INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN, ICED'07
28 - 31 AUGUST 2007, CITE DES SCIENCES ET DE L'INDUSTRIE, PARIS, FRANCE

ON A MODEL DRIVEN APPROACH TO ENGINEERING DESIGN

Peter Hertkorn[1] and Axel Reichwein[2]

Institute for Statics and Dynamics of Aerospace Structures, University of Stuttgart, Germany
[1]hertkorn@isd.uni-stuttgart.de
[2]reichwein@isd.uni-stuttgart.de

ABSTRACT

An important factor for the success of today's companies in global competition is the continuous improvement of the product development process within a product's life-cycle. However, complex technical products affecting several engineering domains and manifold product variants lead to frequent iterations within the product development process. To decrease the development time of a product, it is therefore necessary to use appropriate software tool support and to adjust the numerous tasks in the design phase. In this paper an approach to engineering design is shown according to Model Driven Architecture that is attracting wide attention in software engineering as well as in systems engineering. Based on a central model that allows regarding the dependencies between data of different domains together, this approach uses model-to-model transformations to increase the level of detail of a product design continuously while maintaining a consistent model. The resulting model contains all necessary information to generate code for the connected programs to perform the analyses. The analysis programs return the results back into the model, giving the possibility to restart the model refinement step with a modified model. By automating this task sequence, it is possible to optimise the product design by maintaining a consistent model at the same time.

Keywords: Engineering design, model driven architecture, product lifecycle management

1 INTRODUCTION

One of the most important objectives for companies in global competition is a shorter time-to-market for products by minimizing the total cost of ownership. Thus product development is an essential part in a product's life-cycle. But because of the frequent iterations in the design phase [1] and the increasing variety of products, the amount of time and money for the design of technical products is considerable.

Typically, engineers work with software systems that use a 3D CAD model as the main product model. Since complex technical products normally affect several domains, the geometric model has to be imported into different analysis tools such as finite element analysis or computational fluid dynamics, each of them using a proprietary problem model. Changes to the geometric model in consequence of the analysis results often have to be done manually, leading to another cycle in transferring the geometric model and performing the analyses.

Working with several different problem models makes it very difficult to get an integrated problem view where the dependencies between the different domains can be regarded together. This will only be possible if there is one central model that holds all the necessary information for an engineer to perform his task. To decrease the development time of a product, it is therefore essential to use appropriate software tool support and to adjust the numerous tasks in the design phase.

The ability to model and handle complex systems can be improved by using more abstract languages. The object oriented nature of the Unified Modeling Language (UML) [2] to model concrete objects and classes of objects, to reuse models and model elements and to identify patterns of best practice designs together with a graphical representation of the model are necessary aspects of modern tools in

software engineering. The advantages of these methods and tools are obvious and so the UML has been adapted to systems engineering leading to the Systems Modeling Language (SysML) [3]. As a modeling language the UML and SysML are used more and more in systems engineering [4].

In the last few years the next step towards a more abstract modeling language has been developed in form of the Model Driven Architecture [5] approach. The Model Driven Architecture aims at modeling a software system using a modeling language and generating the program code by performing several model transformations. The expected advantages are an acceleration of the software development process and the improvement in software code quality by minimizing the code errors.

In this paper an approach to engineering design is shown according to the Model Driven Architecture. This approach uses a central model of a product containing information for the connected analysis programs as well. By using model-to-model transformations the detail level of a product design is increased continuously while maintaining a consistent model. The resulting model contains all necessary information to generate code for the connected programs to perform the analyses. These transformations will be reused for further product development and can therefore lead to a high quality library reducing errors in the code for the succeeding programs.

A further acceleration of the development process for products can be reached, if the iterations between modifying the product model and performing the analyses of the modified product model will be automated. Therefore the analysis programs have to return the results back into the model, giving the possibility to restart the model refinement steps with a modified model. By automating this task sequence, it is possible to optimise the product design by maintaining a consistent model at the same time.

A first version of a software tool that uses a central model, model transformations and an automated export of the computed data to connected analysis programs has already been developed in our working group [6], [7]. The approach shown in this paper provides the basis for a modified version of this software that is based on the methods of the Model Driven Architecture.

