MATLAB/Simulink as a building block for FunctionalDMU

Dr. André Stork, Mathias Wagner

Fraunhofer-Institut für Graphische Datenverarbeitung IGD, Darmstadt

Peter Schneider, Dr. Olaf Enge-Rosenblatt, André Schneider

Fraunhofer-Institut für Integrierte Schaltungen IIS, Institutsteil EAS, Dresden

Dr. Thomas Bruder, Christiane Schäfer

Fraunhofer-Institut für Betriebsfestigkeit und Systemzuverlässigkeit LBF, Darmstadt

Andreas Hinnerichs, Carsten Neumann

Fraunhofer-Institut für Offene Kommunikationssysteme FOKUS, Berlin

ABSTRACT

Digital Mock-Up (DMU) is a widely introduced technology to virtually investigate geometrical and mechanical product properties. Functional Digital Mock-Up (FDMU) is a combination of traditional DMU with behavioral simulation in mechatronics. Enhancing DMU with functional aspects, considerably more insight in product properties can be gained. To enable FunctionalDMU two main tasks have to be solved: a) simulators in the areas of mechanics, electronics, and software simulation have to talk to each other (coupling) and b) the simulation results have to be visualized in an interactive DMU environment. In this paper we present an independent and open approach to a FunctionalDMU framework including co-simulation where MATLAB/Simulink can be used as one building block. Starting with proprietary and natively given behavior and geometric models (in formats like JT), we wrap the behaviour models into SysML to enable data exchange on an agreed and standardized format. The native behavior models still are executed in the corresponding simulators. The simulators are linked to the FDMU framework using a wrapping approach. Currently we support simulators such as Simulink, Dymola, Saber, Rhapsody, Simpack, ANSYS with our framework. During simulation a simulator coupling algorithm controls the simulation processes. A dedicated visualization environment enables the user to interact with the simulation, i.e., to send stimuli, change parameters, observe the simulation run etc. This paper introduces the components of the FDMU framework and illustrates the approach with an application example.

INTRODUCTION

In industrial product development, mechatronic components play a rapidly increasing role. Mechatronic components are characterized by the fact, that electronic and software controlled systems influence mechanical parts. In the development process of mechatronic products the domains software engineering, mechanical engineering, and electronic engineering are considered.

For the geometric integration of product models, Digital Mock-Up (DMU) is established as an inherent part of virtual product development in industrial practice. However, DMU and the associated software tools are nowadays mostly limited to the integration of geometry - possibilities for functional integration are missing.

In order to put mechatronic products on a solid base in an early stage of the development process, an extension of DMU with functional aspects towards a *Functional Digital Mock-up* and the support of cooperation between the domains of mechanics, electronics and software development is imperative.

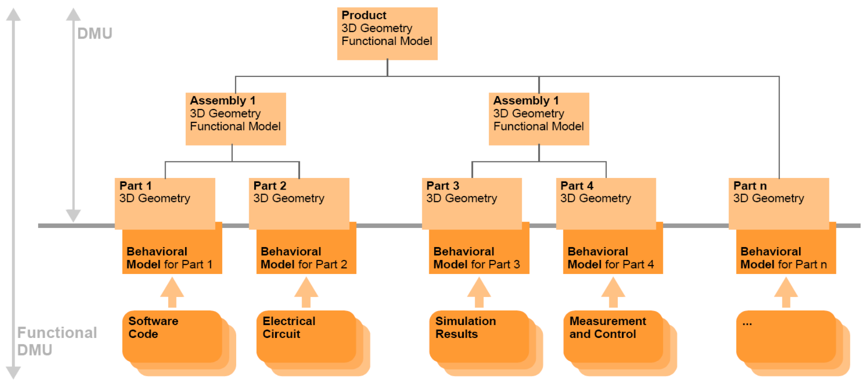
DMU
Functional DMU
Product 3D Geometry Functional Model
Assembly 1 3D Geometry Functional Model
Assembly 1 3D Geometry Functional Model
Part 1 3D Geometry
Part 2 3D Geometry
Part 3 3D Geometry
Part 4 3D Geometry
Part n 3D Geometry
Behavioral Model for Part 1
Behavioral Model for Part 2
Behavioral Model for Part 3
Behavioral Model for Part 4
Behavioral Model for Part n
Software Code
Electrical Circuit
Simulation Results
Measurement and Control
...

