An Executable Unified Product Model Based on UML to Support Satellite Design

Johannes Gross∗, Axel Reichwein∗ and Stephan Rudolph†

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

Dagmar Bock‡ and Rene Laufer‡

Institute of Space Systems, Pfaffenwaldring 31, 70569 Stuttgart, Germany

Due to the multidisciplinary nature of satellite design and the mostly proprietary formats of engineering software tools, a multitude of heterogeneous computer models are employed during satellite design. A unified central product model can guarantee data consistency and manage the interdependencies between the different isolated models. This paper proposes to use the Unified Modeling Language (UML) as the cornerstone for a unified central product model. The approach has been applied to the design phase of the Perseus satellite which is part of the Stuttgart Small Satellite Program. This paper presents the UML lightweight extensions necessary to represent geometric features authored in CATIA and control system features authored in Matlab/Simulink in a common UML model. Next to the unified description of product data, the use of UML to represent design processes was investigated. An iterative design sequence consisting of several CATIA- and Matlab/Simulink-specific evaluations was formulated as an executable UML activity diagram. This study shows that the UML, which is already widespread for software design, has the potential to become the future unifying product modeling language which can close the gap between the traditional distinct mechanical, electronic and software-related disciplines.

I. Introduction

Satellite design is a heavily multidisciplinary task involving design evaluations based on a multitude of software tools. Due to the specialization of software tools for specific disciplines, the design process consists of handling and managing data of many different formats. Satellite design, as most design activities, is never optimal at the first go but consists of several design modifications to reach an optimal configuration. Each design change thereby affects data from numerous disciplines. However, computer-based evaluations of a satellite design are only meaningful if the underlying data is consistent with a specific satellite configuration. Design changes during satellite design therefore require a framework to guarantee data consistency across various disciplines. Moreover, such a framework is not only useful for satellite design but for all multidisciplinary product design activities.

A. Unified product model for data consistency

Data consistency is achieved by linking product data. The links ensure that a local data modification is forwarded globally to all product data and that the dependent data can be updated accordingly. Product data needs to be available in a common format before it can be connected through links. The data is therefore gathered in a common model. Links are then defined in the corresponding format while translators convert the information from the common model to the application-specific models and vice versa. Although this approach was very often undertaken,1–6 no unified standard has yet evolved to describe product data in a common format which can be used across various disciplines and for the design of different product types.

∗Research Assistant, Similarity Mechanics Group, University of Stuttgart, AIAA Member.
†Associate Professor, Similarity Mechanics Group, University of Stuttgart.
‡Research Engineer, Electric Propulsion Systems, University of Stuttgart.
Instead of reinventing the wheel by proposing again a new product modeling language, this paper proposes to use the Unified Modeling Language\(^7\) (UML), which is already widespread in software design, to describe product data uniformly in a unified product model.

Approaches to describe product data uniformly have been successful in specific disciplines. In mechanical engineering, the STandard for the Exchange of Product model data\(^8\)\(^,\)\(^9\) (STEP) is a collection of standards for single disciplines to describe product data independent of software tools. It includes for example standards for 3D geometry, computational fluid dynamics and structural analysis. STEP standards mainly cover mechanical engineering data which is often geometry-related. In software engineering, the UML is used to describe software systems independent of a programming language or of an underlying operating system. Even though STEP and UML are important non-proprietary widespread standards in their respective fields, no standard currently exists to build a bridge between these. As a consequence, linking mechanical and software engineering data is a challenge.\(^10\)

B. Object-oriented modeling

The description of multidisciplinary data in a unified product model should be based on features common to all product data. Although product data is so diverse, it shares common features on an abstract level. The general basic concepts of templates and template instances are omnipresent in many disciplines when editing product data (Fig. 1). In 3D geometry authoring, geometry is defined in parts which represent reusable geometric templates. The geometric assembly model is composed of part instances. In controller design, dynamic systems are described through block diagrams based on block libraries. The block diagram is composed of block instances from predefined block types which play the role of templates. Furthermore, relationships between templates are very frequent such as dependencies between geometric parts or composition relations within the hierarchy of subsystems inside a block diagram. The widespread use of templates, template instances and their relationships represent features which are common to many domain- and application-specific models.