2 MODEL DRIVEN ARCHITECTURE

The Model Driven Architecture is a promising approach by considering models as first class entities which is attracting wide attention not only because of initiatives of the Object Management Group in software engineering, but also due to the increasing use of SysML and UML in systems engineering. The UML has been adopted as standard in software engineering and is also attracting wide attention in systems engineering as can be seen from the standardisation of SysML.

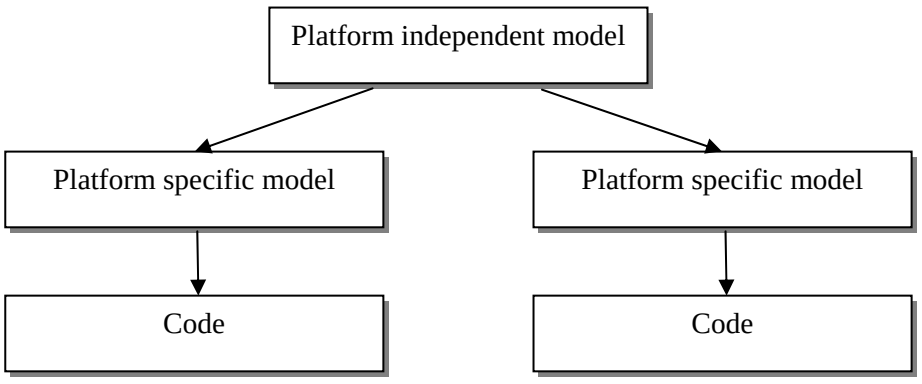

Figure 1. MDA transformations [8]

The MDA aims at developing precise and machine readable models to allow an automated generation of the software architecture on each abstraction level. Figure 1 shows the models used in MDA and the transformations that have to be performed in the MDA development process. The Platform

Independent Model (PIM) defines a model with a high level of abstraction that is independent of any implementation technology [8]. Typically, the UML is used for the PIM.

The PIM is then transformed into one or more Platform Specific Models (PSM) that are linked to a specific technological platform. As most systems today affect several technologies, a PSM for each specific technology platform is generated. The final step in the MDA development process is to generate code from each PSM.

One aim of MDA is to separate design from architecture. The PIM represents the conceptual design by realising the functional requirements whereas the architecture provides the infrastructure with the underlying specific technological platform. This separation allows the reuse of a given design in case of a change or update of the specific technological platform as well as the reuse of transformation and code templates that are used by the transformations to the next concrete level.

3 A MODEL DRIVEN APPROACH TO ENGINEERING DESIGN

According to the Model Driven Architecture approach in software engineering, a model driven approach to engineering design is presented in the following. The modeling in the examples shown has been done with the Eclipse UML2 [9] and Eclipse Modeling Framework (EMF) [10] projects. The transformations have been performed by using Java code and the ATLAS Transformation Language (ATL) [11].

