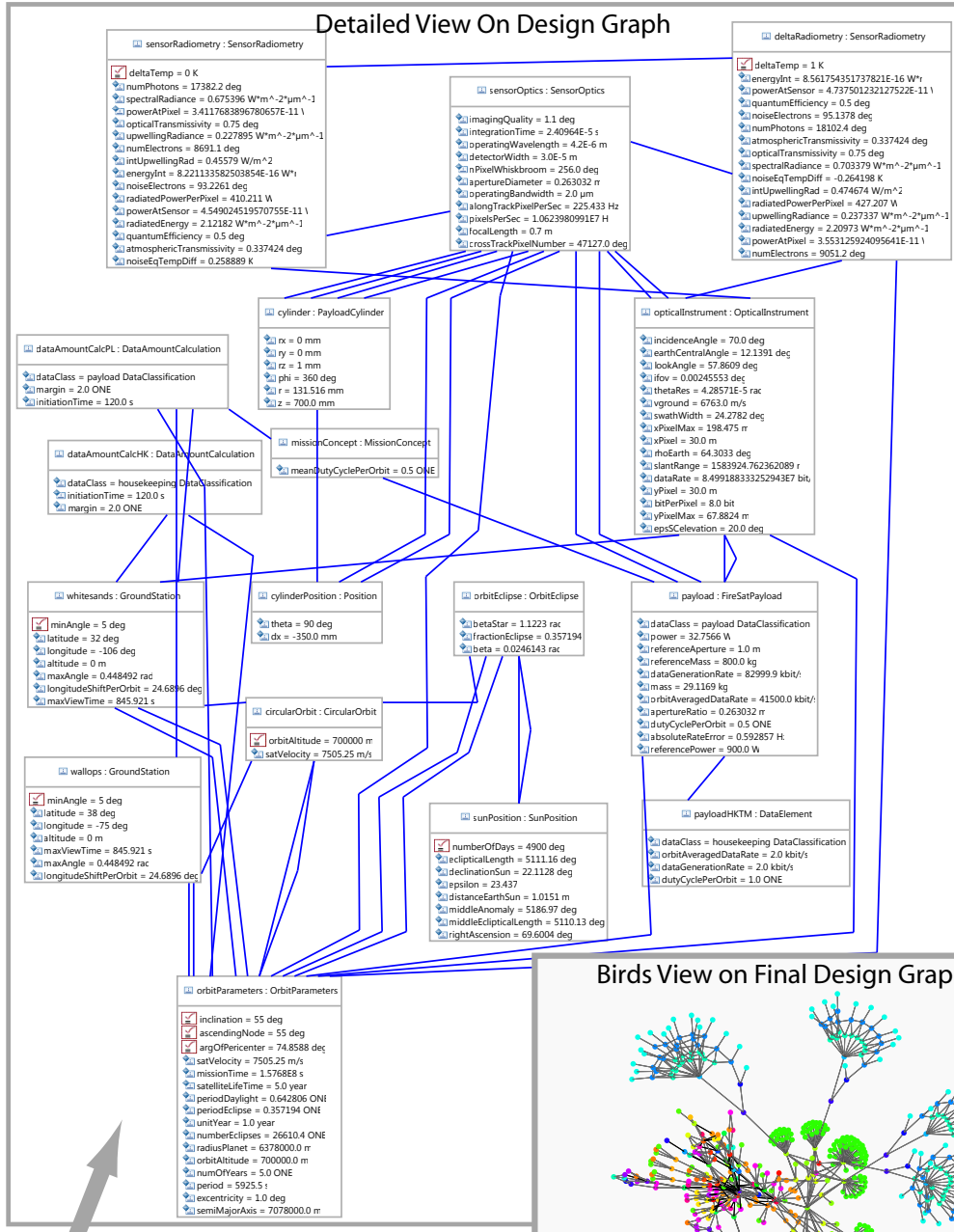
For the evaluation of the analytical equations the solution order of the various equations is found algorithmically by a so called solution path generator based on [Ser87][BoR03]. With the given solution path, the calculation of the values is executed by a computer algebra system (CAS). In this case Mathematica from Wolfram Research is used. Due to the general formulation of the mathematical expressions it is also possible to use other systems like Yacas (open source java CAS). From the solution path a graph can be generated, to analyze the influences and locate design drivers in the current design.

