AUTOMATIC GENERATION OF MULTIBODY SIMULATIONS IN ANSA BY USAGE OF GRAPH-BASED DESIGN LANGUAGES

[1]C. Diez*
[1]Adam Opel AG, germany

KEYWORDS –
Multibody Simulation, Kinetics, Graph, Design Language

ABSTRACT –
Automation is nowadays a crucial part in engineering since automation of manual processes increases the speed by a very high factor and thus saves a lot of ressources. The challenge lies in the automation of creative high level tasks in which the structure of the problem itself is changing strongly and thus knowledge modelling is necessary.
One of these difficult tasks is the design of engineering structures. This task can be automated by graph-based knowledge libraries in which an engineer can lay down his knowledge to solve a task (6).
In the progress of construction many questions arise which can be answered by either tests or simulation. In general one uses a detailed geometry as basis and builds up a simulation model from it. In this process the engineer uses his existing knowledge to determine boundary conditions etc. This task can be automated by laying down the knowledge about this process into a design language which can cope with changing architectures.
In overall it is more appropriate to use a graph as meta-model from which other models like geometry and multibody simulation can be derived by a mere model-to-text transformation. Geometry is not an appropriate central data model since it is lacking a lot of information.
This paper concentrates on the automatic generation of multibody simulations by usage of graph-based design languages. Therefore a previous design language creates a graph with geometry information which will be extended to multibody physics (3). As a result this graph can be transformed to either ANSA (1) or ADAMS (7) automatically. In the end the design language itself returns result data for further analysis and possible decisions.

TECHNICAL PAPER -

1. INTRODUCTION

Graphs can be used as an abstract way of knowledge representation. For example the following multibody model in fig. 1 of a car roof kinematic can be shown as a chain of joints and bodys where each node contains further information about the object-class it represents.

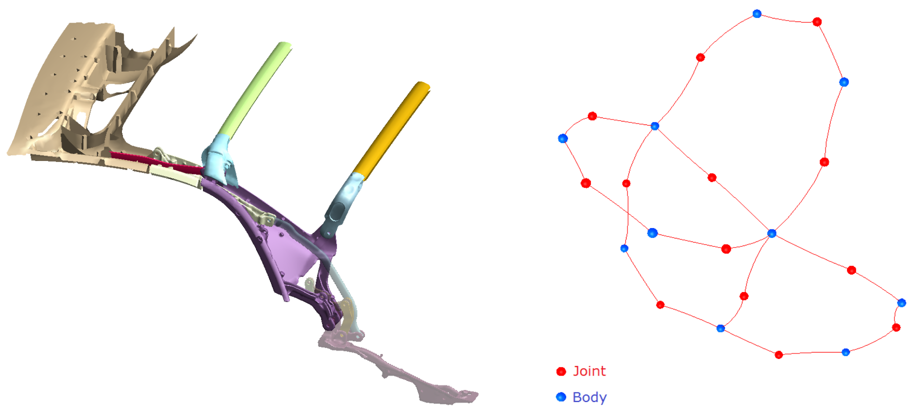

Figure 1 – Multibody model of a car roof (left) and it's graph representation (right).

Graph-based design languages have proven to be very powerful in the automization of engeering tasks (4). In a design language there are two major elements: a class diagram and a production system. In this paper the Design Compiler 43 (5) was used not only to build up the class diagram and the production system but also for execution of the production system and the export of the graph to external programs like ANSA.

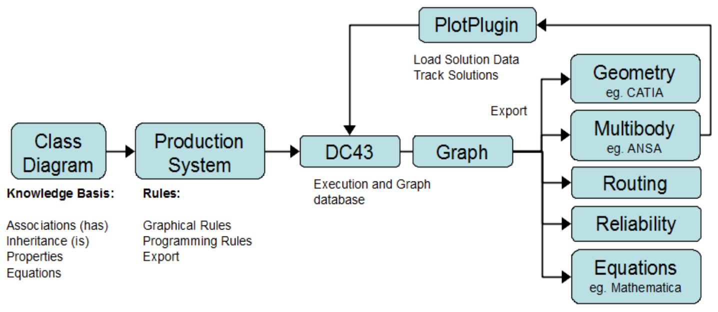

Figure 2 – Modeling process for design languages.