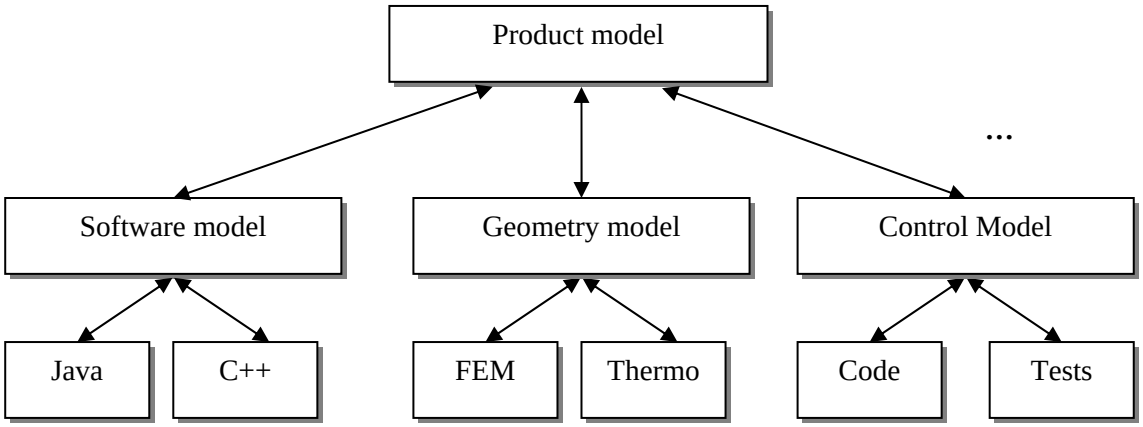

Figure 2. Model driven approach to engineering design

Starting from a central UML product model the detail level of the model is gradually increased by using model transformations. The code for the analysis programs is then generated by applying transformations to the detailed model. Alternatively the central model can be imported from a target system, e.g. the central model can be imported from the geometry model of a CAD system.

3.1 Central Model

The central model stores all data necessary for the product development as well as additional data that is needed by the succeeding analysis programs. As a modelling language for the central model the UML and SysML respectively are used.

Figure 3 shows the UML class diagram of a space station design [12]. The class diagram describes the structure of the space station and the relationships between the identified classes that contain the attributes and operations of the model elements. Additional information that is needed by the connected analysis programs is stored using UML stereotypes that are part of a specific UML profile. Thus, information affecting a specific analysis program can be encapsulated in such a profile. In the class diagram in figure 3 the stereotype <<catiaPart>> is applied to the classes in order to store information needed by CATIA to create the 3D CAD model.

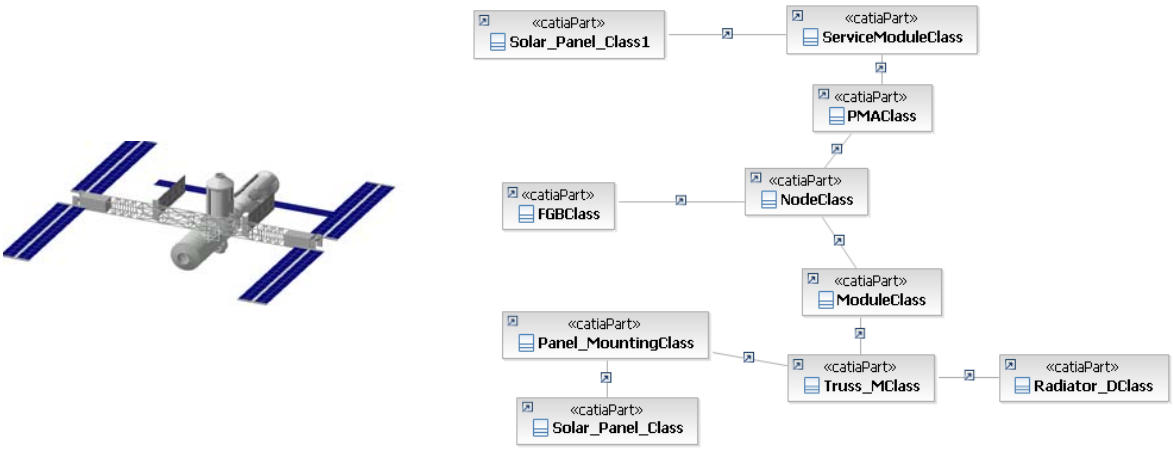

Figure 3. UML class diagram of a space station design

The class diagram as shown in figure 3 only describes the structure of a specific product. To show the complete view of a product, the UML object diagram is used as shown in figure 4. In the object diagram all attributes of the classes and the attributes of the applied stereotypes are set and therefore comply with an instance of a modelled product.