Figure 1: Extending the classical DMU approach with behaviour models to achieve FunctionalDMU

OBJECTIVES

The objective of the described work is the development of a methodology and a framework for the cooperative development and validation of functional prototypes of complex mechatronic products. This flexible framework will permit experiencing functional virtual models. In detail, the objectives are:

- functional integration and validation of virtual mechatronic systems,
- visualization of the interplay of mechanics, electronics and software as a fast and easy way to experience system behavior,
- combination and superimposition of different behaviour models and efficient analysis of variants,
- early identification and communication of conflicts of objectives,
- identification of best possible alternatives for finding cross-domain optima.

COUPLING OF SIMULATION DOMAINS VIA FDMU MASTER SIMULATOR

The basic idea of *Functional DMU* is

- to couple functional and geometry models within a total system model and
- to combine functional simulation and interactive 3D visualization.

One challenge in this context is to find and create basic model components which support a flexible co-simulation and online-visualization. For this reason the concept of *Functional Building Block* (FBB) has been developed. An example is illustrated in *Figure 2*. The FBB Motor covers a functional model describing the electro-mechanical behaviour and 3D CAD data describing all geometrical/visible components of the motor.

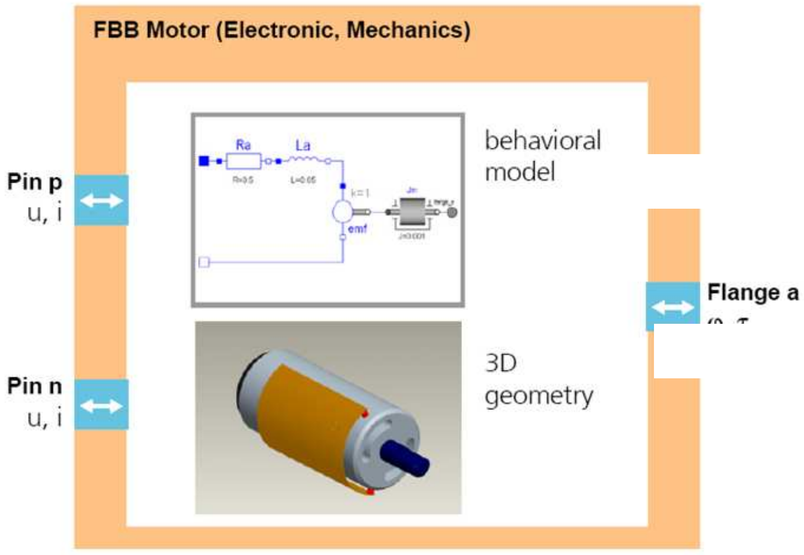

Figure 2: Example for a Functional Building Block (FBB).

A FBB shows one of the main concepts of the *Functional DMU* approach: 3D CAD models are no longer 'dead' models. With a functional simulation in their 'background', they represent and demonstrate the correct physical behavior.

Of course, FBBs are only one part of the concept. The second challenge of *Functional DMU* is to provide a *FDMU framework* as runtime-environment to execute the behavior models nested in FBB. At least three aspects are important:

- Simulation and visualization tools should be integrated in the framework via open interfaces.
- The framework has to organize the data exchange between all FBB within a total *FDMU Simulation Model* (FSM).
- The framework has to provide appropriate co-simulation algorithms for simulators executing the behaviour model.

