Automatic Model Conversion to Modelica for Dymola-based Mechatronic Simulation

Tamás Juhász, M. Sc. and Ulrich Schmucker, Dr. Sc. techn.
Fraunhofer Institute for Factory Operation and Automation, Magdeburg, Germany
Tamas.Juhasz@iff.fraunhofer.de and Ulrich.Schmucker@iff.fraunhofer.de

Abstract

Virtual product development allows us to recognize and evaluate the characteristics of a new product on the basis of simulation at an early stage without having to build a physical model. Currently the most, widely spread commercial CAE systems do not offer direct support to external dynamic simulation applications. Conversely, dynamic simulation of a detailed model is required to maintain good correlation between the behaviour of the real product and its virtual counterpart. In this paper it will be presented, that using a partially automated workflow a convenient Modelica model translation can be achieved from the output of a mechanical CAD system, allowing the final model to be extended independently with new elements from other simulation domains, considering Dymola-based multi-domain simulation. A .NET-based integrated tool for mechatronic model editing and online / offline visualization support using advanced 3D (and stereo) techniques will also be emphasized in this article.

Keywords: Pro/Engineer, Mechatronics, Collision, Modelica, Dymola Simulation, Stereo, 3D Visualization

1 Introduction

Virtual engineering offers a completely new aspect of product development, as thereby all sections of the product life cycle can be independently analyzed and in parallel continuously optimized in the virtual world. Simulation makes the practical verification of the desired behaviour possible in early development stages.

It is very cumbersome to manually create a parametric simulation model representing a complex product that has been designed in a CAD system. Additionally it is often the case that in machine production a family of component parts with varying parameters has to be designed repeatedly. Nevertheless the product planning is usually an iterative practice: some internal model parameters must be fine-tuned, according to model assessment or verification processes. This implies that an automated model conversion is highly demanded in order to accomplish a good workflow. The designing engineer can inspect the behaviour of the given virtual product by utilizing a dynamic simulation of that. For a convenient iterative workflow a solution have to be provided to automate the conversion between the standard output format of the source CAD system and the input format of the target simulator.

In this article it will be presented that using *RobotMax*, our .NET-based mechatronic model authoring and visualization application mechanical CAD data from the widely-spread Pro/Engineer CAD environment can be imported, new mechatronic components can be added, thus multi-domain Modelica models can be generated and the results of the Dymola-based multi-domain simulation can be visualized in a convenient way, even in 3D stereo using various 3D technologies, for example by exploiting autostereo monitors, polarizer- or liquid crystal shutter glasses.

2 Translation from CAD data to Modelica models

We have interposed an own developed tool into the design workflow to achieve automated conversion to Modelica models from a Pro/E CAD model assembly (e.g. for mechanics: geometry, mass / inertia parameters, joints).

Similar work has been done in [1], but using the AutoCAD Mechanical Desktop system, and a different, shallower structure of Modelica models. Our approach allows a 3rd party to extend the mechanical model with additional elements from other engineering domains in such a way that a designer can still change / fine tune parameters in the CAD environment (and re-export the mechanical model), without sacrificing the extra work that another expert might have already done within the other model domains, where there might be connections to the previous mechanical model.

2.1 Basic steps of the translation process

There is a large amount of commercial and non-commercial applications (e.g.: "3D_Evolution" or "TransMagic") available on the market offering native conversion between common standard (STEP, IGES) and other well-known (AutoCAD, CATIA, Inventor, Pro/Engineer, SolidWorks, Unigraphics, etc.) CAD data formats. Thus without subsequent restrictions it is assumed our source data is available in the format that our CAD system (Pro/Engineer) is able to import.

