Curs 10

Interactive Simulation and Visualization

17.05. 2018

Interactive simulation and visualization

Most researchers who perform data analysis and visualization do so only after everything else is finished, which often means that they don't discover errors invalidating the results of their simulation until post-processing. A better approach would be to improve the integration of simulation and visualization into the entire process so that they can make adjustments along the way. This approach, called computational steering, is the capacity to control all aspects of the computational science pipeline. Recently, several tools and environments for computational steering have begun to emerge. These tools range from those that modify an application's performance characteristics (either by automated means or by user interaction) to those that modify the underlying computational application. A refined problem-solving environment should facilitate everything from algorithm development to application steering. The authors discuss some tools that provide a mechanism to integrate modeling, simulation, data analysis and visualization.

There are many other kinds of information that are important in software design, which are not covered in this work. Particular examples are documents and document structures, entity-relation diagrams, scenario diagrams, database visualization and diagrams of program execution paths.
Here we discuss and compare several stages where interactive graphical environments can be used:

- model editing (in particular, browsing and editing of model component diagrams during the design stage),
- input data editing,
- visualization during execution (e.g. interactive control of simulations, execution monitoring, and computational steering)
- output visualization in the form of 2D graphs and 3D interactive animations.

User Interaction

Users are expected to be active agents in the process of design and use of scientific software. Therefore user interfaces play a dominant role.

In descriptions of interactive tools we attempt to evaluate the quality of user interfaces. For this purpose typical user tasks are selected and user efforts required for performing these tasks are estimated. A good interface for visualization and editing should provide the following

- consistent and compact presentation of components (layout quality),
- ease of use in navigation and location of relevant subcomponents (navigation quality),
- consistent feedback from user actions, such as data editing or simulation steering (feedback quality).

Taxonomy of Visualizations

In this taxonomy seven kinds of visualizations are related to seven corresponding data types. Generation of some of visualizations in this taxonomy can be automated.

1-dimensional: This class of visualization and data structures includes textual documents and program source code. However, the models we consider are usually quite complex. Therefore, one-dimensional visualizations are not sufficient for presentation of such models. In particular, program source code has a well-defined structure. We suggest that this source code is converted into hierarchical structures with nodes (classes, objects, instances) and connections (relations between them). This structure is generated automatically from the textual representation and is visualized by our tools. For instance, *ObjectMath* models are represented by hierarchical diagrams.

2-dimensional: This kind of visualization includes geographical maps and plans, as well as 2-dimensional plots of functions (X-Y plots). This is a widespread way to present results from scientific computing applications. Most of the tools for scientific computing, e.g. *Dymola, Mathematica, MathModelica, Beast* have facilities for plotting variables in different ways and options for choosing variables to be plotted. The graphical user interface for selecting variables to be plotted is automatically generated from data structures, e.g. for *Dymola, MathModelica* and *Beast*.

3-dimensional: Items with volume, e.g. real world objects, and object designed using *3D CAD* modeling tools are most naturally shown using 3-dimensional visualization. Both real and abstract objects (such as three dimensional plots of functions) can be viewed in this kind of visualization. Visualizations in three dimensions are often used for physics-based simulations. In this case, the structure of objects and their interrelations usually correspond to the structure of the mathematical model.

… 3-dimensional:

Such a model contains descriptions of each physical component (e.g. rigid body) which is visualized as a separate graphical object in *3D*. Therefore, creation of such visualizations can be automated (see paper 5). There are relations between the visualized components (contacts and various motion constraints), which usually are not visualized explicitly as graphic elements, but which can easily be noticed when moving objects are observed.

There exist, however, 3-dimensional visualizations where the structure of graphical objects is completely different from the structure of mathematical model. For instance, this happens in scientific visualization tools used in computation fluid dynamics. Instead, the structure of graphical user interface for visualization control often corresponds to the structure and dimensionality of data.

Temporal: Visualization of time lines, historical information, and events are examples of temporal visualization. In our case, visualization of temporal data is just one feature for other visualization types, in particular 3-dimensional visualization. We use time in order to represent changes in objects and their relations during physics-based simulation. Output data of such simulations contains values of various variables at each time instant. Animation is used for presentation of such simulations.

Multi-dimensional: Tools working with objects with many attributes need multidimensional visualization. Such objects become points inn-dimensional space. Visualization tools map objects with these attributes to a 2- or 3-dimensional representation. In this thesis, work in this direction has been done with parametric functions of many parameters, which were defined in *Mathematica*. In our tool the way of mapping n-dimensional parametric functions to space coordinates and time can be selected interactively.

Tree: Tree-structured visualization is useful for structures with relations between parent and child nodes. A tree is a convenient way to represent data structures of a model. Furthermore, it is possible to automate creation of interactive visualizations based on data structures. This automation has been designed for *C++*, *ObjectMath* and *MathModelica*.

Networks and general graphs: Arbitrarily linked relations between nodes are conveniently visualized by networks and general graphs. This visualization is used where objects are related by connectors, for instance in *Modelica* and *Dymola*. In mechanical models joints and other contact elements are used as relations between rigid bodies. These relations can be generated using a graphical user interface. For instance, relations between bodies are specified using a *CAD* interface.