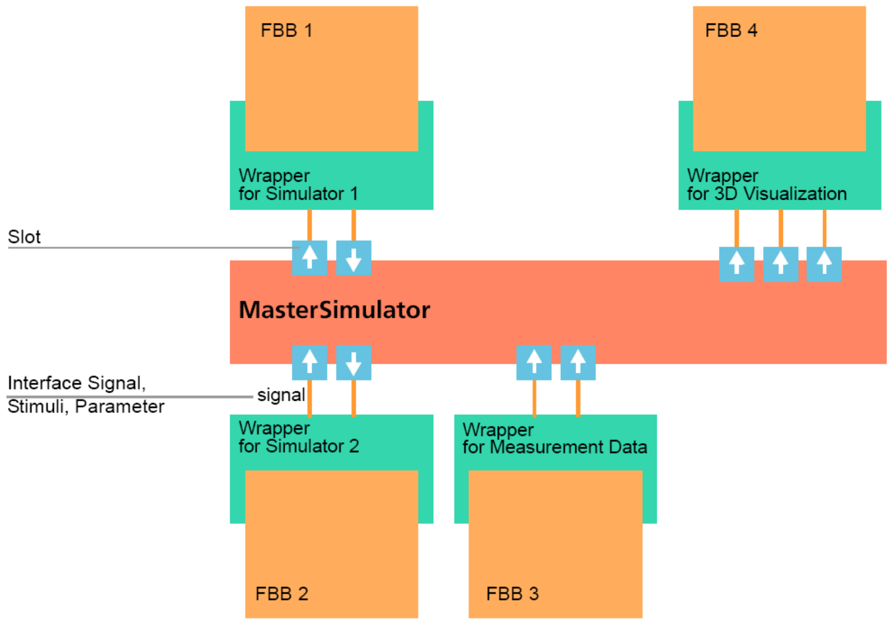
FBB 1
FBB 4
Wrapper for Simulator 1
Wrapper for 3D Visualization
Slot
MasterSimulator
Interface Signal, Stimuli, Parameter
signal
Wrapper for Simulator 2
Wrapper for Measurement Data
FBB 2
FBB 3

Figure 3: Main components of the overall FDMU framework.

Figure 3 shows some main components of the FDMU framework. The FDMU Master Simulator provides all data exchange facilities and co-simulation algorithms. Each integrated tool is encapsulated by a small adaptor called FDMU Wrapper. All wrappers have a common interface. The wrappers perform all interactions with the Master Simulator.

The main task of the Master Simulator is to send and receive data to and from wrappers. A *Slot* serves as a unique communication endpoint within the Master Simulator. An example is shown in *Figure 4*: FBB *Switch* sends the signal *up* to FBB *Controller*, inside the Master Simulator this connection is represented by the output slot *Switch.up* and the input slot *Controller.up (output here* means that the FBB *sends* signals and *input* means the FBB *receives* signals). To establish a connection as shown in *Figure 4* the Master Simulator has to connect slot *Switch.up* with slot *Controller.up*.