The translation from a CAD source file to Modelica description needs the following basic steps:

- Assuming the CAD model has already been imported into *Pro/Engineer*, you can export the hierarchy and geometry information of the actual model to VRML files simply through a click over the File menu "Save as…" command. Note that in a general case you get a main hierarchy file and the geometries of the subsequent parts in separate .WRL files. Geometry information is essential if you want to model collision between the parts during the simulation (see section 3.1 for further information).
- *SimMechanics* is a single-domain extension of the Simulink environment developed by MathWorks, and can be used for modelling and simulation of mechanical systems. Under Pro/Engineer environment the freely available *Pro/E-to-SimMechanics* plug-in [2] lets you export the given CAD assembly to a single (so called "Physical Modeling XML") descriptor file, which is invented to ease the generation of SimMechanics models out of Pro/E data in an automated way. The result XML file contains global hierarchy-, constraint- and physical parameter information (inertia-tensors, masses, etc.), but no geometries at all.
- We developed an application (it is called *RobotMax*) the core logic of which processes the aforementioned XML descriptor file matching with VRML hierarchy/geometry files, thus generating an internal multibody model out of the CAD information. In *RobotMax* the internal (original) model can be extended interactively with various electromechanical elements (e.g.: with parametrical motors from a model library: see section 4.1) to form a more complex mechatronic (multi-domain) model. Finally, our tool is able to export its final mechatronic model to Modelica models using the built-in conversion module, and on demand by the same time it propagates the geometry to DXF format mesh files, in order to use those as visualizing shapes in Dymola environment.

2.2 Building a draft hierarchy out of XML information

A single "Physical Modelling XML" file enumerates all parts (= XML bodies) in the original root CAD assembly (= XML subsystem) from which it was created. The special *RootGround* part represents a fixed point in the environment. Each normal XML rigid body entry contains information about the physical parameters (mass, inertia, surface area, volume, etc.) of the given Pro/E part and has at least two coordinate frames in *World* space: the one that defines the location of the centre of gravity (*CG*) of the rigid body, the other that shows the origin transformation of the body's geometry (*CS1*). XML bodies can carry any number of additional frames (*CS2, CS3* …), which all have a unique integer ID: these unique numbers are used by us to find the corresponding parts between joints.

As it was mentioned before, the XML file also contains information about joints, which represent the constraints of the original CAD assembly. Each XML joint (J_i) has two integer IDs that are uniquely referencing two different frames (these are named "*Base*" and "*Follower*" in a SimMechanics model).

The special *weld* joints are used to mount two rigid parts together with no degrees of freedom left between those. There can also be a series of primitive joints between two frames, representing various degrees of rotational / translational freedom between those parts. In the hierarchy this always implies the following sequence: "*Follower*" → "J_1" → … → "J_N" → "*Base*", where "a → b" shows that "a" is the child of "b" in the hierarchy (i.e. it inherits all transformation from that). Using the XML Joints' frame references you could build a skeleton (a draft hierarchy) of XML bodies. Unfortunately this does not imply automatically that the final hierarchical model is also ready: the geometries of the possibly colliding (but point-sized so far) bodies are still missing at this point.

In CAD systems it is quite often the case that more parts in an assembly share the same name (you can imagine a "template" part that has been used many times as a building element). On the contrary, in case of the target language Modelica, the variable names must be unique inside each model. Via translating the mechanical CAD information, a single, pure mechanic Modelica model has to be generated first. This initial model contains only the parametrical bodies and the mechanical joints, which are connected by "connect" Modelica clauses. All exported bodies and joints must have an individual, unique instance name.

As the auto-generation of VRML and XML files are independently done, the partially auto-generated names inside the result files (e.g.: "*Obj01*", "*Obj02*" vs. "*Obj*", "*Obj-1*") will neither be globally unique nor match each other. In order to find the corresponding entities both in VRML and in XML domains, you have to follow a sophisticated procedure. This is in the most cases inevitable, because there are usually less XML bodies than actual VRML geometries. You must know which geometries form together a single rigid body, if you want to have a consistent collision handling during the simulation.

2.3 Matching hierarchies in XML and VRML domains

All VRML geometry nodes have a homogenous transformation matrix (which can arise derived from their respective parents, recursively), from which you can derive their global pose (position and orientation) in the 3D world. This derived 4x4 matrix is also used to transform the local vertices of a given VRML shape into the global (*World*) coordinate system during rendering, for example. Fortunately the same pose information is also included in XML file with *CS1* frame of the XML bodies.