Three dimensional graphical user interfaces

Development of modern technologies has made it possible to apply three-dimensional visualization in many application areas. In particular *3D* is used for visualization of computation results and for modeling real world objects, such as objects constructed using *CAD* tools. Due to development of graphic hardware *3D* animation recently became widely available for the users working on average computers and therefore *4D* data (three space dimensions and one time dimension) can be used for visualization.

3D visualization in scientific computing falls into three categories:

1. *Visualization of numerical results*
2. *Visualization of fixed shapes*
3. *Volume visualization*

1. *Visualization of numerical results*, where displayed shapes depend on a particular computation. These shapes might depend on some specific parameters, e.g. time.

   We assume that points in 3D are denoted as (x,y,z). The set of displayed points in three-dimensional coordinate space can be expressed as

   $$\{(F_x(u,v), F_y(u,v), F_z(u,v))\},$$

   where $umin < u < umax$ and $vmin < v < vmax$.

   Some components of a graphical user interface for such visualization can be generated automatically.

2. *Visualization of fixed shapes.* Each shape corresponds to a real world object which is modeled as a rigid body (e.g. by a *CAD* tool). Movement of the body is constrained by the laws of physics. This kind of visualization is also called physics-based visualization.

   Visualization of results from physics-based simulations can be automated.

3. *Volume visualization,* where displayed shapes are isosurfaces computed from some volume data. This data can be the result of some other computations or measurements. The set of points displayed can be expressed as

$$\{ (x,y,z) \;/\; F(x,y,z) = 0 \}.$$

   Rendering such visualizations is more difficult since it is hard to find the set of points and translate to 3D graphic primitives.

Interactive Visualization of Numerical Results of Computations Specified in Mathematica

Usually parametric surfaces are used for visualization of computational results when many inputs and outputs are involved in a computation. If two input and one output variables are used, the visualization of such a function is a surface composed by all points

$$\{ (x,y,F(x,y)) \} ,$$

where $xmin < x < xmax,\ ymin < y < ymax.$

If three input and three output variables are used, a dynamically changing surface can be composed from all the points