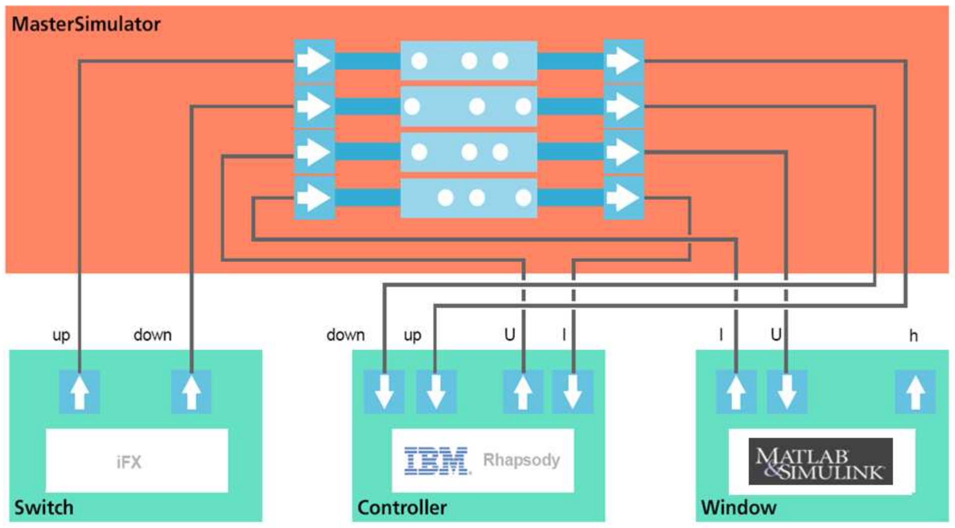

Figure 4: Data exchange via Master Simulator.

DATA TRANSFER HANDLERS

The simplest case for connecting two slots is performing a direct data transfer. In this case each data value received by the Master Simulator at an output slot (e.g. *Switch.up*) will be sent (without any processing) to the connected input slot (e.g. *Controller.up*). The FDMU Master Simulator supports data transfer with more complex behavior too. Dedicated transfer handlers provide re-sampling functionality and protocol conversion. *Figure 5* illustrates some examples for down- and up-sampling and conversion between event-based and sampled data flows. The red arrows are the timelines (simulation time) and signal values are available at the black markers.

In the future the concept of transfer handlers will be further improved. Currently the stream handlers connect one output slot to one input slot. It is planned to remove such restrictions and to implement more complex handlers for any number of output/input slots. These new kind of handlers will be fundamental for realizing dedicated co-simulation algorithms. A first version for calculating optimal step-size (sample rates) for single couplings between FBB is currently in experimental stage and will be evaluated for application examples in the near future.

The set of all directed slot-to-slot connections can be described as a routing table. All necessary information can be derived from the FDMU Simulation Model (FSM), which consists of all FBB and connections between them. The Master Simulator supports static and dynamic routing. In case of static routing the Master Simulator loads a routing table at start-up time. Furthermore, during a running FDMU session a wrapper/client can dynamically establish new connections via API functions *addSlot()* and *connectSlots()*.

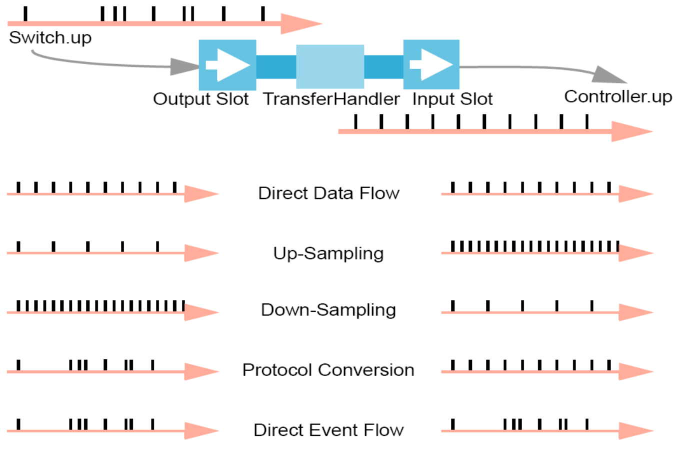

Figure 5: Transfer Handlers for re-sampling and protocol conversion.

As described above, the Master Simulator and their slots play a key role for connecting FBB. In addition to data transfer the synchronization between different FBB is another important task of Master Simulator. While the main part of synchronization is implicitly performed through the (sampled) data flow between FBB, the Master Simulator tries to decouple the co-simulation of all FBB as good as possible. For this reason, all transfer handlers work in parallel in different threads. To avoid unnecessary blocking during the data transfer, all slots can buffer incoming data values. From the wrappers's (simulator's) point of view data can be sent as fast as possible and without waiting for free processing time in the Master Simulator. Of course, simulator coupling typically slows down performance. Therefore, the FDMU framework allows to combine several FBB within one wrapper and to simulate them with the same simulator instance. So, unnecessary couplings can be avoided to improve overall performance.