In Fig. 14 a cutout of the solution path with the payload and orbit instances is shown. The names of the nodes are built after the pattern "instance name", "variable name". The arrows point from the values used for the calculation to the values which are calculated. Every row represents one solution step

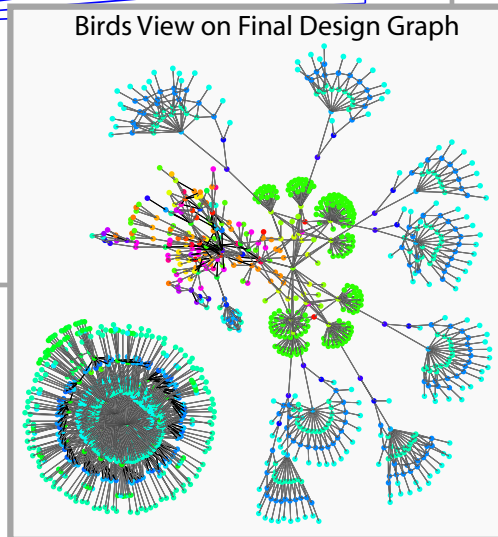
### Evolution of the Design Graph



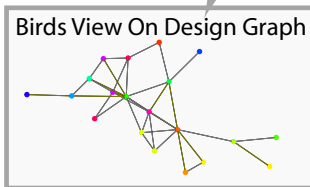
### Detailed View On Design Graph



### Birds View on Final Design Graph



### Birds View On Design Graph



**Figure 13.** On the left hand side, the evolution of the design graph in the beginning is shown. The big graph is the detailed view of the instances for the payload and the orbit. The graph on the lower right corner is the corresponding graph in a birds view. The graph on the lower left corner shows the final design graph with 1467 nodes and 2619 edges and contains all necessary information for the generation of the simulation models (CAD see Fig., Thermal see Fig., Simulink see Fig.).



in time i.e. all variables in one row can be solved in parallel. The values in the subsequent rows are dependent on at least one value from above.



Figure 14. Excerpt of the solution path for the payload instances

## 5. GENERATING SIMULATION MODELS

For the generation of the geometry-based simulation models an abstract geometry representation is created in the semantically rich central model. After the execution of the constraint solving mechanism described above, the solution values of the equations are written to the instances. Subsequently the semantically rich central model can be translated to different simulation models.

### 5.1. OpenCascade Geometry Model

The OpenCascade model consists of either primitives, like cuboids and cylinders, or solids built up from scratch by points, lines and faces. At this time, the geometry within the updated abstract geometry description is being integrated into the project. Fig. 15 below is already generated from this updated model. Before using this approach the geometry was modeled in Catia-specific UML extensions as described in [Gro09] and in more detail in [Rei11].

### 5.2. Esatan Thermal Model

Esatan-TMS is a software package for the prediction of both steady-state and transient temperatures and heat flows in a thermal network using the lumped parameter or finite

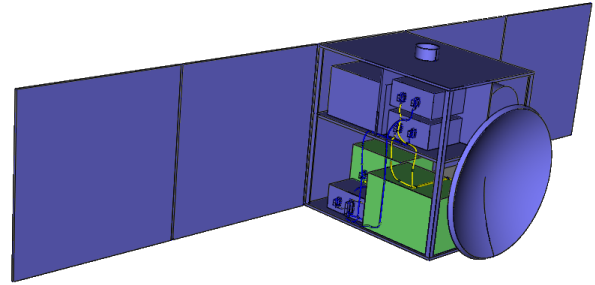


Figure 15. CATIA Model of FireSat

difference method [ESA11]. The model consists of simplified geometric representations of the satellite parts. The initial interface export capabilities of the simulation model are described in detail in [Koc10]. In addition to the geometry of the model, the material properties for heat conductance are exported. The surface properties for the heat radiation and the topological information for the generation of conductive conductors are also exported from the central model. In Fig. 16 the exported geometry is shown in the Esatan environment.