$$\{ (F_x(u,v,t),\ F_y(u,v,t),\ F_z(u,v,t)) \},$$

where $umin < u < umax,\ vmin < v < vmax,\ tmin < t < tmax.$

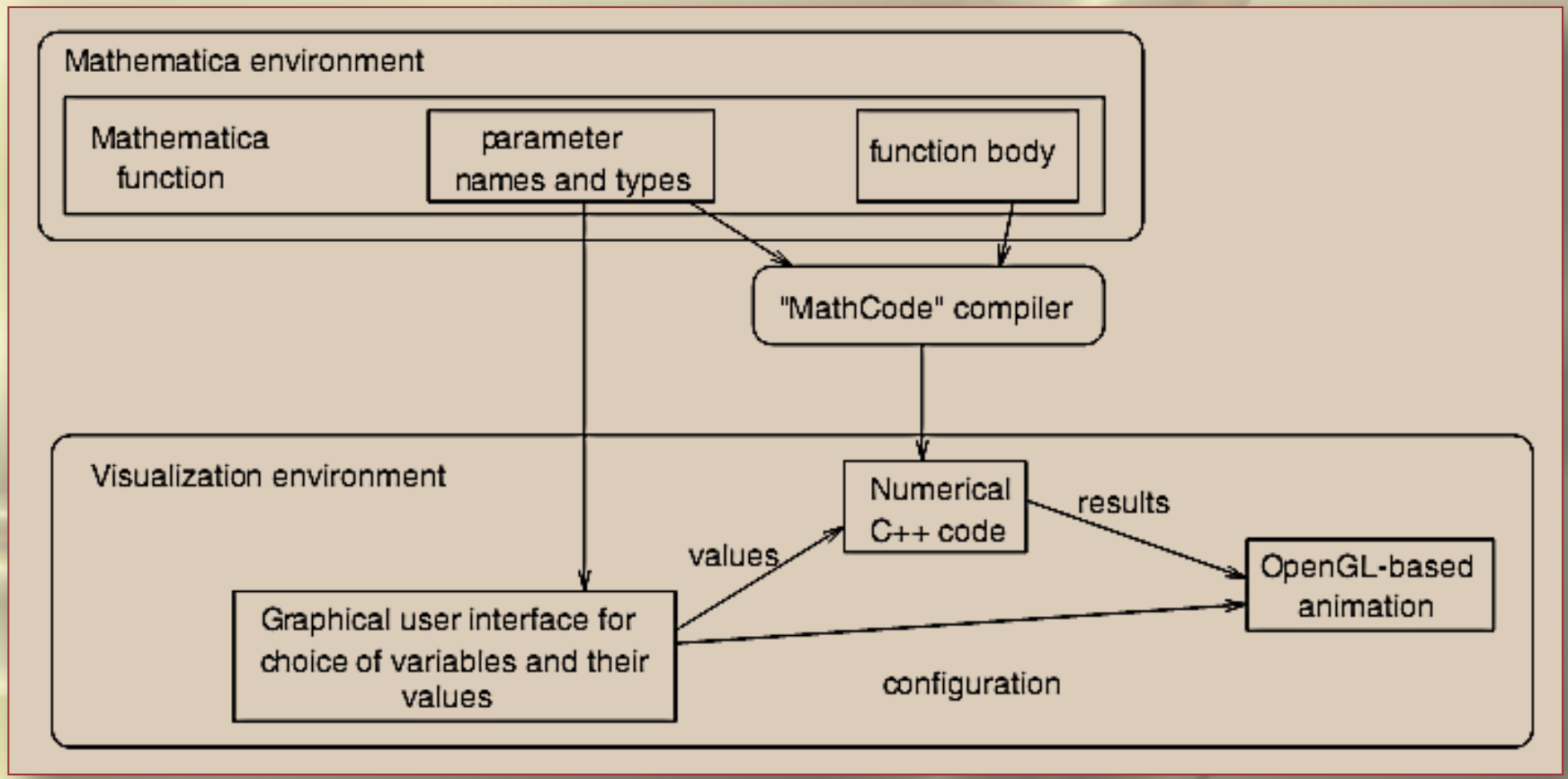
In general, an arbitrary function

$F : R^m \rightarrow R^n$ can be visualized using this method.

The limitations are:

- Input values for a number of dimensions (m-3 dimensions) should be fixed.
- Three dimensions are chosen from n, whereas the other n-3 dimensions are omitted.

Generation of visualization for functions with multiple arguments and multiple output values defined in *Mathematica*:



A multilanguage environment for interactive simulation and development controls for power electronics

Virtual prototyping of complex systems presents interesting challenges, especially with regard to systems in which different languages are used to represent different parts. We describe here one approach to solving such a problem. In particular, we describe how to integrate the virtual test bed (VTB) solver engine with the Simulink solver. This produces a rich environment for virtual prototyping of power electronic applications due to the inherent mixture of circuital and control problems. The integration is conducted within the context of the resistive companion approach. We present here the theoretical foundation of the approach, and also suggest the generality and extensibility to other solver engines such as SPICE. The theory is then enriched with some examples that illustrate the approach including the use of the VTB high-level graphic user interface

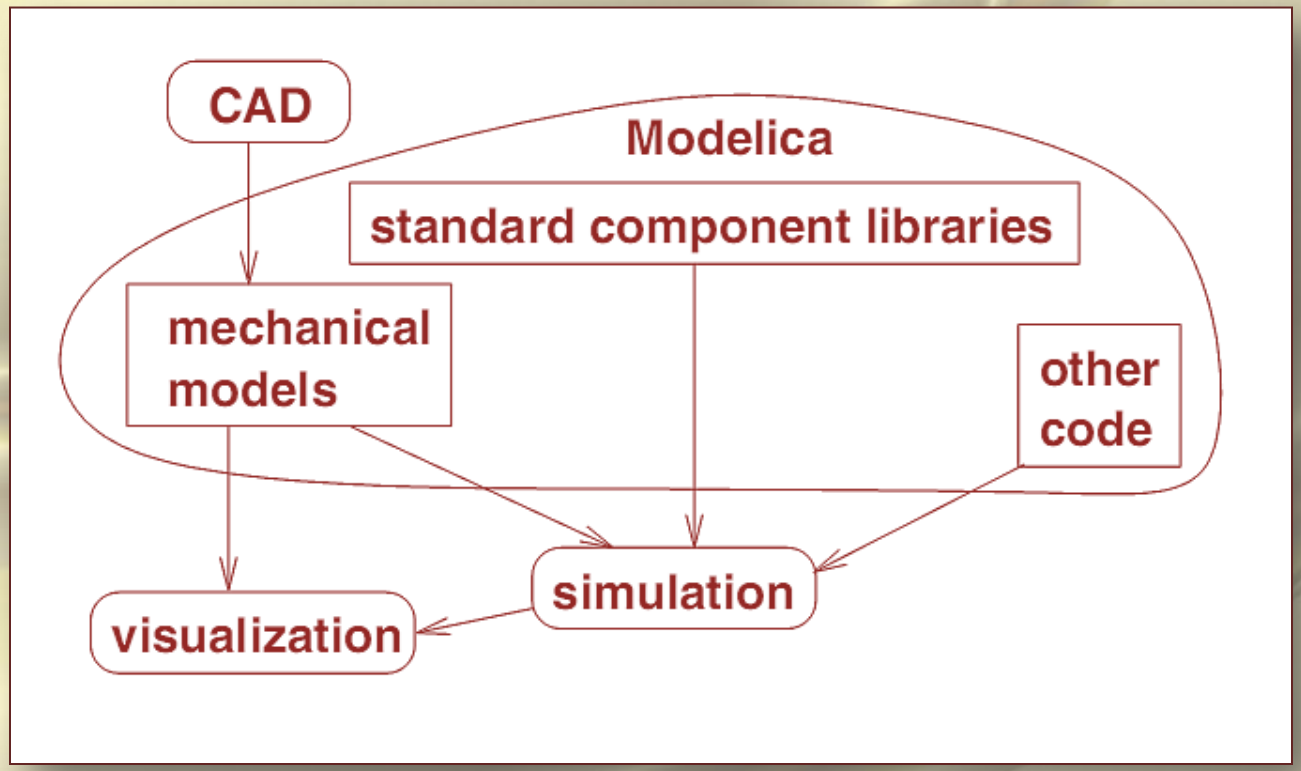
An Environment for Design, Simulation and Interactive Visualization for CAD Models in Modelica

The use of computer simulation in industry and construction is rapidly increasing. Simulation is typically used to optimize product properties and to educe product development cost and time to market. Whereas in the past it was considered sufficient to simulate subsystems separately, the current rend is to simulate increasingly complex physical systems composed of subsystems from multiple domains such as mechanical, electric, hydraulic, thermodynamic, and control system components.