Class Diagrams

In a class diagram an engineer can lay down his knowledge structure as classes. These classes can be used later in a production system to implement the solution strategy. The classes can have properties and associations as well as inheritance inbetween. It also is possible to define equations between the classes themselves. The class diagram does not represent the model itself. It is solely some kind of data structure for the instances which will be created later. As already mentioned the most important relationship indicators are inheritance and association which can be spoken as ’is’ and ‘have’. Inheritance transfers all properties of a class on the inheriting one, including the classifier.
For example in a class diagram for a car one can define that a car is a vehicle and has four wheels. The diagram in Fig. 3 has no geometrical representation yet but as an engineer one is able to talk about for example wheels or their properties without specifying the geometry.

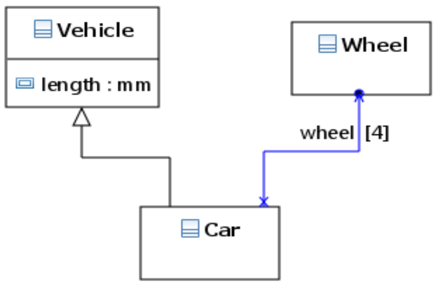

Figure 3 – Example of a class diagram for cars. A *Car* is a *Vehicle* thus also has a length. Additionally it has four *Wheels*.

Production Systems

The production system can be executed to build up the graph. Therefore one defines a build sequence of rules. This rules can be graphical or string-based. There also are more elements like decision nodes to choose different paths or define quality gates. The graphical rules work with a if-then scheme. If there is a given pattern in the graph then it will be modified like specified on the then-side. Most of the time one simply extends or replaces parts of the graph. Additionally there is the possibility to use external software by transforming the graph to other software systems to answer questions by simulation.

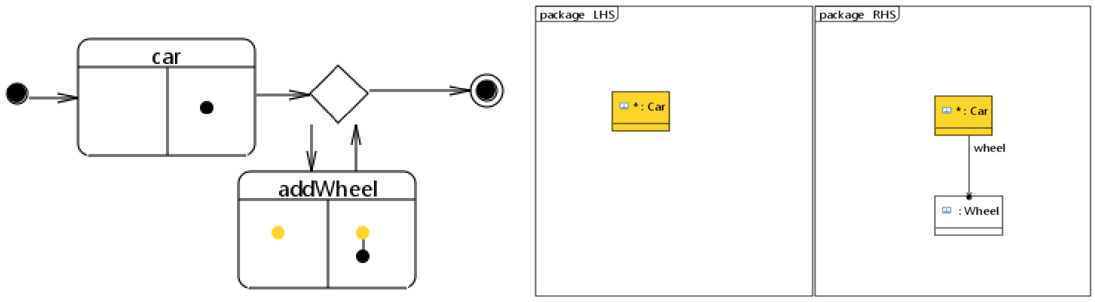

Figure 4 – Example of a production system (left) and a graphical rule (right). The graphical rule just adds an instance of the class wheel to any graph-node of the class-type 'Car' in the database.

2. CLASS DIAGRAM FOR MULTIBODY DYNAMICS

In order to generate the multibody simulation automatically one has to provide a class diagram for multibody simulation. When starting the transformation process an interface will run through the graph and check the instances for the multibody classes which then will be transformed into a text input for a software system. In the case of ANSA the design compiler creates a python script and launches the program with it. During the transformation many checks take place to ensure a mostly consistent multibody model. One has to emphasize the comfort that it is possible to implement semantic checks which ensure not only clean model data but also that the user makes less modelling mistakes. ????

The class diagram for multibody dynamics is split into several parts.

> Model-Creation
> Joints
> Loads
> Contacts
> Measurements

Each topic itself tries to be as small as possible since in knowledge engineering minimality is a hint for optimality of understanding.