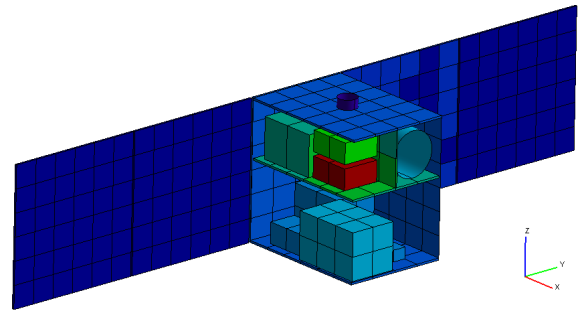


Figure 16. Esatan Thermal Simulation Model

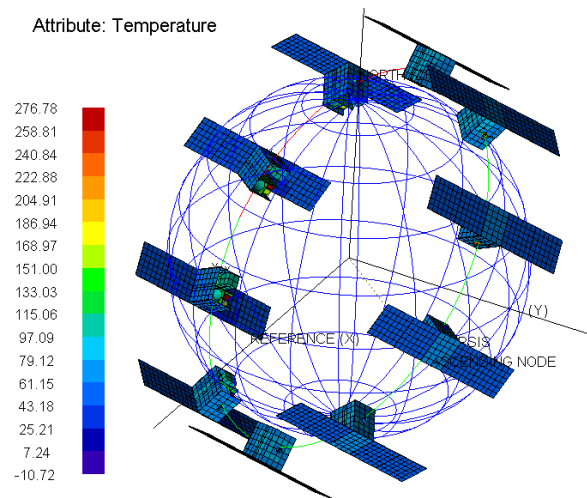


Figure 17. Esatan Orbit Simulation Generated from a graph-based design language in UML

The model has a relatively low number of nodes due to the early design phase of the satellite. The number can however be set either centrally at one point in the design process, or be adjusted individually for each part. The whole model can be generated along the design phases of the satellite and the additional rework in the Esatan software to start a simulation is reduced to a few clicks. The parabolic antenna is still missing in Fig. 16 due to the above mentioned changes in the geometry modeling. To include the antenna, the translation of the paraboloid has to be defined. To simulate the mission, the orbit information is written to a radiative case file. This file is the starting point for a ray-tracing based analysis of the view factors. In Fig. 17 the orbit with an inclination of  $55^\circ$  defined in the rule shown in Fig. 5 can be recognized. It is a circular orbit and the satellite is in nadir pointing mode.

### 5.3. Simulink Orbit Simulation Model

For the simulation of the satellites' attitude control, a Simulink model is generated. The model generation is described in detail in [Rie11]. The generation capability comprises a simulation with different actuators, sensors and pointing modes. The control algorithms are adapted to the situation and a visualization in a VRML viewer can be generated. In Fig. 18 a sequence of the simulation of an acquisition maneuver is shown. The satellite is first in a free floating mode and then nadir pointing is switched on. In the simulation, differ-

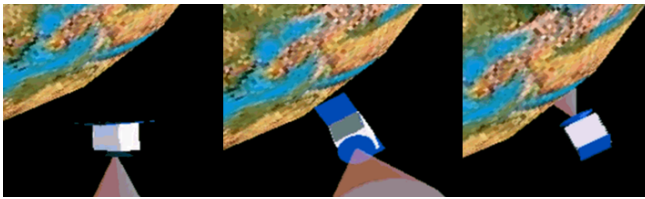


Figure 18. Orbit Simulation in Simulink

ent situations of the satellite operations can be investigated. In Fig. 19 an example measurement of the offset of the three axes from an abruptly set target direction is shown over time.