Modelica is a language for dynamic simulation. In particular, mechanisms (such as construction tools, robots, vehicles) and constructs under dynamic load (such as hanging bridges) have been modeled, and interactively simulated. Thermodynamics applications are modeled too; in particular, models for indoor climate and energy simulations were developed in *NMF* which is similar to *Modelica*. This application inputs a building map designed in a *CAD* tool and generates equations for indoor climate simulations.

Structure of the integrated environment

Modelica is a standard notation which is used for standard domain libraries and for applications that use these libraries. Tools and environments are built to comply with this standard. The structure of the environment that leads the user from interactive design to interactive visualization:



Translation and simulation

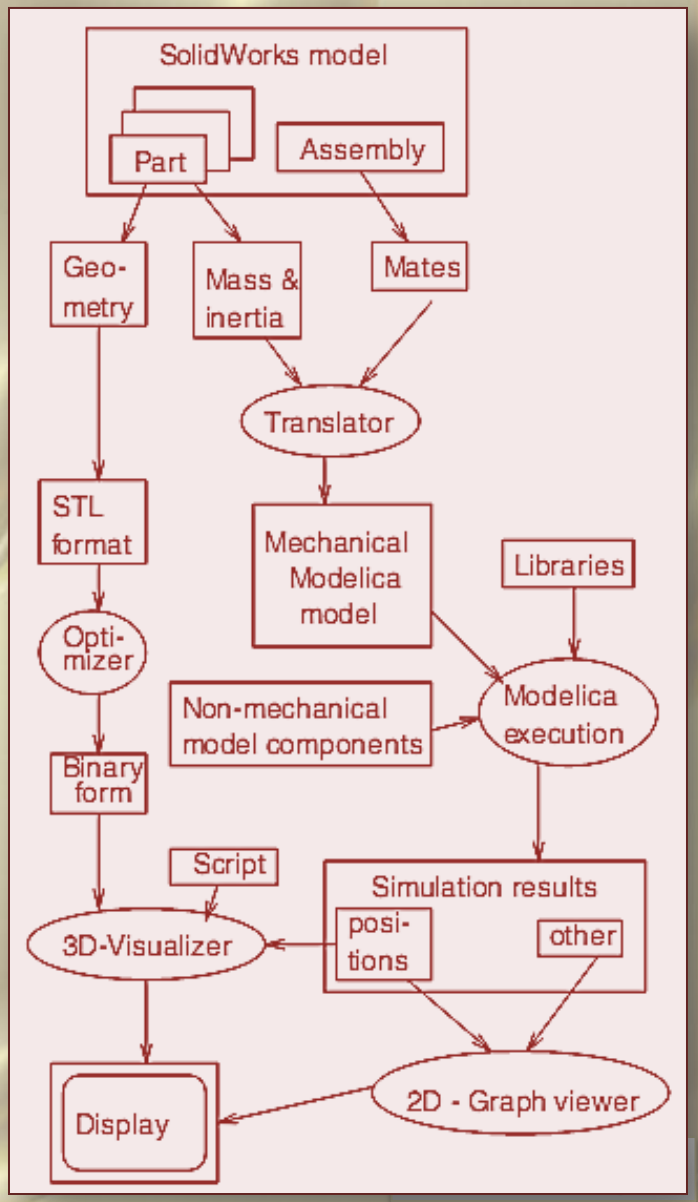
The components of the environment needed for visualization. Our translator from *SolidWorks* to *Modelica* takes information about the mates and produces a corresponding set of *Modelica* class instances with connections between them. The mass and inertia tensors for each part are computed by *SolidWorks*. These are extracted and used in a *Modelica* model. Geometry information is saved in a separate *STL* file for each part. All external forces that are applied to the bodies, as well as motor forces that are applied to revolute and prismatic joints should be specified. This is done outside the *SolidWorks* model by adding code for new class instances to the *Modelica* model. A control subsystem that controls the forces according to a certain plan (mission) can be written in *Modelica* . If necessary, external code in C can be added to the model. When a *Modelica* model is simulated, the position, orientation, velocity and acceleration for each part (*Body* instance) is computed. For *Modelica* simulation we use the *Dymola* tool with *Modelica* support.

Visualization

The integrated environment includes a visualizer that provides online dynamic display of the assembly (during simulation) or offline (based on saved state information for each time step).

The STL format is a very simple format suitable for visualization. All surfaces are divided into triangles and the coordinates of the triangle vertices, as well as normal vectors of the triangles, are listed in the STL-file.