<u>Model Creation</u>
The model creation part is about body creation which divides into body positioning and body properties.
First one has to create a body and give it a position. The positioning can be taken from already existing geometry, it can be done locally to other bodys or globally. One can define positioning equations which can be solved by the solution path generator before the geometry export.
After the positioning follows the assignment of rigid body properties. Therefore the user can assign a geometry to the body (stl,igs,...). If one wants to calculate the rigid body properties by density, one has to define geometry since otherwise the volume integration can not be performed.
At last one must assign inertia to the body. This can be done by either assigning mass and inertia explicitly to the model or using just a density which allows the calcuation of the rigid body properties if there is a geometry. Additionally there also might be markers on the rigid body for further positioning, contraints or boundary conditions.

<u>Joints</u>
Joints are used to couple rigid bodies. This can be done two type of joints: *KinematicJoints* and *DynamicJoints*. *KinematicJoints* are kinematic constraints which couple the degress of freedom in a specific location. *DynamicJoints* are softer and couple bodys with a classical spring-damper-equation.

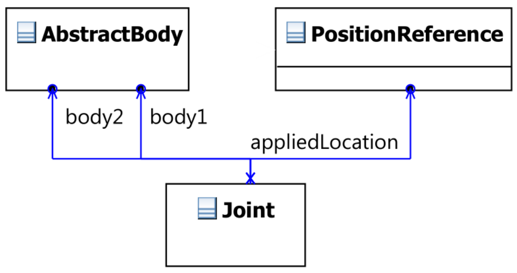

Figure 5 - Any *Joint* can be defined between two *AbstractBodys* (*Body*, *Ground*). Additionally one can specify a location for the *Joint* which may be any *PositionReference* (*Body*, *Ground* or *Marker*).

Most software systems provide many different types of joints like revolutes or sphericals. In this work another approach was used. One can specify a coordinate system (marker) where the joint shall be created. Then one is able to 'lock' specific degrees of freedom (DOF) of that marker. With this approach most of the joint types in commercial tools can be reduced to one class, the *GeneralJoint*. It is much easier just locking DOFs than rotating joints into their proper orientation.

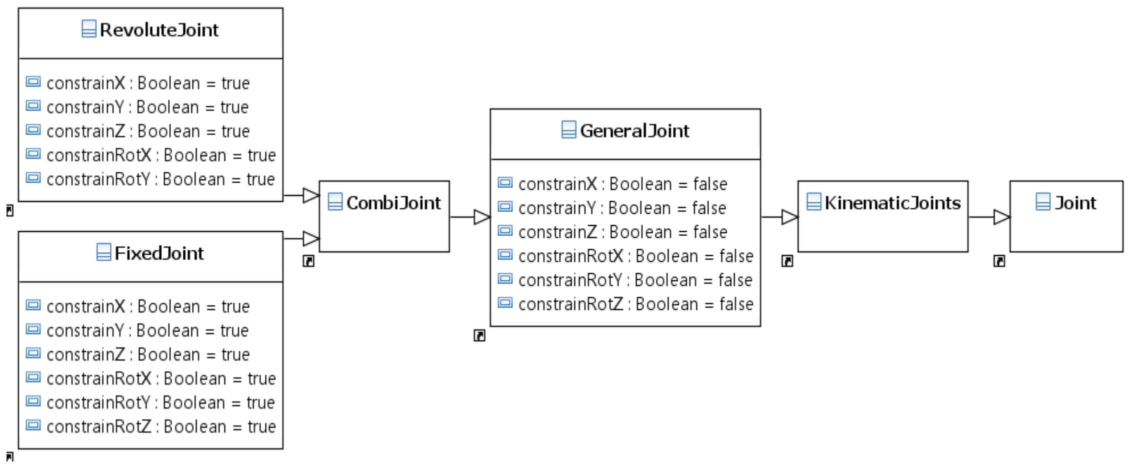
RevoluteJoint
constrainX : Boolean = true
constrainY : Boolean = true
constrainZ : Boolean = true
constrainRotX : Boolean = true
constrainRotY : Boolean = true
FixedJoint
constrainX : Boolean = true
constrainY : Boolean = true
constrainZ : Boolean = true
constrainRotX : Boolean = true
constrainRotY : Boolean = true
constrainRotZ : Boolean = true
CombiJoint
GeneralJoint
constrainX : Boolean = false
constrainY : Boolean = false
constrainZ : Boolean = false
constrainRotX : Boolean = false
constrainRotY : Boolean = false
constrainRotZ : Boolean = false
KinematicJoints
Joint