DATA EXCHANGE

Within the FMDU framework a great variety of data items can be exchanged between FBB. The Master Simulator differentiates between the following categories: *type*, *cardinality*, and *mode*. *Type* means one of basic type like *DOUBLE*, *FLOAT*, *INTEGER*, *LONG*, *BOOLEAN*, and *STRING*. Cardinality describes whether a single value (*SCALAR*) or an array of values (*ARRAY*) are transferred. And there are two modes: *VALUE* and *SAMPLE*. Sample means data items with timestamp information – transferred as ordered pair *(timestamp, value)* and value means data items without any explicitly attached time information.

As explained above, slots support queuing of data. With this feature in mind all slot-related data access functions provide an appropriate time semantic. For sending and receiving data many groups of API methods are available.

Based on the Master Simulator's data exchange API many data transfer and synchronisation protocols can be implemented. The API is powerful enough to provide flexible co-simulation and fast online visualization within the FDMU framework.

WRAPPERS

Since a FBB contains functional and/or geometry models, a simulation and visualization environment is needed at runtime. If, for example, FBB Motor contains a MAST model for describing the electro-mechanical behaviour, a Saber simulator is used for simulating the functional model. If an FBB contains a MATLAB function, MATLAB/Simulink is used, of course. Note that many simulations can be used in combination.

The analogue holds true for geometry models, CAD data in various formats can be processed and visualized. The FDMU Visualization (iFX [4]) based on OpenSG has been used and extended towards the FDMU requirements to display and interact with all 3D geometry models.

The FDMU Wrappers integrate such tools (simulators, legacy code) into the FDMU Framework. Main tasks of the wrappers are:

- provide a common interface and hide all tool-specific properties;
- perform start-up and shutdown of encapsulated tool;
- handle errors and exceptions;
- convert input and output data if needed

As an example, the wrapper for Simulink will be explained in detail. The wrapper is implemented in C. Its interface provides functions for configuration, initialization, start, stop, suspend, resume, and termination of the encapsulated simulator. The wrapper starts Simulink as a separate process. All user interactions are suppressed. Simulink receives control commands from the wrapper using inter process communication and runs the simulation. All logging information is caught by the wrapper and cached for later analysis.

For using Simulink models within the FDMU framework, the models have to be slightly modified

- All input and output signals as well as parameters, the user wants to communicate, see and/or modify, have to be identified.
- Each input and each output signal to be connected to an external connector block which implements the internal communication to the wrapper. For this end an adaptor library has been developed (see Figure 6).

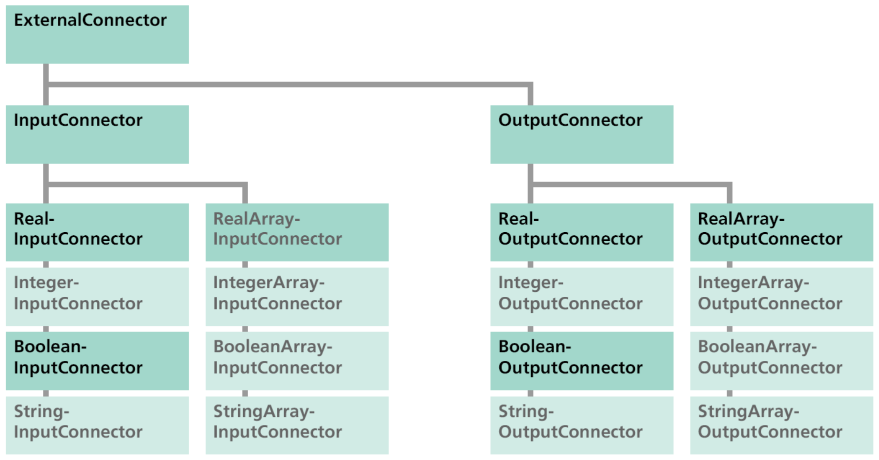

Figure 6: Connector library available for some simulators

The implementation of an external connector block depends on the modelling language and the simulator used. For the FDMU framework, solutions for Simulink (Mathworks), Modelica (Dymola) and MAST (Saber) have been developed. Many modelling languages support the integration of external C functions or external user models implemented in C. As an external connector can be integrated as C code, a TCP/IP-based communication between the simulation model and the wrapper is straightforward.