First you have to search for matching the position of all *CS1* frames (extracted from XML) with an origin frame location from VRML geometries being just imported. If there are more bodies having the same CS1 frame position, you continue filtering the "candidates" by differences in *CS1* frame orientation. Assuming there are still more than one parts with the same global pose in *CS1* (which is blissfully a rare case), you can compare the names of the XML bodies and VRML shapes (namely just their prefixes: e.g.: "*Obj01*" will match with "*Obj*" or "*Obj-1*") to find the highly demanded single positive match. It is hardly imaginable that there are more parts in the CAD assembly with exactly the same pose and name. This should indicate that there is an error in the source CAD plan.

It is often the case, that there are subsequent levels in the VRML geometry hierarchy: in this case these child shapes are to be merged into the same higher level geometry.

After assigning the VRML geometry to the corresponding XML bodies, the final, pure mechanic multibody model can be finally generated. For this sake, the necessary physical parameters (masses, locations of *CG* frames, inertia tensors) have to be substituted into the final Modelica actors' parameters.

3 Expanding the standard Modelica library

The Modelica Standard Library is a standardized and free package that is developed together with the Modelica language by the international Modelica Association [8]. The Mechanics Multibody Library (MML) is a package in the main library providing 3-dimensional mechanical components to model mechanical systems in a convenient way.

The MML does not include support for rigid body collision handling. Handling contacts between mechanical objects can be very important in many disciplines of mechatronic simulation (e.g.: robot manipulating tasks).

3.1 Collision Handling

We extended the MML library with support for collision handling using a spring and damper material model, suggested by the article [3], but based on more robust Bullet collision library in our recent implementation. We discussed the details of our implementation in [4]. In this section it will be presented what sort of new Modelica components have been developed for this purpose.

The basic *World* model in MML represents a global coordinate system fixed in 3D space origin. The behaviour of the basic World model has been extended via inheritance: from the original base model a *Collision Manager* (CM) subclass has been inherited that is responsible for collision handling in simulation of multibody systems.

The standard Modelica implementation of rigid bodies (see *BodyShape* component in MML) needed also to be extended to handle collision (via communication with the CM). Our *Actor* class encapsulates the physical kinematic- (pose, velocity and acceleration), dynamic- (mass, location of centre of gravity and inertia tensor) and material- (stiffness and restitution) parameters (also initial values of those) of a rigid body. Note that actors don't have any geometry information.

The *Shape* class extends the *Modelica.Mechanics.MultiBody.Visualizers.Advanced.Shape* class, offering 3D visualization possibilities in Modelica environment. Each *Shape* instance must connect to a single actor with a respective 4x4 transform of local origin of the geometry. These objects represent the geometry of the rigid body they connect to. The *Collider* class is the subclass of Shape, which can serve the collision geometry of that part. In order to ease the export to Modelica, these classifications of new shape classes make handling of geometry orbicular from both the aspect of Modelica and *RobotMax*, our CAD translator application.

For online, real time visualization support (see section 5.3) each *Shape* instance has a 7 component pose vector (3D position + a quaternion orientation) simply assigned by their local origin frame's pose. There is a pre- allocated ***P*** pose matrix (dimensions: 7 by N) reserved for the N shape objects, stored in a shared memory. The shared memory is implemented in a C++ class, is compiled to a DLL and it offers C interfaces to Modelica. The columns of ***P*** are updated every simulation step by the respective shapes' poses using the external C function invocation *setPose( )* from the Shape instances' Modelica source.

The singleton *Collision Manager* instance stores information about the positions, orientations, angular- and linear velocities of all *Colliders* existing in the global collision set. At the initial phase of the simulation each Collider instance reports the CM its geometry, which cannot change during the simulation. The CM updates the external collision forces on each colliding shape in each simulation step. These shape instances propagate the external collision force through their connectors to the respective actor instances.