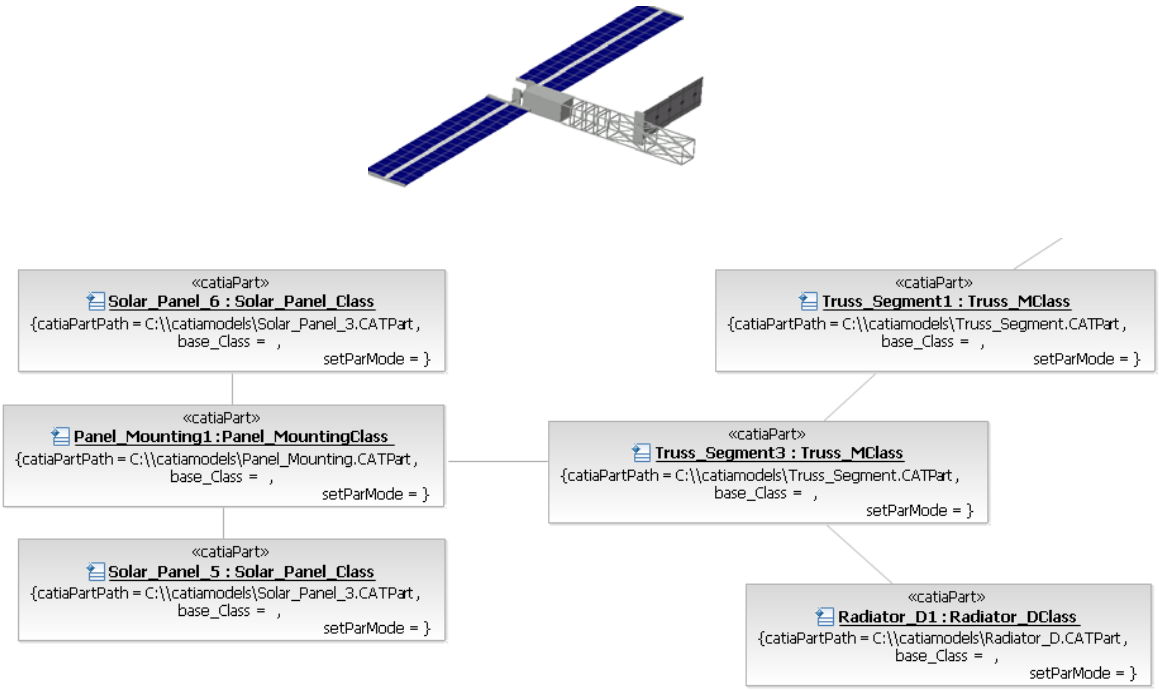

Figure 4. Part of a UML object diagram of a space station design

Using a central model has the advantage that all relevant product data are connected to each other. For this reason existing dependencies between data of different domains can be regarded together to obtain consistent product data by meeting the given constraints. In the UML constraints are expressed using the Object Constraint Language (OCL) [13].

The model of the space station in figure 3 and 4 has been modeled in CATIA and afterwards imported into a UML model using import functionality. With such an importing functionality the product designer can switch between different modeling levels when modifying the model having all necessary information in the central product model.

Figure 5 presents another example of a model exchange between the central model and the target system. It shows the result of the conversion of a reference control model [14] to a UML activity diagram. The example describes the movement of a ball bouncing on the floor. When running the simulation, the equations of movement of the ball are integrated to compute the velocity and the height of the ball. As shown in the example, UML activity diagrams allow a precise description of the dynamics of a system.