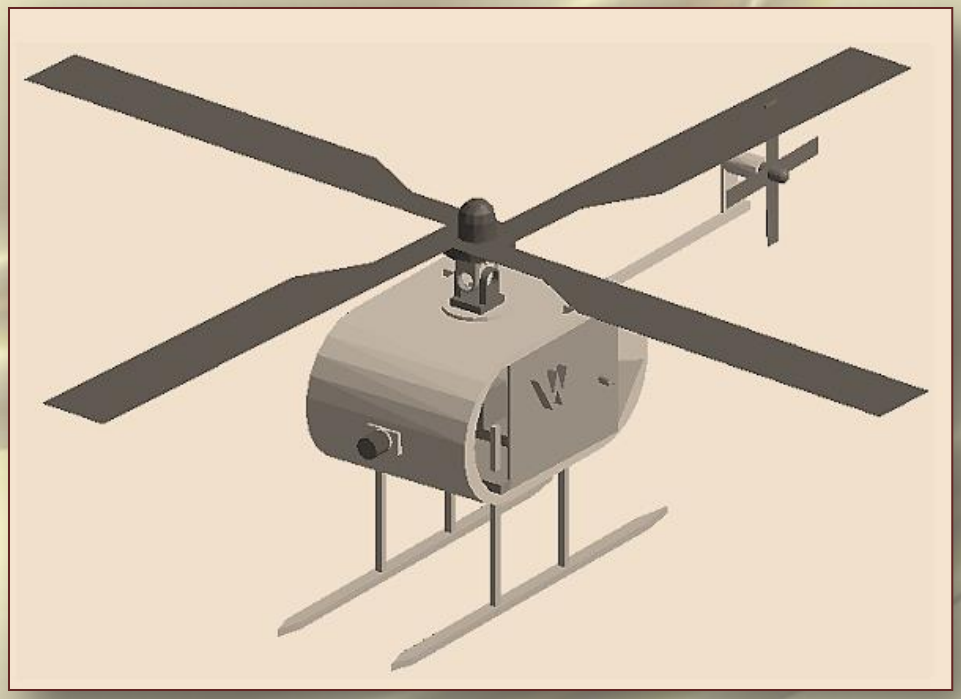
The path from *SolidWorks* model to dynamic system visualization:



… Visualization

The helicopter is designed in *SolidWorks* and consists of 10 parts, 4 revolute and 2 prismatic joints.
It is possible to export the visualization to *3D Studio MAX* (to create and save movies) an *MultiGen* on *SGI* (to design *Virtual Reality* applications).

Helicopter dynamic visualization:



Simulation and Visualization of Autonomous Helicopter and Service Robots

To decrease the costs and the time it takes to develop and test new products, computer simulations are very helpful. Models can be simulated, and their behavior can be examined. This applies not only to hardware, but even to software products that can be divided to several components, so that their cooperative work is simulated in a virtual environment. Some components of this environment can later be replaced by physical, real world devices.

Some other components can be just prototypes, and they are replaced later with more complex and realistic software components. In any case the idea is to construct a model and simulate both software and hardware before the actual production starts.

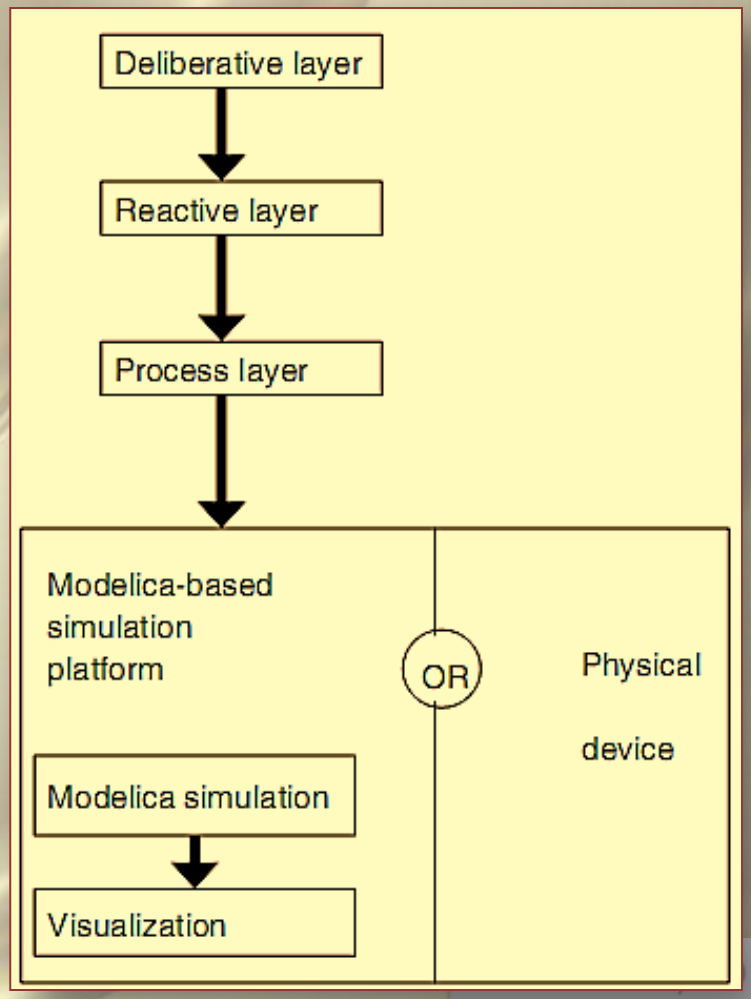
In the project there is a need to develop a system which contains helicopters, robots and various control software and hardware. In particular there is a need to simulate the dynamic behavior of an autonomous aircraft within a virtual environment. There is a need to simulate a service environment, where robots can interact with the landed helicopter.