Figure 6 - Subset of the *Joint* classes. The *GeneralJoint* can constrain any DOF of a marker. A *RevoluteJoint* for example is a *GeneralJoint* but constrains 5 DOFs (as standard).

DynamicJoints are applicated rotationally or translationally. Kinematic constraints have an infinite stiffness which results in a hard couplings. It is more appropriate to use a finite stiffness and damping. These 'soft' couplings just correspond to spring-dampers. In the class diagram one can use either *TranslationalSpringDampers* or *RotationalSpringDampers*.

Loads

If one wants to solve the equation of motion for a body, one has to calculate the acceleration of the body from forces and momentum. These accelerations can be transformed into displacements by integration in time. Thus a load can be seen as on the one hand dynamic sources like forces but also on the other hand definition of the kinematics of a body. One should always keep in mind that by definition of the kinematics of a body, additional forces and momentum have no effect anymore.
Again in the classes there is a separation between a *KinematicLoad* and a *DynamicLoad*.

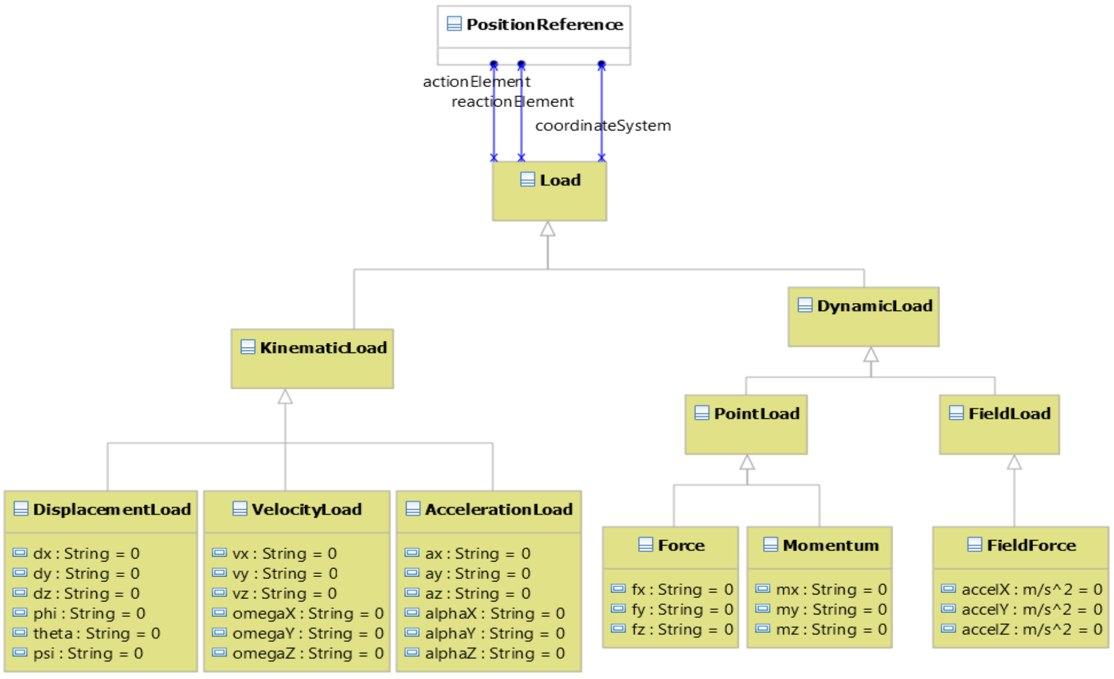

Figure 7 - A load has after Newton an *actionElement* and a *reactionElement*. One may not only define constrans but functions of time as well.