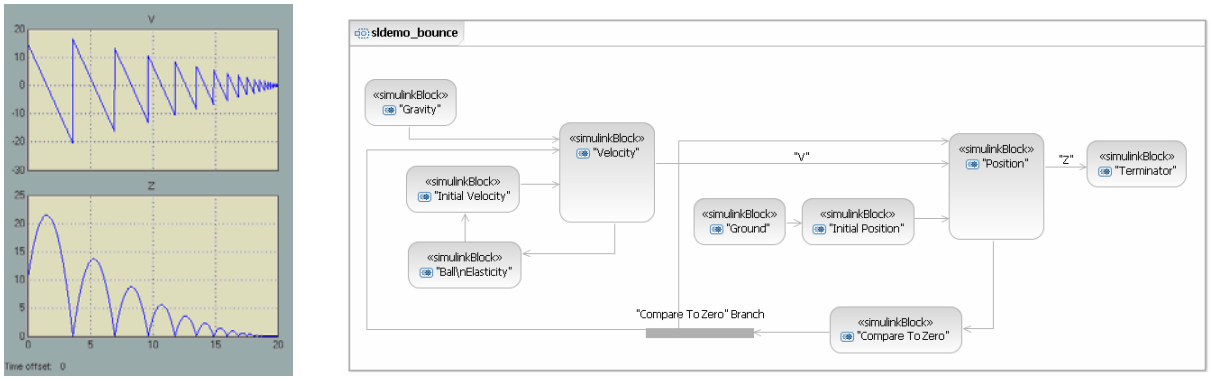

Figure 5. UML activity diagram depicting the dynamics of a bouncing ball

3.2 Model Refinement

In the design phase, models usually have to be modified and detailed over several iterations. The detail level of a product design can be continuously increased by performing model-to-model transformations, beginning with a start model. The sequence of transformation steps that have to be executed to get the resulting detailed model are determined with a UML activity diagram.

Figure 6 shows an example of a transformation that modifies a given space station design. The transformation consists of a condition part and an action part. When the transformation is applied, the tool is searching for the model elements that are given in the condition part of the transformation. If the model elements could be found, they are replaced by the given model elements in the action part of the transformation. To ensure that product model is consistent after performing a transformation, constraints expressed in the OCL can be added to the condition part and also to the action part of each transformation.

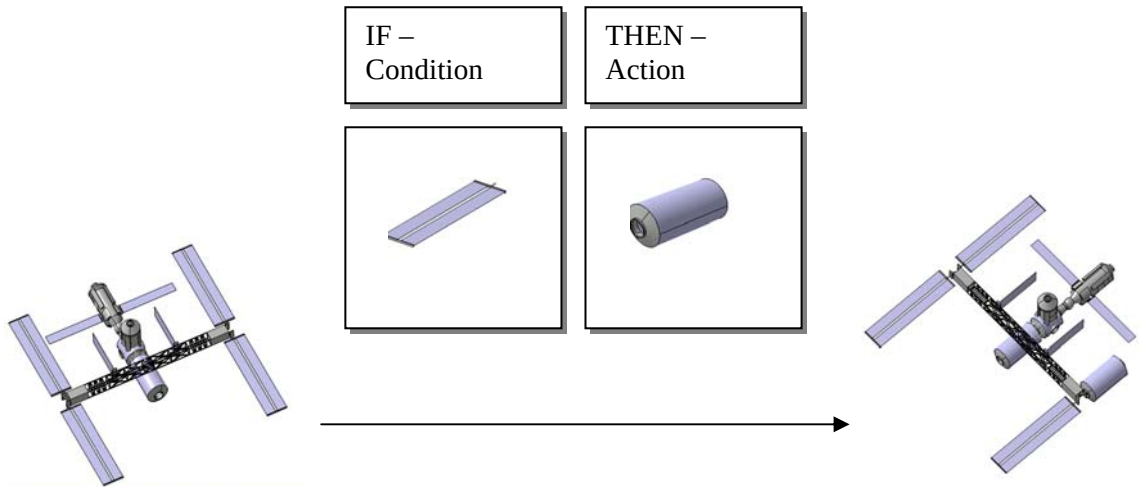

Figure 6. Modification of a space station design using a transformation

When using transformations one can start with an initial model and refine the model gradually during the design process. This allows documenting the decisions made during the design process explicitly, giving the possibility to connect knowledge based systems and therefore assist the engineer by providing expert knowledge. Furthermore, existing product models can be used to generate product variants or prototypes simply by modifying or extending the existing transformations or by modifying the transformation sequence. Due to the constraints attached to the transformations, a consistent model can be maintained.

If the conditions for applying a transformation depend on mathematical equations, these equations have to be solved within the model. Therefore the equations contained within the model are automatically exported to a mathematical solver. After solving these equations the solutions are re-imported into the model to update the demanded values.

In case that the computation of the equations or the variables has to be done by different mathematical solver or analysis programs, it is essential to determine the correct sequence of the equations to be solved. Figure 7 shows a solution path generator that is used to derive the solution path of the system design equations [15].