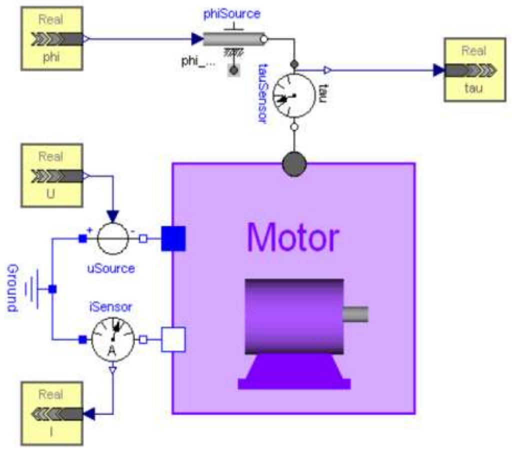

Figure 7: Examples of external connectors for a simple motor model.

INTERACTIVE 3D VISUALIZATION

3D visualization plays a key role in DMU and especially in the FDMU framework. The implementation of the visualization components is based on the iFX scene graph viewer software [4]. iFX is a flexible and extendable framework for the interactive visualization of simulation data. It supports cooperative work over low-bandwidth network connections, it is based on OpenSG [2] and it can handle huge models with millions of elements. iFX runs on Windows and Linux operating systems and supports multi core processors. The architecture of iFX was designed to be extensible from the ground up. Thus, FDMU-specific extensions could have been integrated quite easily.

iFX has been integrated in the FDMU framework via a wrapper. At start-up time, iFX loads the geometry data referenced by the FBBs contained in the FSM and displays the geometry model. During the simulation, iFX visualized object transformations as calculated by the behavior models, esp. the multi body system. Furthermore, it displays all simulation values and parameters in an integrated way, thus generating a single point of access to the possibly distributed co-simulation. In addition, it displays the parameters of all FBBs to the user and allows him/her to alter simulation parameters at runtime.

A couple of additional analysis and documentation features are available in the interactive visualization environment:

- Collision detection
- Visualization of signal flow between source and destination units, e.g. ECU and motor
- Cut planes
- Distance measuring – even under dynamically moving objects (steered by the simulation)
- Annotation creation

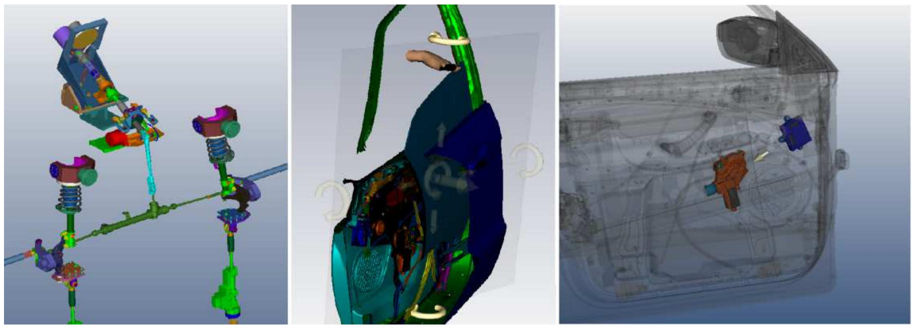

Figure 8: FDMU Visualization: complex model (left), cut plane feature (middle) and source-destination visualization (right).

Another feature (see figure 11) are 2D graphs to display simulation values in detail. Using 2D graphs, a user can get a deeper understanding of what is going on, in addition to just looking to the transformed 3D geometry. He/she can observe system response after changing input/stimuli signals without any complex or difficult result interpretation process.