In software design, template-based concepts are described formally through object-oriented modeling. It is used to describe software systems based on the basic concepts of classes and objects, which correspond in general terms respectively to templates and template instances. The object-oriented modeling concepts arose through the development of the Simula\(^11\) programming language in 1967. Objects represent encapsulated units which contain data and methods. The object-oriented modeling concepts describe the classification, composition and communication of classes and objects and have first been used for software programming. As a result, they have contributed through a higher modularization to an increased maintainability and reuse of software code. They are now present in modern programming languages such as Java or C++. The object-oriented modeling concepts are generic and not specific to a discipline. They are therefore recognizable in many other discipline- and domain-specific models\(^12\) which also share template-based concepts. Object-oriented modeling principles seem to be the most widespread non domain-specific feature shared by nearly all models. Furthermore, it is most likely that the applications will integrate more object-oriented modeling concepts in the future to achieve a higher modularization and easier reuse of information. As a consequence, the unified product modeling language should be built upon the widespread object-oriented modeling concepts to ensure the most intuitive, understandable and unified representation of domain- and application-specific data.

C. UML-based unified product model

The Unified Modeling Language was established in 1997 and resulted from the fusion of the previously competing object-oriented modeling methods of Booch, Rumbaugh and Jacobsen. The UML is the most well known standard for object-oriented modeling. Since a unified product model should reflect the prevalent use of template-based features which can be formally described through object-oriented modeling concepts, the
unified product model should be built upon the standard for object-oriented modeling which is UML. Furthermore, the UML can be extended to describe domain-specific information and its diffusion is encouraged through its non-proprietary specification.

The UML is composed of modeling elements to describe information of both static and dynamic nature. Only a few elements related to classes and objects, which are respectively described in class and object diagrams, were used in this study to represent the application-specific models related to the Perseus satellite. The most important modeling element in the UML is the class. It shows the features, constraints and semantics which are common to a group of objects. These are often also called instances or, according to the UML specification, instance specifications. They can represent anything, whether software or physical entities. The relationships between classes, such as associations, dependencies or generalizations are depicted in a class diagram.

A UML model can describe domain-specific information by being extended. It is thereby important to favor the slimmest possible UML extension in order to share the highest interoperability with other UML users and tools. A lightweight extension mechanism in the form of stereotypes was therefore used for this study. Stereotypes can be added like tags to any UML modeling element and can own properties for further detailing. They can also be represented through an icon to better recognize their associated domain-specific information. The stereotypes specific to a domain are regrouped in a separate document called profile.

The approach using a central UML-based product model has been applied to the design of the Perseus satellite. It is described in Section II together with its corresponding CATIA and Matlab/Simulink models. The mapping of the application-specific models into the unified product model is demonstrated. Section II also shows the linkage of CATIA- and Simulink-specific data in the common product model to guarantee data consistency. Section III displays a possible sequence of executable design steps described in a UML activity diagram. The conclusion presents the main results and an outlook.

II. Integration of the Perseus Models

The Perseus satellite is part of the Stuttgart Small Satellite Program from the Institute of Space Systems at the University of Stuttgart. The Program includes a small satellite mission to the moon for which two different electrical propulsion systems are developed. The Perseus mission is intended to accomplish the in-orbit test and the validation of new low-cost electric thruster systems. Afterwards, the satellite is anticipated to accomplish UV astronomy in the spectral band of 120 nm to 180 nm with an on-board telescope.

The satellite is designed with different engineering software tools. The two models considered in this work are a geometric model authored in CATIA and a dynamic model of the satellite authored in Simulink. The integration of the domain-specific data into the UML-based unified product model is realized by lightweight extensions in the form of stereotypes. The UML model elements representing the application-specific data are tagged with application-specific stereotypes. The Perseus application-specific models and their unified UML representation based on stereotypes are shown in the next subsections.

A. CATIA Model

Like most CAD tools CATIA enables the authoring of geometry according to object-oriented paradigms. Geometry is described in part or product documents. A part contains detailed geometry while a product describes an assembly of parts. They represent templates which can be instantiated and placed in different geometric assembly models. A CATIA product contains part instances and can itself be instantiated and placed in a larger product. Due to the object-oriented nature of geometry modeling in CATIA, the mapping of CATIA features into UML is often straightforward.