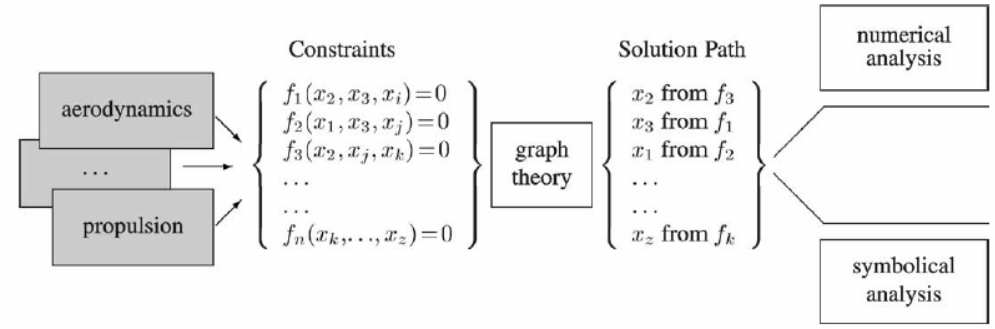

Figure 7. Solution path generator [15]

3.3 Generation of Target Models

Having performed all model-to-model transformations, the input data for the subsequent analysis programs is automatically generated using model-to-code transformations. Figure 8 shows a part of the ATL rule that transforms the engine hood model to CATIA code. The respective parameters and data for the analysis programs are taken from the central product model and then transformed using script templates or code libraries, producing code that can be executed by the specific analysis programs to create a target model. Consequently, the code templates for transforming the model data into program code for the analysis programs can be reused which can lead to a high quality code library.

```
query EngineHood2Catia =
  (thisModule.preString()
  +EngineHood!InnerPanel.allInstances()->asSequence()->first().toString()+'\n'
  +EngineHood!Point.allInstances()->collect(e|e.toString())->sum()+'\n'
  +EngineHood!Hole.allInstances()->collect(e|e.toString())->sum()
  +thisModule.postString())
  .writeTo(thisModule.getParameter('outputPath').debug('outputPath'));


helper context EngineHood!Hole def : toString() : String =
  let numbers : OrderedSet(Integer) =
    self.points->iterate(id;numbers :OrderedSet(Integer) = OrderedSet{0} | numbers->append(numbers->last()+1))->excluding(0) in
  '\tRedim pointNames('+numbers.size().toString()+')\n'
    +numbers->collect(i|'\tpointNames('+i.toString()+')="Ausschneidungspunkt'+self.points.at(i).id.toString()+'"\n')->sum()
  +'\tgenerateRoundedPolyline '+self.id.toString()+',pointNames,holeRadius\n'
  +'\tdoCutting "Umriss'+self.id.toString()+'","Support",depth,rimWidth\n\n';
```

Figure 8. Part of an ATL rule

The engine hood models shown in figure 9 have been generated from the same central UML model. The information for the analysis programs is stored in the central model by applying stereotypes to the model classes and model objects. The information stored in the model is passed to the model-to-code transformations and thus to the code templates in order to generate the particular code for the specific analysis systems.

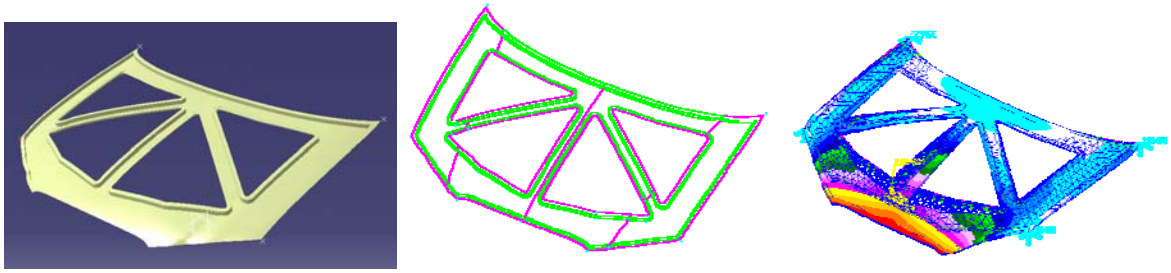

Figure 9. Engine hood model in CATIA, Patran and Nastran

After creating the models in the analysis programs and performing the desired analysis methods, the computed results can be led back into the model by modifying the model elements and model parameters. This can be done either by importing the modified target model into the central model or by modifying the elements of the central model directly.