Contacts

It is possible that during a simulation contact between two rigid bodies occurs. In that case it is necessary to define contact properties between these rigid bodies. This can be done by the classes for an *ImpactContact* or a *RestitutionContact*. Besides the parameters for both classes, one simply has to draw a link from the instance to the two bodies which might collide.

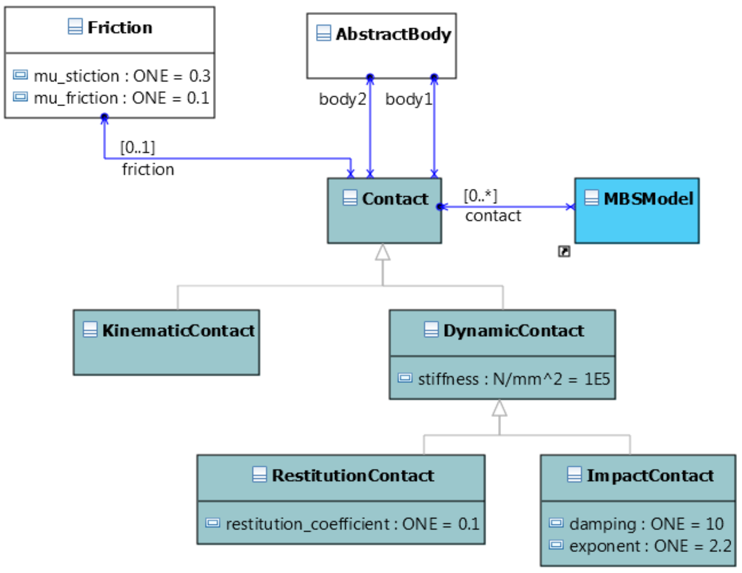

Figure 8 - A *Contact* needs two bodys and can have *Friction*.

Measurements

If one needs results from the simulation there also is the possibility to measure properties. Therefore again we seperate between the interest in kinematics or dynamics. Whereas kinematics tries to measure between bodys the dynamic measurements focus on the the force flow through the rigid body. Sensors can be attached to measurements to trigger specific events when they reach a specific value.

3. EXAMPLES FOR PRODUCTION SYSTEMS

The classes of section 2 can be referenced in the production system rules. One can either set up an example of it's own by geometry files or use the classes as extensions in another design language. First does not use the power of design languages that much but one still enjoys the help of a semantic check during the model to text transformation. Design languages show their full power when building up large systems automatically.

Car Roof

As a first example a car roof was taken. This model consists of only a few geometry files which were put together by a production system. This was simply to show that these classes do not need to be used in a design language but also can be handy for just assembling an multibody model from geometry.

The production systems contains of one programming rule which loads every geometry file as an own rigid body. The other rules are graphical ones and just connect the rigid bodys by their names.

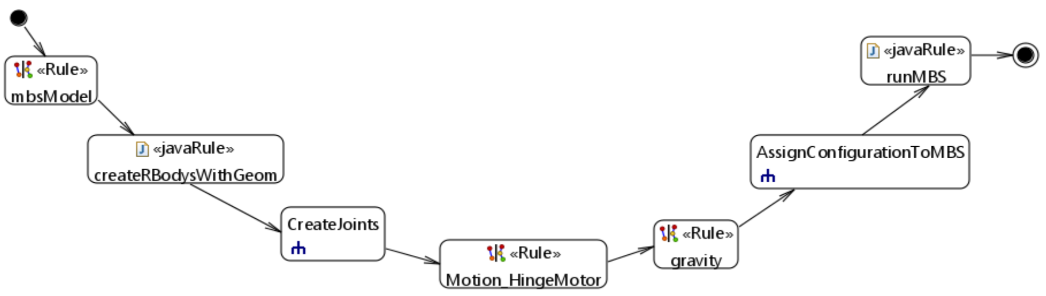

Figure 9 - The production system of the car roof contains not only rules but also sub-programs (rh) which themselves contain rules.