In Figure 2, the CATIA product of the pulsed plasma thruster (PPT) and an instance of it are depicted. The corresponding UML elements are drawn below. CATIA parts and products are mapped respectively into UML classes with the «catiaPart» and «catiaProduct» stereotypes. CATIA product and part instances are represented by UML instances. The PPT instance for example has the classifier PulsedPlasmaThruster.

CATIA-specific geometric properties need to be represented in the UML-based product model as they have an influence on the dynamic behavior of the complete satellite. For the integration of CATIA-specific measures into the unified product model, stereotypes can be applied to the properties of a related «catiaPart»

---

a DASSAULT SYSTEMES, http://www.3ds.com/products/catia/
or «catiaProduct» class. The CATIA-specific mass, center of gravity and inertia measures belong to a part or a product and are therefore described in corresponding UML class properties which are tagged with the related «catiaMass», «catiaCG» or the «catiaInertia» stereotype. As an example, the position and orientation of the thruster are described through properties tagged respectively with «catiaEulerRotation» and «catiaOriginShift» stereotypes (Fig. 2). The values for these properties are retrieved from the CATIA instance and are stored as literal string values in the slots of the UML class instance.

According to the CATIA geometric hierarchy, a product can contain further products or parts. The CATIA-specific composition hierarchy can be translated one-to-one into a UML class composition hierarchy. In Figure 3, the composition of some CATIA products and related UML classes belonging to the Perseus satellite is depicted. The class representing the top-level product in the composition hierarchy is labeled with a «catiaRootProduct» stereotype. The containment relations of the CATIA products are modeled with UML composite aggregations.

Figure 2. Left: CATIA product and corresponding UML class. Right: CATIA product instance and corresponding UML class instance

Figure 3. Selection of the CATIA model composition hierarchy and corresponding UML classes hierarchy
CATIA part instances are positioned in an assembly according to assembly constraints. They are translated into UML constraints. CATIA constraints are owned by a product, so the UML constraints are owned by the related «catiaProduct» class. The geometry referenced by the CATIA constraint is described through UML properties which are affected by the corresponding UML constraint. According to the type of the assembly constraint, the UML constraint is tagged with a specific stereotype such as «catiaAngle» or «catiaCoincidence». If necessary, the specific stereotype owns attributes for a complete description of the constraint. A «catiaAngle» stereotype for example owns attributes to specify an angle value and angle sector.

Changes in the UML model, such as the choice of parts or packaging strategy, are reflected automatically through the programmed interfaces in the CATIA model and vice versa. For large CATIA models, the complete translation of a UML model to generate a new CATIA model or vice versa may take too much time. To facilitate for example the update of CATIA parameters and assembly constraints, the «catiaUpdate» stereotype can be applied on UML properties and constraints. Only the CATIA features related to the «catiaUpdate» tagged UML elements will then be updated. This allows a quick update of a complete assembly consisting of many parts and composition levels.

Obviously, the UML model does not incorporate the complete CATIA geometric information but only captures the relevant geometric features playing a multidisciplinary role. In the work of Böhlke,16 more sophisticated CATIA features like user-defined features were integrated in the UML-based product model.

B. Matlab/Simulink Model

Mathematical relations between properties of the unified product model are described through UML constraints. Each contains a UML expression representing a symbolic mathematical equation. The orientation and position properties for example of the PulsedPlasmaThruster class in Figure 4 represent CATIA-specific measures which need to be converted in corresponding Simulink-specific A_PPT_Thr and O_PPT_Thr properties according to another reference frame. So the computation of the O_PPT_Thr property for example depends on the orientation and position properties as displayed in the blue-colored equation. The equations can be computed using the values of the UML instances stored in the UML model. To automatically evaluate the equations, Matlab is used as algebra system since most properties are matrices or vectors. The language attribute of the UML expression is set to “Matlab” so that only the Matlab-specific expressions are collected and evaluated. The results of the calculations are transferred back to the UML model. In the PPT instance of Figure 4, the O_PPT_Thr vector is computed by Matlab. The Matlab-specific UML elements are colored in blue.