## 6. LESSONS LEARNED

In a state-of-the-art design process, for the shown analysis of a satellite design, several rather intensive steps of manual modeling are necessary. The analytical lay-out of the different subsystem has to be carried out. Then a geometrical model has to be set up. In respect to the properties of the CAD model both, a consistent Simulink simulation and a thermal Esatan model have to be created. Just recently the latest version of Esatan supports geometry import. Furthermore, the information architecture of the state-of-the-art design process prevents the generation of application specific data from more abstract data. For example the topological information of the

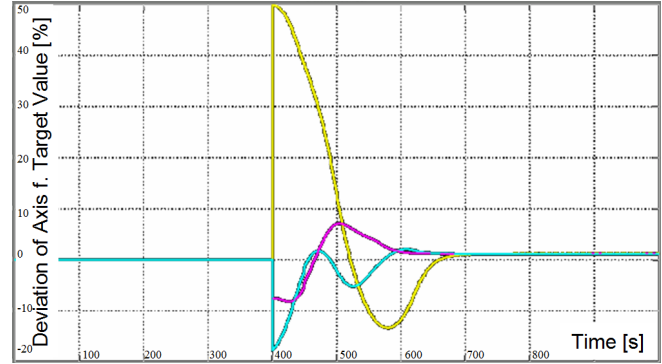


Figure 19. Results of Simulink Simulation

mounting plane of a box can not be mapped from the geometrical model automatically to the thermal model. However, for the thermal model this represents a crucial information.

### 6.1. Graph-based design languages

With the method of graph-based design languages in UML, the upfront investment is much higher compared to the state of the art. Since searching a file once manually on your computer might be much faster than implementing a search algorithm first, the design language process can show its strength only in the repeated execution with different initial setups. Then however, the generation of alternative design variants becomes very straight forward. It is possible to change some initial parameters, as for example the size of the satellite structure, and all the subsequent design processes are still fully executable. In Fig. 20 different variants of the given design are shown. In the first line the example presented above is represented, in the second line, the size of the structure was increased. In this survey it can be seen that the solar panel system has not created secondary panels but only a body mounted one. In the third line, the structure is chosen way too small just to point out the fact that in this case five solar panels would be required to fulfill the power need against one or three panels in the other two variants.

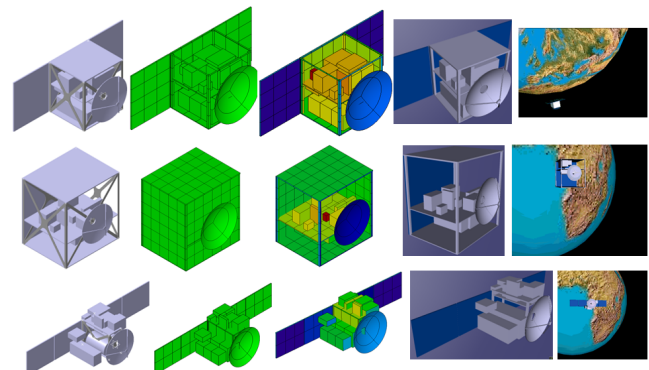


Figure 20. Variations of the given Satellite

## 6.2. Scalability

In computer science scalability describes the performance of a software system when the input data is increased. In the approach of graph-based design languages, UML models are used and transformed, so the performance of the operations on the UML models are crucial to the performance of the design compilation process. In this project, the models of the satellite are not larger than a few megabytes, even with the shown complexity including a detailed geometry model. The largest model is the model for the generation of the orbit simulation with 17MB. This model is still performing well even though the used open source UML editors are not specifically designed and optimized for large data amounts. The problem of scalability can yet be overcome quite easily due to the procedural nature of the design tasks. When complex geometry objects are created, they can for example be saved as a STEP-file and integrated as an existing component. Then they need only one string for the file name and the model can be kept light. With this technique of model transformations and the adequate level of abstraction in the information representation it is even possible to develop entire aircraft panels and the subsequent digital factory within one design process without performance issues (shown in [ArR12]).