In the end of the production system the multibody interface is triggered in rule *runMBS*. This transforms the graph to ANSA. The car roof model may be checked and watched manually if the user does run ANSA with gui. If there is no interest in human interaction then one may also run ANSA in batch mode.

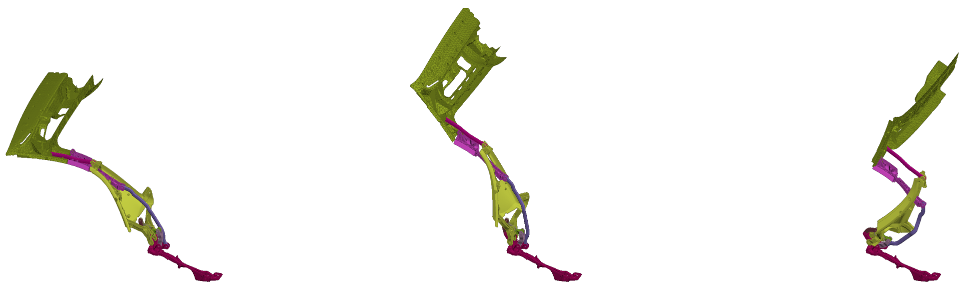

Figure 10 - Time series of the car roof simulation in ANSA.

Aircraft Landing Gear

This example represents an aircraft landing gear test after regulation CS-25. This regulation contains law specifications on aircrafts. One of these test cases is the landing of the aircraft. This test specification was taken to build up a simulation model. The knowledge about construction of landing gears was assembled and extracted into a design language in (3). This design language was extended in (4) by multibody physics. The modification in the class diagram was solely the attachement of a rigid body class to the class which represents physical parts. It can be spoken as: a part has a rigid body (model). Equations transfer the part position onto the rigid body. Additionally the *LeftWingLandingGear* (LWLG) also was extended by the class *MBSModel* to show that only the LWLG was used in the simulation.

The production system of the wing landing gear was extended by another sub program which contains the rules for the joints, load case boundary conditions and so on. One very comfortable feature for the setting up the loadcases is the nonlinear equation solver since position changes like an inclination of the wheels can be propagated very easy to geometry and rigid body model at once.

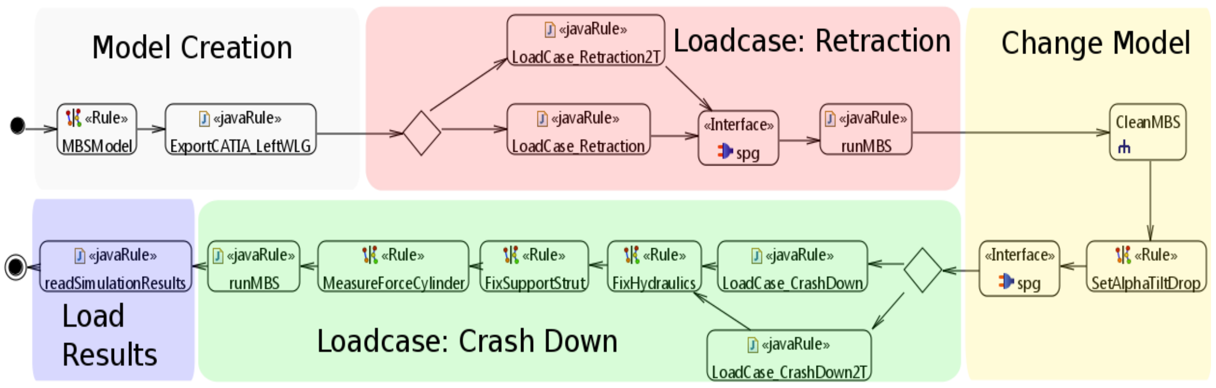

Figure 11 - The sub program for the extension to multibody dynamics contains two load cases: retraction and crash down.

The sub program in figure 11 is seperated into different groups. First the rigid body model is created by using the parametric CATIA geometry. Thereafter follows the load case of retraction in which a kinematic movement of retraction is given and the force of hydraulics is measured to perform this movement. After changes in geometry position the load case of a landing crash down is set up and calculated. In it the force of the damping cylinder is measured. Finally all these results are loaded back into the engine for further analysis.