The Simulink model of the satellite serves for the simulation of the attitude and orbit control system (AOCS). The model simulates the behavior of the actuators and sensors as well as the behavior of the on-board computer (OBC). The environmental conditions are also included in the model and thus different in-orbit satellite operations can be simulated. The testing of the thrusters for example is one of the important
operations to be simulated accurately in advance. Since the access to the Simulink model of the Perseus satellite was limited due to proprietary toolboxes, the Simulink model was launched and accessed from the Matlab command line through a Matlab function.

A Matlab function is represented in UML by an operation with the «matlabFunction» stereotype. In Figure 4, the startSimulation operation can be seen in the PulsedPlasmaThruster class. The input and output arguments of the Matlab function are described as UML parameters. The Simulink simulation based on the O\textsubscript{PPT}Thr and A\textsubscript{PPT}Thr properties, describing respectively the thrust origin position and orientation, is launched through the Matlab startSimulation function. The related UML operation and properties are colored in orange in Figure 4. The time until the satellite reaction wheels, under the application of a specific thrust origin and direction, have saturated is the simulation result simTime which is written back in the UML model.

C. Data Consistency

Changes in the CATIA-defined geometry of the satellite have an effect on its dynamic behavior described in Simulink. To automatically keep the Simulink model consistent with the CATIA model, a data exchange from CATIA to Simulink via the central UML-based product model has been implemented. After a change in the CATIA geometry, the CATIA measures are imported into the UML-based product model and the Simulink values are updated according to the CATIA values and constraints in the unified product model. The results of the calculations are then exported to Simulink to initialize a simulation which is consistent with the CATIA-specific geometry.

UML constraints link CATIA- and Simulink-specific properties, as both are represented in a common UML-based product model. The properties are linked through symbolic mathematical equations. As displayed in Figure 5, the Simulink-specific O\textsubscript{PPT}Thr property for example depends on the CATIA-specific orientation and position properties. The CATIA model of the pulsed plasma thruster (PPT) is shown on the left hand side. In the middle, the unified product model elements representing the PPT are depicted and on the right hand side, the PPT-dependent Simulink blocks contributing to the dynamic behavior of the complete satellite are shown. The UML class element in the top-center contains two attributes tagged with CATIA-specific stereotypes which mark the import of the orientation and the position of the CATIA product. The UML instance contains the values imported from CATIA as indicated in blue. The values required for the initialization of the Simulink model are below the CATIA values and indicated in orange. The constraint, marked partially orange partially blue, sets the link between the CATIA- and the Simulink-specific data.

Instead of creating consistency through constraints which link data, several application-specific stereotypes can also be applied on the same UML property. Consistency is then guaranteed without constraints as the same underlying UML value is then forwarded to the corresponding applications due to the overlapping stereotypes. The application of different stereotypes on the same UML property avoids the representation and linking of semantically equivalent application-specific properties in the central model. The mass value of a designed object might for example be required in several application-specific models but only needs to be described once in the central UML-based product model as it is unique.
III. Simulation

With the unified product model, design changes can be propagated from one engineering model to another. For propagation of changes, the subsequent import, linking and export of data are required. To enable an automatic propagation of changes, the interfaces between the UML-based product model and the application-specific models can be accessed programmatically in Java. The application programming interface of the UML-based product model is also in Java. As a consequence, a multitude of design alternatives can be examined automatically by running a Java program.

In this section, the use of the unified product model for an automated multidisciplinary evaluation of design alternatives is presented. The design changes are applied to the unified product model. Afterwards, the remote models are updated and evaluated. This is done in a loop process, to examine the design space automatically. The process steps are described graphically in a UML activity diagram.

In a test scenario including a dependency between a geometric configuration and a related dynamic behavior, the impact of different thruster orientations on the saturation time of the satellite reaction wheels was determined. The orientation of the satellite is disturbed by the torque the thruster applies on the satellite. The reaction wheels of the attitude and orbit control system (AOCS) counterbalance the disturbance torque by increasing their rotation speed in order to keep the satellite aligned. Over time, the reaction wheels build up stored momentum that needs to be cancelled. If the wheels have reached their maximum rotation speed, saturation is reached and the thruster has to be cut off. The aim is to maximize the saturation time of the reaction wheels by finding the optimal thruster orientation relative to the other satellite components. This design optimization requires to evaluate several satellite configurations.