Unfortunately the Modelica language specification being used at the development time (it was version 2.2.1) did not allow having a collection containing polymorphic references to the instances of a user defined class (i.e. abstract models) themselves: our Modelica arrays can contain only basic data types. This introduces a little performance loss: the CM has to store duplicated information in separate arrays about the positions, orientations, angular- and linear velocities of all shapes existing in the global collision set.

Some shapes can be *individually* excluded from collision handling via disabling their collision flags (for example in draft motion tests). On the other hand, sometimes it is desired (usually for simplified models) to allow *also pairs* of bodies to constantly interpenetrate each other during the simulation, without any internal tension or force between them. For this purpose the user of the extended library can assign a matrix to the CM containing the IDs of unwanted collider pairs.

There is a permanent bidirectional communication between the colliders and the collision manager. The external collision response forces and -torques that are calculated and responded by the CM, act together on the given actors automatically as it was told before. This is due to the behaviour of bidirectional Modelica "*connect*" equations.

The Modelica standard has a well-designed interface to external software modules [5] (e.g.: Fortan or ANSI C: sometimes allowing more powerful algorithm implementation). Accordingly, we were not confined to implement the whole collision manager class in pure Modelica. For the algorithmic core functionality of collision detection and -response calculation the C interface could be used:

For each supported collider shape type a C++ class had to be implemented, having parameters similar to their Modelica counterparts. These classes are instantiated at the initial phase of simulation: as soon as a Modelica *Collider* is initiated, the corresponding C++ constructor is invoked from Modelica code, through our C interface wrapper.

In each simulation step the C++ part of the CM updates the pose of all C++ shapes via their Modelica counterparts' pose, and invokes the main method to query the actual collision forces and -torques for all active geometry in the scene. In the background the free Bullet library [7] is being used to query collision information among our rigid bodies (these are being treated as independent ones, no joint-constraints are introduced here). The penetration checking functionality of Bullet is done the following way:

For each pair of shape types, a certain collision algorithm is assigned, by using an internal dispatcher. The collision detection library part of Bullet can retrieve contact points between any triangular geometry types (for some concave-convex case the primitive geometry types – such as sphere or cylinder – need to be tessellated to triangles). The used algorithms are a modified version of the GJK algorithm [6] with the EPA - Expanding Polythope Algorithm for convex-convex cases, and GIMPACT for the cases involving concave geometry.

Our pair-wise collision response calculation method (spring and damper technique: dependent on penetration velocity, relative motion of colliding parts, material stiffness- and restitution parameters) is discussed in [4]. A single invoke on the external C++ library can solve the collision response for the whole system at once, thus the external forces on the Modelica colliders can be updated in each simulation step.

3.2 Abstract joint models allowing domain independency

Our purpose is to simulate articulated multibody mechatronic systems having multiple rigid bodies connected by joints. The original test CAD models, which we seized to test our conversion process, have either no motor information, or this information is not accessible from the outside (i.e. cannot be exported from) the CAD system. This implies that in *RobotMax* all XML joints will be converted first to abstract ones by default (prismatic, revolute and spherical joints, or serial combination of those are supported in the entire system). Note that spherical joints (allowing 3 rotational degrees of freedom in their coupling centre point) are always passive: they cannot be actuated in the original manner.

We implemented abstract joint models in Modelica for prismatic and revolute joints, which are exactly the pure mechanical constraints, representing the allowed single degree of translational or rotational freedom between their 3D frame connectors. These abstract joint models have a one dimensional *'Drive'* flange, as you can see on Figure 1:

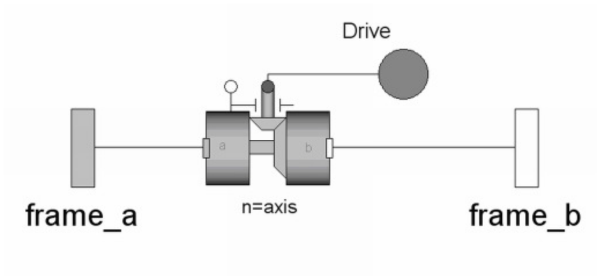


Figure 1: Abstract model of a revolute joint

If the *'Drive'* flange connector is not connected from the outside in the container Modelica model, an abstract joint will be equivalent to an ideal, free joint. On the contrary, connecting a motor's drive part to the drive flange of these joints makes actuated joints in the final mechatronic model. For the details please refer to section 4.2.