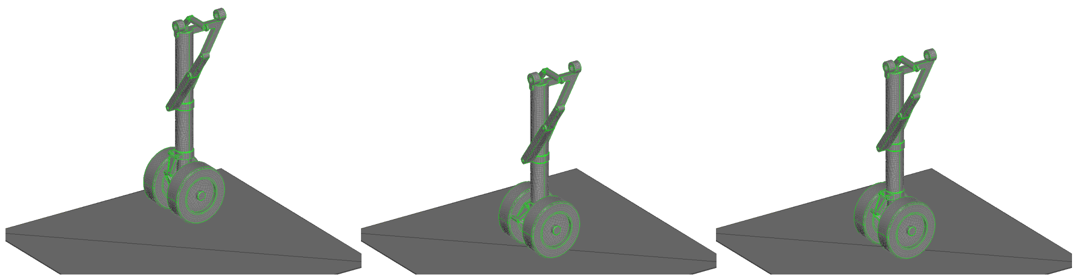

Figure 12 - Time series of the crash down load case. There is a virtual airplane mass on this landing gear according to the experiment specifications.

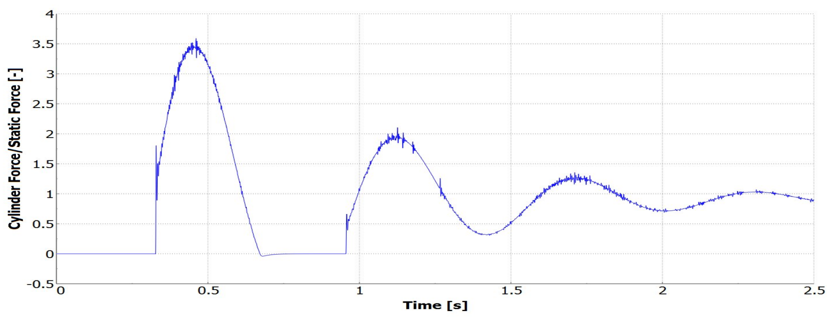

Figure 13 - Force over time from the simulation in the cylinder. The force curve is normalized with the static load on the landing gear and shows a peak of 3.5 times.

3. CONCLUSIONS

Design languages have shown to be quite powerful in automation of engineering design processes. The creation time of a design language depends heavily on the amount of already available design languages to solve the task (similar to software engineering). The advantage lies in the fact that every change in requirements can be propagated automatically thus every second run is free. As a result this method is recommended for reoccuring tasks. The most important necessity for shortening the creation process is to provide standard knowledge libraries for the different engineering domains like multibody physics. The class diagram explained in this paper can be used in other class diagrams and production systems to provide multibody modeling capabilities. An interface was written to translate the graph which contains instances of the multibody classes to an external multibody simulation software. In the case of ANSA the comfortable python interface ensures a stable translation with many checks taking place during runtime. These results of the simulation can be loaded back into the engine for further design decisions.

REFERENCES

(1) ANSA version 12.3.0, BETA CAE Systems S.A., April 2015
(2) A design language for passenger aircraft landing gears, D. Heim, 2014, Diploma Thesis, Insitute of Aircraft Design, University of Stuttgart.
(3) Creation of an ontology for design languages for automatic generation of multibody simulations, C. Diez, 2014, Master Thesis, Institute for Statics and Dynamics of Aerospace Structures, University of Stuttgart.
(4) Design languages for multi-disciplinary architectural synthesis and analysis of complex systems in the context of an aircraft cabin, S. Rudolph, 2014, CEAS Conference Toulouse 25.-27. september.
(5) Design Compiler 43, IILS GmbH, Februar 2015.
(6) Generating Simulation Models from UML - A FireSat Example, J. Gross S. Rudolph, 2012, Spring Simulation Multiconference, Orlando Florida.
(7) ADAMS Student Version, MSC Software, 2014.