In this report a study of object-oriented modeling of mechanical systems using *Modelica* is presented. Mechanical features of an autonomous helicopter have been modeled in order to verify the control system. A robot which is able to grab, move and release objects using automatic or manual control has been modeled. The geometry and dynamic structure of these systems has been designed in *CAD* tools and later integrated with control systems for steering these devices. The simulation has been performed in *Modelica*.

Layers of simulation architecture for an autonomous vehicle

The most complex part of the project includes design of control systems working in several layers. These control systems are organized as software tools in three layers:

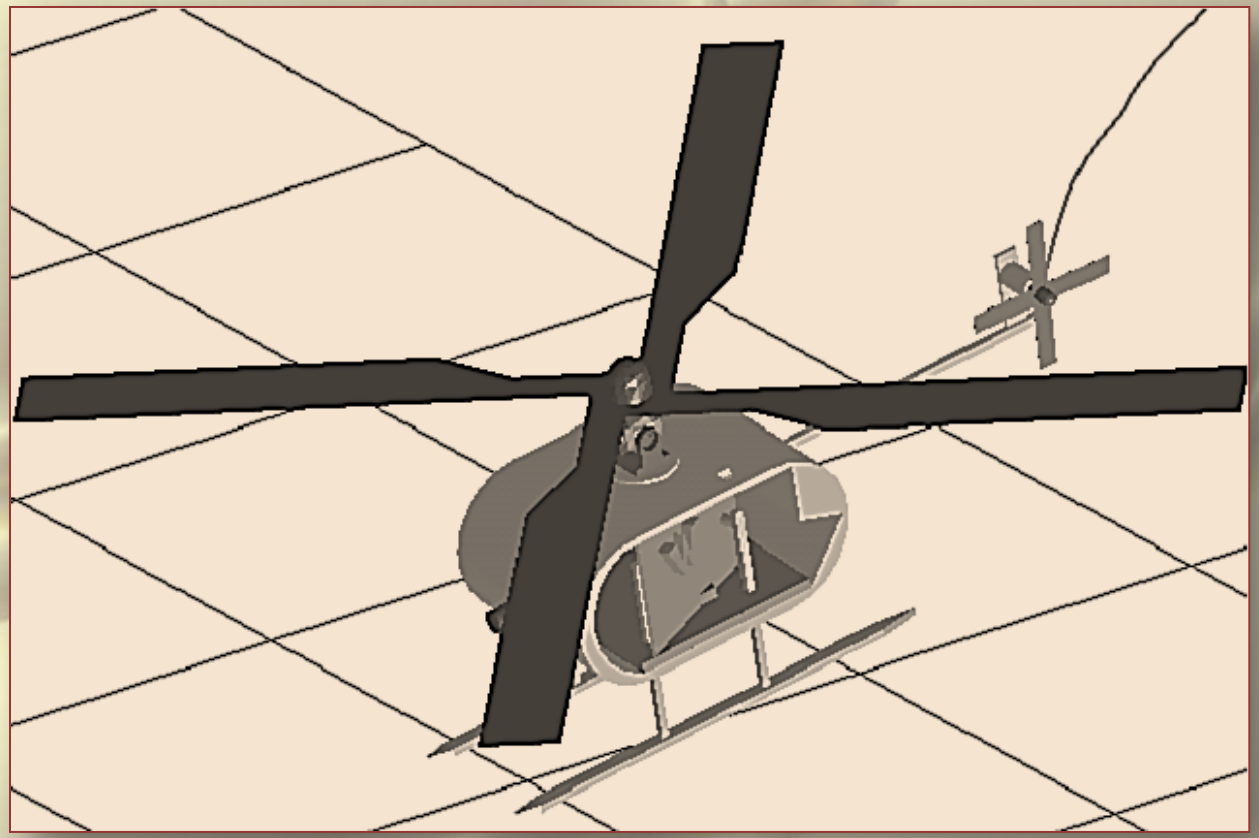
- A *deliberative* layer produces plans, e.g. a plan of movement.
- A *reactive* layer produces responses when certain events happen.
- A *process* layer gets information from sensors and produces signals for mechanism actuators (e.g. motors).



The control system

The part of the control system which communicates with the Modelica model is the process layer. This system takes the plan (helicopter mission) as commands expressed in a language FCL (Flight Command Language).