## 7. OUTLOOK

In this paper, a detailed analysis of the FireSat example is presented in a short form. The basic design method in form of graph-based design languages in UML is motivated and described, relies however on a huge amount of already published knowledge ([LaW99], [ScR05], [Gro09]). An extract of the required classes are shown and their purpose is explained. Some design rules are illustrated to clarify the design language approach. The execution of the automated design process is shown along with the activity diagrams to manage the rule execution. The abstract central data model is shown in two different views along with a design driver analysis which is available all along the design process. To illustrate the effectiveness of the approach in the generation of simulation models, different models, all automatically generated from the graph-based design language represented in UML, are shown. With more space available the authors could go into more detail in the different subsystems as well as in the different simulations. In this compact form however the paper can only serve as an appetizer for a closer look on graph-based design languages, their potential for simulation model generation and their underlying novel design methodology.

## REFERENCES

- [AlR03] Alber, R. and Rudolph, S.: "43" - A Generic Approach for Engineering Design Grammars. Proceedings of American Association for Artificial Intelligence, Spring Symposium, Stanford, 2003
- [ArR12] Arnold, P. and Rudolph, S.: Bridging the gap between product design and product manufacturing by means of graph-based design languages, TMCE 2012 Symposium, Karlsruhe, 2012
- [Bal00] Balanis, C.: Antenna Theory Analysis and Design. 3rd Ed. Wiley-Interscience, Hoboken, New-Jersey 2005
- [ESA11] [http://www.esa.int/TEC/Thermal\\_control/SEM58\\_ZYNZBG\\_0.html](http://www.esa.int/TEC/Thermal_control/SEM58_ZYNZBG_0.html) last access: 2011-11-13
- [Gil02] Gilmore, D. G. ed.: Spacecraft Thermal Control Handbook. 2nd ed. Reston, Virginia: The Aerospace Press. 2002
- [Gro09] Gross, J. et al.: An Executable Unified Product Model Based on UML to Support Satellite Design. AIAA 2009-6642, AIAA Space Conf. Pasadena, California, 2009
- [Kar00] Kark, K.: Antennen und Strahlungsfelder, 3. ext. Ed., Wiesbaden, 401-405, Vieweg+Teubner, 2010
- [Koc10] Kocak, M.: Erstellung einer Schnittstelle zur generischen Thermalsimulation von Satelliten. Diploma Thesis, Institute of Statics and Dynamics of Aerospace Structures, University of Stuttgart, 2010
- [LaW99] Larson, W. J., and J. R. Wertz, eds.: Space mission analysis and design. 3rd ed. El Segundo, California: Microcosm, 1999
- [Rie11] Riestenpatt genannt Richter, M.: Eine Entwurfssprache zur Auslegung der Lage- und Bahnregelung von Satelliten. Diploma Thesis, Institute of Statics and Dynamics of Aerospace Structures, University of Stuttgart, 2011
- [Rei11] Reichwein, A.: Application-specific UML profiles for multidisciplinary product data integration, PhD Thesis, University of Stuttgart, 2011
- [Rud06] Rudolph, S.: Know-How Reuse in the Conceptual Design Phase of Complex Engineering Products Or: Are you still constructing manually or do you already generate automatically? In: Tichkiewitch, S., Tollenaere, M. and Ray, P. (Eds): Proc. Conf. on Integrated Design and Manufacture in Mechanical Engineering 2006 (IDMME 2006), Grenoble, France, 2006
- [ScR05] Schaefer, J. and Rudolph, S.: Satellite Design by Design Grammars. Aerospace, Science and Technology (AST) 9, 81-91, 2005
- [Ser87] Serrano, D.: Constraint Management in Conceptual Design. PhD Thesis, MIT, 1987