The FDMU framework is not limited to the intrinsic FDMU visualization, other visualization environments can be linked to the Master Simulator as well, e.g. JT-based visualization tools. Similar to integrating simulators, 2D or 3D viewing tools can be plugged in via wrappers as long as they provide sufficient APIs.

FDMU APPLICATION EXAMPLE: WINDOW REGULATOR

The application example *Window Regulator* demonstrates the co-simulation of functional models from the three different physical domains: software, electronics, and mechanics. Furthermore, the example allows user interactions during a simulation run and shows the results within the 3D FDMU Visualization immediately. If a user pushes the UP or DOWN button of the window regulator the window in the 3D model starts moving correspondingly (see *Figure 9*).

Inside the door, a mechanism moves the side window up and down (*h*). It is driven by a gear of an electric motor (τ, φ). The electrical control unit (ECU) provides the voltage *U* of the motor and is monitoring the current *I* of the motor at the same time. If the current of the motor exceeds a threshold *I_max*, the voltage is switched off by the ECU (*U*=0). Furthermore, the ECU obtains and processes signals *up* or *down* from the push buttons for opening or closing the side window. The evaluation of the signals is carried out by software inside the ECU.

The system model is divided into different FBB corresponding to the different physical domains: *Switch, Controller, ElectricalUnit, Motor, MechanicalUnit, Window*. All FBB exchange their data via an instance of the Master Simulator. The (directed) interface signals are *up, down, U, I,* τ, *φ, h* (see *Figure 9*).

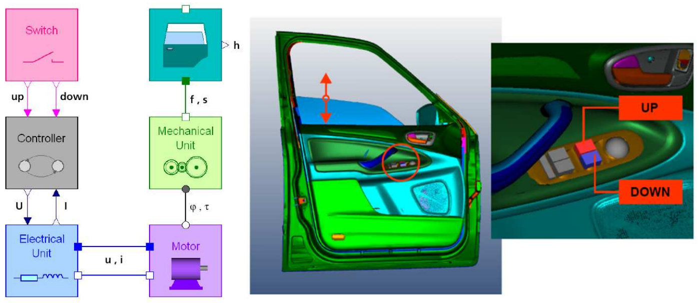

Figure 9: WindowRegulator: System model (left) and interactive 3D FDMU Visualization (right).

The interplay of all components in the framework is shown in *Figure 10*.

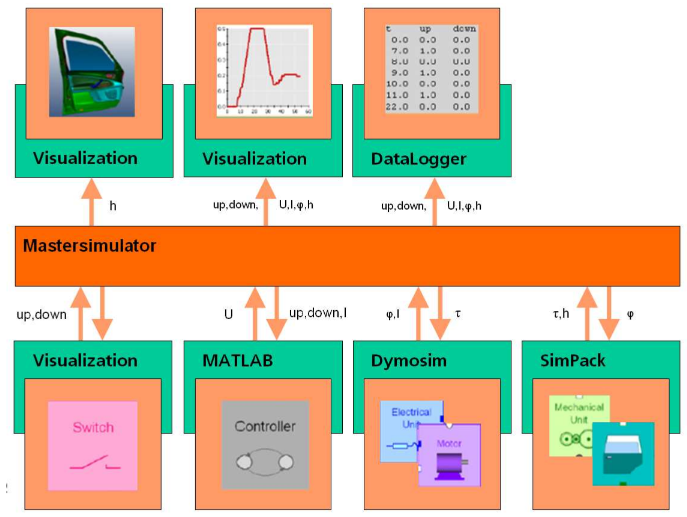

Figure 10: WindowRegulator scenario with all FBB connected to the Master Simulator.

From the user's point of view the following questions may be interesting, which can be answered by using a FDMU simulation:

- How fast moves the side window?
- Will an obstacle block the movement of the side window?
- Does the system work with changed values of friction?
- Does the controller work correctly?
- Which impact do motor parameters, friction, and current threshold have?