Using this abstraction we could decouple the pure mechanical model from other electromechanical components: these can be exported to a separate top-level Modelica model.

4 Adding electromechanical components to the internal model in RobotMax

Our goal is to support the simulation of the dynamic behaviour of the product being designed in the source CAD system. Assuming you have a CAD model of an industrial robot having a few joints that should be actuated by motors, you could easily ask what kind of motors should be applied in order to achieve a pre-defined speed along the desired path of the tool centre point, or to stay below the maximal allowed positioning error.

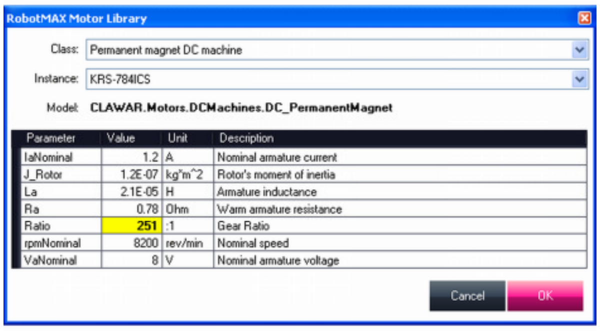
Unfortunately, we can't seize so far any description of the possibly occurring electromechanical components from the Pro/E CAD system, which we could embed automatically into the final mechatronic model at the end of the conversion process.

You can say that the requirement of having motors in an articulated multibody system is more than desirable. Without such elements you could not simulate / verify the active dynamic behaviour of a moving virtual structure.

4.1 The Motor Library in RobotMax

We developed an XML-based extensible Object Library that can contain parametric components of any modelling domain. The special modelling domain of electromechanical components (motors) will be emphasized in this section. For example the Motor Database inside the Object Library contains motor classes (e.g.: DC motors or induction machines) as entries.

Every class in the library has an absolute path reference to the Modelica implementation of the model represented by it. These classes enumerate their parameters, which all must have a unique name (referring to their respective variables in the Modelica model). Each parameter must also have a type (Float, Integer, String, etc.) and a Boolean flag indicating whether its actual value is editable by the user. For example changing the gear ratio parameter in a final motor instance is still allowed. A general parameter can also have a physical unit (like 'Ohm' or 'kg·m^2': one should use SI standard units, unless it is not specified here differently), minimum / maximum limits and a descriptive comment optionally.



Class: Permanent magnet DC machine

Instance: KRS-784ICS

Model: CLAWAR.Motors.DCMachines.DC_PermanentMagnet

Parameter	Value	Unit	Description
IaNominal	1.2	A	Nominal armature current
J_Rotor	1.2E-07	kg*m^2	Rotor's moment of inertia
La	2.1E-05	H	Armature inductance
Ra	0.78	Ohm	Warm armature resistance
Ratio	251	:1	Gear Ratio
rpmNominal	8200	rev/min	Nominal speed
VaNominal	8	V	Nominal armature voltage

Figure 2: An example entry in our motor library

Figure 2 shows a screenshot of our library's browser dialog displaying the parameters of our DC permanent magnet motor class. The user can also edit here the highlighted gear ratio, before it will be inserted to the internal model in *RobotMax*.

The instances of a class are enumerated in the library after each class declaration, defining the actual / initial values for each parameter in all occurring instances. The instances must have a global unique ID (a primary key along that column), which is always required in a relational database (e.g.: during searching).

4.2 Our actuated joint models in Modelica

For the most mechatronic simulation purposes one has to set continuous reference values of the active joints in the system (defining position, velocity or acceleration parameters of those) in order to make the parts follow a pre-defined trajectory. A well-designed controller should be introduced that minimizes the error between the actual and the reference values of each joint in every single moment.