Helicopter model designed in *SolidWorks*:



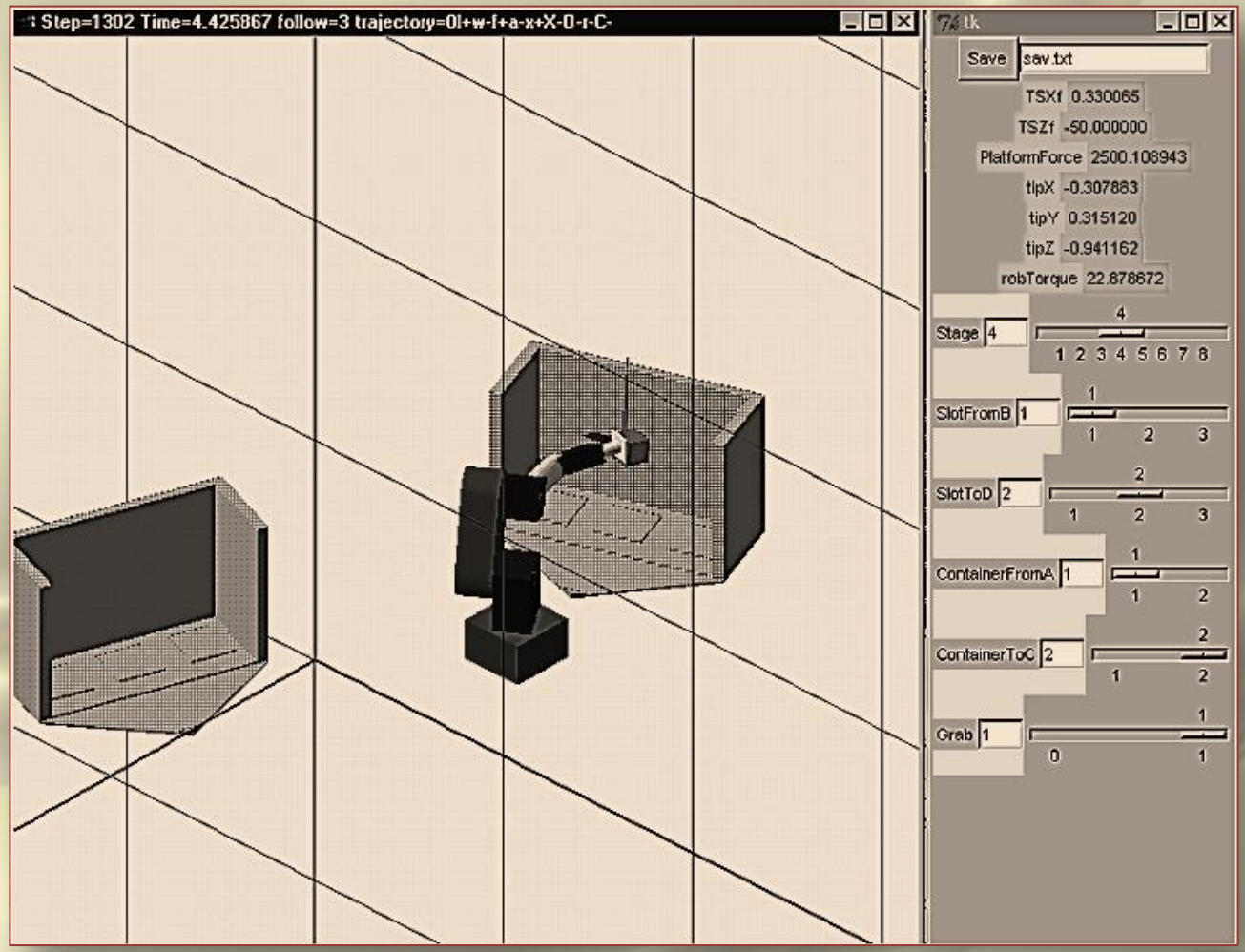
Robot Modeling

In a typical situation the task of the robot can be described as ”to take some load from one container and to place it to another container”. The position and rotation of the containers are known to the control system of the robot. Finally the robot modeling problem was formulated the following way:

- *Input*:
  - Given positions of containers (later also slots) and their identifiers, such as c1 or c2; Given a sequence Robot Command Language (RCL) commands (missions), such as MOVE(c1,c2).
  - Given a virtual robot constructed in SolidWorks.
- *Output*:
  - Torques applied by the motors in order to perform the mission, for instance, to grab a load from container c1 and move it to container c2.
  - The movement trajectory of the robot and all its joints when the mission is performed.

Environment Model

A virtual environment includes a robot, load (a small cube) and two containers:



Scenario for Load Movement

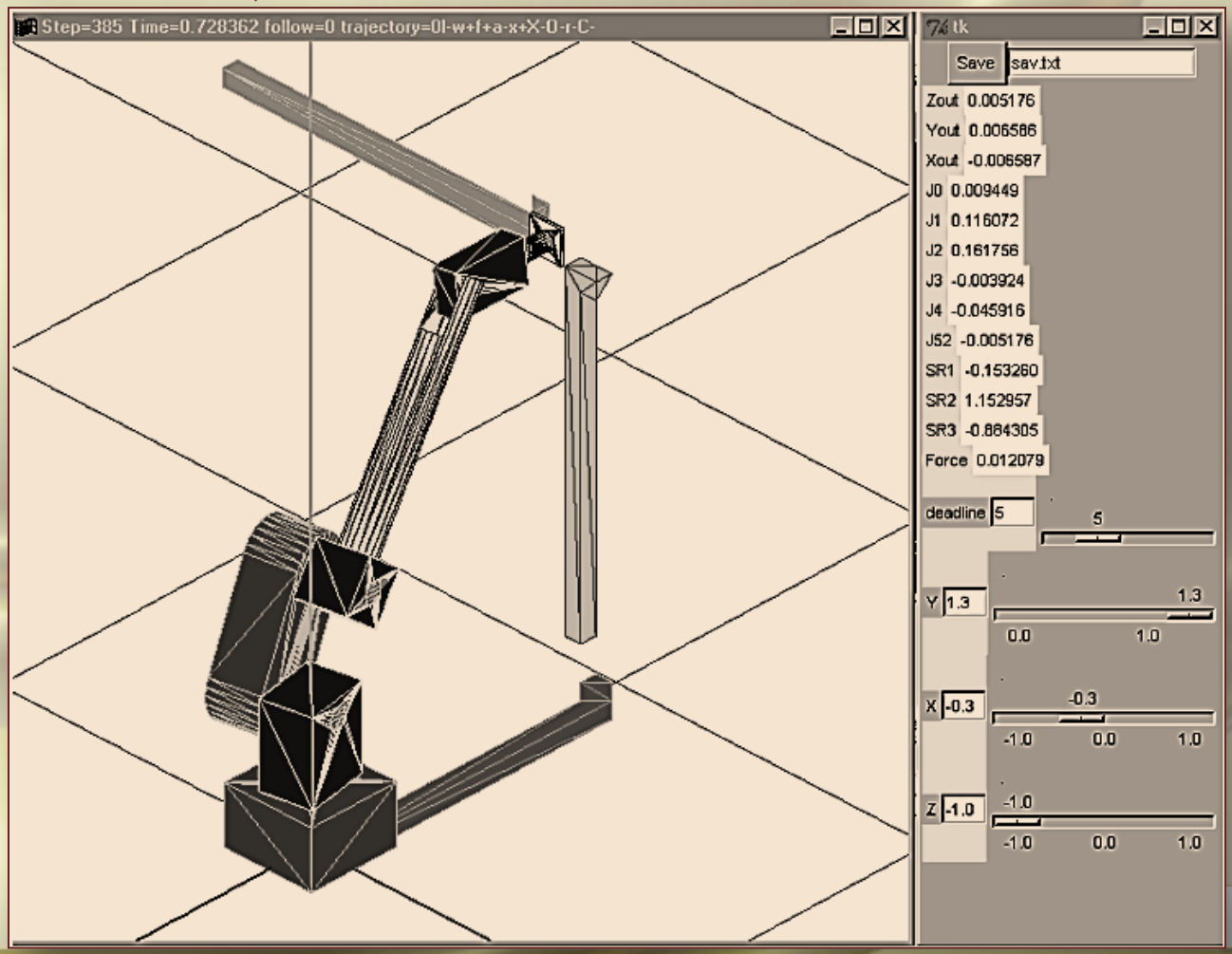
The controller switches from one step to another when certain conditions become true. At each step specific target values for the actuators are specified. The platform motors can move on the plane between the origin position, source and target container. Every container has a reference point where the platform should be located. The actuators for the platform compute force necessary to move the platform in needed direction.

Steps of scenario for moving a load from source to target:

Step	Platform	Manipulator	Grabbing	Terminate if
0	origin	home	off	immediately
1	source	home	off	$D_{platf} < \varepsilon$
2	source	source	off	$D_{manip} < \varepsilon$
3	source	source	on	$D_{grab} < \varepsilon$
4	target	home	on	$D_{platf} < \varepsilon$
5	target	target	on	$D_{manip} < \varepsilon$
6	target	target	off	immediately
7	origin	home	off	$D_{platf} < \varepsilon$

The Inverse Robot in 3D

The inverse robot with additional bars has a kinematic loop. Required position of the tip is set up by the three scale bars (for X, Y , and Z):



References

1. *Interactive simulation and visualization*, Johnson C.; Center for Sci. Comput. & Imaging, Utah Univ., Salt Lake City, UT, USA ; Parker, S.G. ; Hansen, C. ; Kindlmann, G.L., ...
2. *A multilanguage environment for interactive simulation and development controls for power electronics*, Lovett, T. ; Dept. of Electr. Eng., South Carolina Univ., Columbia, SC, USA ; Monti, A. ; Santi, E. ; Dougal, R.A.
3. *An Environment for Design, Simulation and Interactive Visualization for CAD Models in Modelica*, Vadim Engelson, Hakan Larsson, Peter Fritzson, Linkoping University, Sweden, Proceedings of 1999 IEEE International Conference on Information Visualization, IEEE Computer Society, 14-16 July 1999, London, pp. 188-193,
4. *Simulation and Visualization of Autonomous Helicopter and Service Robots*, Vadim Engelson, PELAB, IDA, Link¨ oping University
5. *3D Systems, Stereo Lithography Interface Specification, 3D Systems*, Inc., Valencia, CA 91355, http://www.vr.clemson.edu/credo/rp.html.
6. *Dymola, Dynamic Modeling Laboratory, User's Manual, Version 4.0,* Hilding Elmqvist, Dag Bruck, Martin Otter,, from Dynasim AB, Research Park Ideon, Lund, Sweden, http://www.dynasim.se