When the central model has been modified, the task sequence of model transformations, generation of code and initiation of the computations done by the connected analysis programs can be performed again. It is evident that this approach simplifies the use of an optimizer, maintaining a consistent model while optimizing the design at the same time.

4 CONCLUSION

The model driven approach to engineering design presented in this paper aims at accelerating the product development process in the design phase by reusing models, transformations and code templates and improving their quality at the same time.

According to the Model Driven Architecture in software engineering, this approach uses a central model to store the information about the product together with additional data for the succeeding analysis programs. Using the UML and SysML as the modeling language for the central product model allows designing a product model on a high abstraction level and thus handle the complexity of technical products much better.

The transformations used throughout this model driven approach serve several purposes. Firstly, with model-to-model transformations initial models can be detailed gradually, allowing to check the given constraints for each transformation and to maintain a consistent model. Furthermore, the design decisions for each transformation can be documented providing expert knowledge when reusing these transformations for other designs.

Secondly, existing product models can be reused and modified easily by changing the transformation sequence or the transformations itself leading to product variants whereas the consistency of the model will be maintained. Thirdly, transformations are used to generate code for the subsequent analysis programs. The code templates for transforming the model data into program code can be reused and constantly improved which can lead to a high quality code library.

Due to the possibility to generate input data for the analysis programs and to import data into the central model, an automation of the task sequence in the design phase can be implemented. Controlled by an external optimiser, e.g. a genetic algorithm, the product model is modified on the basis of the results that have been computed by an analysis program. The optimiser then initiates the transformation steps and the computations done by the analysis programs. Finally, these new results lead to the next iteration by modifying the product model again until the desired target values have been reached.

REFERENCES

[1] Pahl G. and Beitz W. *Engineering Design*, 1996 (Springer).

[2] Object Management Group. UML 2.0 Infrastructure Specification. http://www.omg.org/technology/documents/formal/uml.htm, July 2005.

[3] Object Management Group. SysML Specification v. 1.0. http://sysml.org/specs.htm, May 2006.

[4] Lykins H., Friedenthal S. and Meilich A. Adapting UML for an Object Oriented Systems Engineering Method. *Proceedings of the 10th International INCOSE Symposium*, 2000.

[5] Object Management Group. Technical Guide to Model Driven Architecture: The MDA Guide v1.0.1. http://www.omg.org/mda/presentations.htm, June 2003.

[6] Rudolph S. Aufbau und Einsatz von Entwurfssprachen für den wissensbasierten Ingenieurentwurf. 3. Forum Knowledge Based-Engineering, CAT-PRO, Oktober 2003.

[7] Kormeier T., Schaefer J. and Rudolph S. System Design with Design Grammars. *Proceedings International Congress on FEM Technology*, November 2003.

[8] Kleppe A., Warmer J. and Bast W. MDA Explained. The Model Driven Architecture: Practice and Promise, 2003 (Addison-Wesley).

[9] Eclipse UML2. http://www.eclipse.org/uml2, 1/2007.

[10] Eclipse Modeling Framework. http://www.eclipse.org/emf, 1/2007.

[11] ATLAS Transformation Language. http://www.eclipse.org/m2m/atl, 1/2007.

[12] Irani M. Space station design rules. *SAE Aerospace Engineering*, October 2005.

[13] Object Management Group. UML OCL Specification. http://www.omg.org/technology/documents/formal/uml.htm, May 2006.

[14] The MathWorks. Bouncing Ball Model. http://www.mathworks.com/products/simulink/demos.html, 1/2007.

[15] Rudolph S. and Bölling M. Constraint-based conceptual design and automated sensitivity analysis for airship concept studies. *Aerospace Science and Technology*, 8 (2004), pp. 333-345.

Contact: Dr.-Ing. Peter Hertkorn
University of Stuttgart
Institute for Statics and Dynamics of Aerospace Structures
Pfaffenwaldring 27
70569 Stuttgart
Germany
Phone: +49-685-63657
Fax: +49-685-63706
E-Mail: hertkorn@isd.uni-stuttgart.de.