We implemented 1-1 parametrical, translation- and rotation based drive model in Modelica (for prismatic- and revolute joints, respectively), which contain a separated control- and actuator part, and is decoupled from the given joints' mechanical part. Figure 3 shows our general model for an actuated joint:

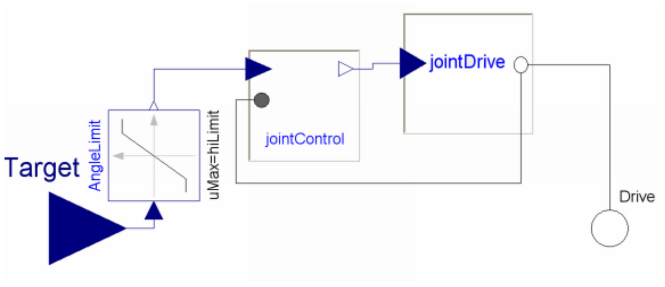


Figure 3: The schema of our joint drive subsystem

The general *"jointDrive"* actuator model has a replaceable motor and gear component. If a component is declared *replaceable* in Modelica it means that one can transparently exchange the implementation of this part with another model, unless the given external connector interfaces are kept intact.

As it was told before, the pure mechanical model was exported to a separate Modelica text file. As long as the names of the joint entries are not changing, we will find the way to connect the respective '*Drive*' connectors in both models. Thus the user can experiment with fine-tuning the motor parameters and simulate the new model without the need to redo the CAD / XML conversion process from the beginning again.

5 Simulation and visualization

The workflow presented so far had been finally extended with a motion planning task, which can be carried out right before the Modelica export step, in order to define a continuous-time function in a convenient way for each joint's path.

5.1 Defining motions and simulating the model

RobotMax – our .NET-based CAD translator / environment editor application – offers keyframe-based motion planning and has built-in support for inverse kinematics that was used in the following example to model a palette manipulating motion with an industrial robot model. The user can also fine-tune the motion by interactively adjusting the values of the selected servos.

The following image sequence shows the three basic steps of the CAD to SIM process (in Pro/E → RobotMax → Dymola order), presented so far:

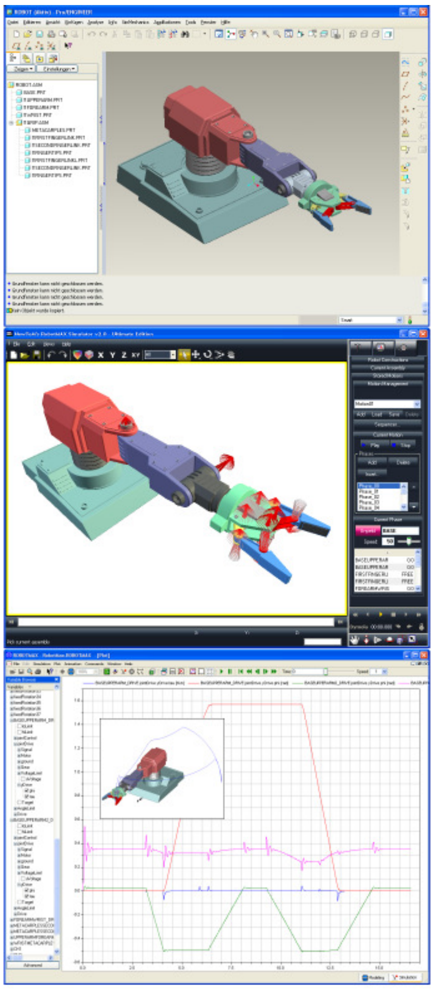


Figure 4: The Dymola simulation of an industrial robot-arm designed in Pro/Engineer and converted by RobotMax using the presented workflow

5.2 Creating test scenarios in RobotMax

The *Scene Editor* in *RobotMax* can be used to create various static / dynamic scenarios, allowing testing the interaction between the actual virtual product (that is being designed in CAD) and its environment, which is usually modelled separately or is sometimes simply neglected. For example a bumpy road can be added to the 3D world in case of testing a new car suspension assembly. VRML geometry can be imported, or the user can create new static / dynamic objects from primitives with the interactive tools in *RobotMax*. The parameters (materials, dimensions, positions, etc.) are interactively changeable (in case better precision is needed, can be set also manually) and the modifications can be undone, thus allowing an iterative approach of testing with various scenarios. Multiple viewports and various alignment tools are helping you to make the test setup as precise (and as informative) as possible.