In figure 6, the activity diagram for the automated evaluation of satellite configurations with different thruster orientations is shown. The process steps are described in UML as call operation actions which are connected with UML object flows to determine the order of execution. Process input values correspond to UML activity parameter nodes. In this example, the parameters are shown on the left hand side of the diagram. The upper input parameter contains the path to the UML model which is to be loaded. The two lower ones are used to describe the value and name of the relevant parameter to be changed for the evaluation. Thus every parameter of the unified product model can be accessed and changed easily. In this example, the angle of the thruster was changed around the x-axis (Fig. 7) of the satellite in 1.5 deg steps.

The activity diagram is executable by linking the diagram actions with Java methods which either modify the UML model or launch the import/export interfaces. The execution of the diagram in figure 6 starts at the filled black UML initial node on the top of the diagram. The first action loads the UML product model and the next one exports the first satellite configuration to CATIA. The thruster orientation has an impact on the thrust direction, the thrust origin, the center of gravity and the inertia of the satellite, which are therefore measured in CATIA and sent back to the UML model. UML constraints which link CATIA-specific and Simulink-specific properties are evaluated with Matlab. The Simulink simulation which determines the saturation time is then launched based on the UML model with the latest CATIA-specific measures. The simulation results are stored back in the UML model. The decision node, depicted as rhombus in the lower left corner, either ends the evaluation loop or starts a new evaluation based on a new parameter value which is set in the UML model. The CATIA- and Simulink-based evaluation via the central UML product model is then repeated.
Figure 7. Results of the evaluation of different geometric configurations

Figure 7 shows the results of the automated evaluation of different configurations. Each point of the graph represents a different geometric thruster configuration. On the horizontal axis, the orientation angle of the thruster around the y-axis (Fig. 7) is drawn. On the vertical axis, the resulting saturation time of the reaction wheels is depicted. The different colours stand for a change of the thruster angle around the x-axis (Fig. 7) of the satellite. The saturation time depends on the lever arm of the disturbance torque created by the thrust and of the satellite position within the earth magnetic field.

IV. Conclusion

This paper presents the UML lightweight extensions necessary to describe geometric features authored in CATIA and control system features authored in Matlab/Simulink in a common UML model. The mapping principles have been implemented to support an automatic bidirectional translation between the central UML model and the domain-specific models. Update mechanisms were also implemented to support a quick reconfiguration of models and avoid a time-consuming generation of new complete models. Furthermore, a design process was described as an executable UML activity diagram which could refer to the product data saved in the UML-based central product model.

By integrating enough information, the central product model represents a central data repository which can be used for knowledge-based engineering or for multidisciplinary optimization. External design frameworks only need to access the central model instead of many separate engineering models to create and evaluate new product configurations. A design process can be automated by applying a set of design rules, similar to the design compiler approach in Alber and Rudolph,\textsuperscript{17} on a UML-based product model.\textsuperscript{16} In software design, model-driven development consists of generating code based on a most often UML-based central model. By sharing the same UML-based central model, model-driven development techniques can also be applied to product design to generate engineering models.\textsuperscript{15}

Due to the extensibility of the UML, the central product model can further be adapted to integrate other domain-specific or tool-specific data. The presented UML extensions specific to CATIA and Matlab/Simulink can naturally be further improved to include more features. These application-specific UML extensions should ideally be defined in cooperation with the respective software vendors and the Object Management Group, which is a consortium aiming at setting standards such as the UML.

By the presented methodology of an executable unified product model, a standardized consistent integration of multidisciplinary data was enabled. This approach subsequently facilitates the creation of many customized design alternatives and their evaluation. Just as the UML proved helpful in providing software developers with a common ground and in producing better software, the use of a unified product model based on a widespread modeling language such as the UML can ultimately improve the communication between engineers originating from different disciplines and lead to better engineered products.
References