5.3 Problem of online visualization

The Dymola simulator [9] being used in our project has a 3D viewer (animation) support for multibody models containing 3D geometry, but has a limited functionality and is not user friendly enough.

If you want to visualize the simulation results from a 3rd party application while Dymola is running in the background, the poses of the various geometries have to be gathered and transmitted to the viewer application online. Although Dymola stores the output of a simulation in a file (in Matlab® format), this file is exclusively locked: thus no other application can read from that file until the simulation finishes. Another solution had to be found to access pose information during the simulation.

5.4 Visualization in RobotMax

Our idea to transfer data to a viewer was to query it from the shared memory containing the ***P*** matrix of actual shape-poses (see section 3.1). On a viewer side there is usually no need to update the pose of the objects after each solver step (e.g.: a step size of 1 ms would lead to 1000 frames/second required refresh rate). The problem can be turned around: you can retrieve (poll) the actual pose of any shape from the shared memory at a desired, smaller frequency (e.g.: 50 Hz).

The *RobotMax* has all functionalities a modern 3D viewer application requires, with multiple orthogonal or perspective viewports, interactive camera setup and built-in support for advanced 3D visualization techniques – including real 3D methods such as autostereo (for monitors with lenticular lens layer), time-interleaved (for liquid crystal shutter goggles), spectral-interleaved (for red-cyan anaglyph spectacles) or dual output (for two projectors and polarizer glasses) – representing the actual internal model in 3D. For a broader overview of these techniques please refer to the article [10].

If the user switches *RobotMax* to online visualization mode, it polls the pose information for each shape continuously and updates the viewports with the preset frequency only.

In order to be able to inspect the simulation results multiple times, there is a support for offline visualization, of course. A simulation output file can be parsed by an external application only after it has been completely written and released by Dymola. In offline visualization mode RobotMax invokes Dymola with the generated Modelica models (according to the process described in Section 2) and waits for the lock of the result file to be released.

The sequences of samples of each simulation signal are stored in this file, including input / output variables, state variables and their derivatives. In *RobotMax* after parsing all the exported signals into memory, the signals belonging to the world transformation matrices are used to setup a keyframe animation, which can be sought and played back from a desired position at the desired speed.

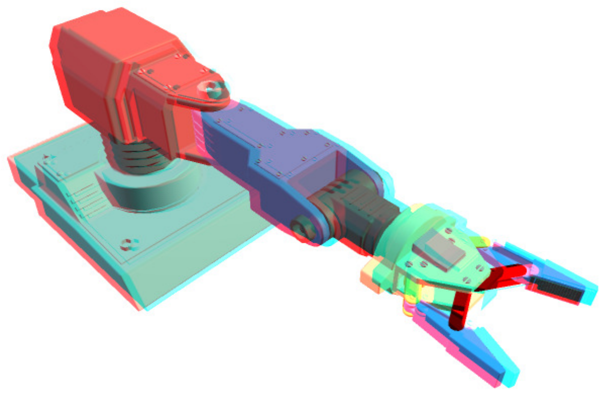


Figure 5: anaglyph mode 3D visualization, screenshot

On Figure 5 a screenshot can be seen that was taken in *RobotMax* showing the red-cyan spectral-separated anaglyph stereo image of the previous industrial robot-arm example at initial pose at the very beginning of the simulation.

6 Conclusion and future work

A highly automated, convenient conversion workflow from Pro/Engineer CAD data to multi-domain Modelica simulation models has been presented in this paper. The relevant online / offline visualization methods – with advanced 3D techniques within the same integrated tool used for model translation – were also discussed here. For a schematic overview of the presented workflow see Figure 6.

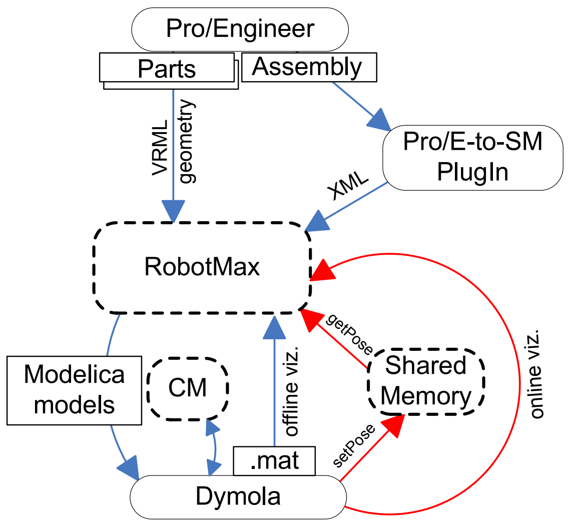


Figure 6: schematic process overview

In our future implementation we will use the Pro/ToolKit API to directly access data of other, newly added components in Pro/Engineer WildFire 3, such as springs, dampers and motors, and translate these elements also into the final Modelica models. Using this interface the functionality of the Pro/E-to-SimMechanics plug-in can be completely replaced later by our implementation.

There is a free Modelica library – called BondLib, available at [11] – for bond-graph represented analog electronic circuits, including a full implementation of Spice models. The presented CAD conversion process can be generalized to introduce complex models of electrics / electronics domain, to be converted to Modelica. We will investigate the possibilities to introduce *OrCAD* layout plans and *P-Spice* models into our virtual mechatronic workflow.

In the next version of our *RobotMax* tool we will implement a 2D Plot functionality to be able to inspect simulation signals as 2D curves in a given viewport. Our collision response calculation in tangential space (which is currently very simplified) has to be improved to achieve more realistic friction forces and –torques between contacting bodies.

References

[1] Engelson, V.; Bunus, P.; Popescu, L.; Fritzson, P.: “Mechanical CAD with Multibody Dynamic Analysis Based on Modelica Simulation”; In Proceedings of the 44th Scandinavian Conference on Simulation and Modeling (SIMS-2003), September 18-19, 2003, Västerås, Sweden

[2] MathWorks: SimMechanics Translators: http://www.mathworks.com/products/simmechanics/description5.html

[3] Otter, M.; Elmqvist, H.; Díaz López, J.: ”Collision Handling for the Modelica MultiBody Library”; In Proceedings of the 4th International Modelica Conference, March 7-8, 2005, Hamburg, pp. 45-53

[4] Juhasz, T.; Konyev, M.; Rusin, V.; Schmucker, U.: ”Contact Processing in the Simulation of CLAWAR”; In Proceedings of 10th CLAWAR International Conference, 16-18 July 2007, Singapore, pp. 583-590.

[5] Fritzson, P.: “Principles of Object-Oriented Modeling and Simulation with Modelica 2.1”, Wiley Press 2004, ISBN 0-471-471631, pp. 311-322.

[6] Gilbert, E. G.; Johnson, D. W.; Keerthi, S. S.: ”A Fast Procedure for Computing the Distance between Complex Objects in Three-Dimensional Space”; In IEEE Trans. Robotics and Automation 4 (Vol2), April 1988, pp. 193-203.

[7] Bullet 3D Collision Detection Library: http://www.bulletphysics.com/Bullet

[8] Modelica Association – http://www.modelica.org

[9] Dynasim AB: Dymola 6 – http://www.dynasim.com/index.htm

[10] Juhasz, T.; Vajta, L.: “The Role of 3D Simulation in the Advanced Robotic Design, Test and Control”, International Journal of Advanced Robotic Systems – Cutting Edge Robotics 2005, ISBN 3-86611-038-3; pp. 47-61.

[11] Cellier, F.: BondLib – Modelica library: http://www.modelica.org/libraries/BondLib