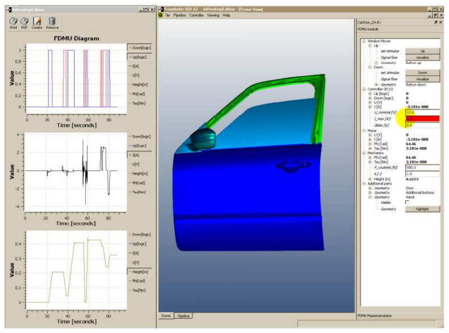

Figure 11: 2D/3D online visualization and interactive parameter configuration.

The FDMU framework allows to interactively influence a running simulation. A situation out of one experiment is shown in *Figure 11*. The user has played with different gear ratios already to see how fast the window moves under which conditions. Now he/she wants to investigate into the effect of varying friction values onto the software-controlled window mover. In figure 11 he/she just modifies the friction value to study the effect.

The FDMU framework supports the integration of ODE/DAE-based simulation tools as well as solvers based on partial differential equations (PDE) like FEM and multi-body simulators (ANSYS, SimPack). Currently the WindowRegulator example is extended to a more complex system model which includes the simulation of thermal effects. Investigations are focused on the impact of power loss and heat dissipation of electrical unit and the thermal effects around the motor. *Figure 12* shows an extended scenario where ANSYS is included for simulating electro-magnetic fields and thermal effects inside and around the motor.

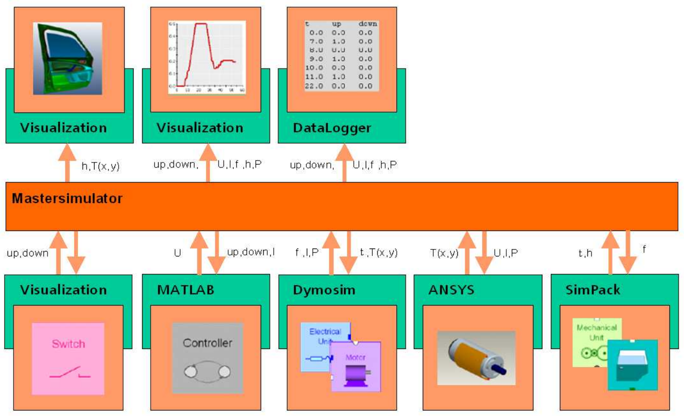

Figure 12: Extended WindowRegulator with all FBB connected via the Master Simulator.

Here the question is how the thermal effects influence the behavior of the electrical unit. The designer's goal is to guarantee the correct window behavior at all times and under different conditions.

CONCLUSION

The paper presented *Function Digital Mock-up* as a fundamental extension of the well established DMU approach. The main advantages are the co-simulation of different physical domains and the combination of functional simulation and 3D geometry visualization in a flexible design environment – the FDMU framework. With the concept of Functional Building Blocks (FBB) users can reuse and combine existing behavior models. A total system model consisting of various FBB can be simulated and visualized within a framework. Data exchange and synchronization between FBB is performed by the FDMU Master Simulator. The FDMU framework provides flexible open interfaces for integration of simulation tools, visualization programs, and other applications supporting virtual product design of mechatronic systems. Currently, FDMU addresses many important challenges in automotive industry. More complex multi-domain systems can be developed faster. FDMU allows early integration to detect potential problems when they are cheap to solve.

REFERENCES

[1] Clauß, C.; Schneider, A.; Schneider, P.; Stork, A.; Bruder, T.; Farkas, T.: Functional Digital Mock-Up für mechatronische Systeme. Multi-Nature-Workshop, Reisenburg-Ulm, 2009

[2] OpenSG: http://opensg.vrsource.org

[3] Schäfer, C.; Voigt, L.; Bruder, T.; Stork, A.; Schneider, P; Clauß, C.; Schneider, A.; Farkas, T.; Hinnerichs, A.: FDMU – Plattform zur Unterstützung der Produktentwicklung mechatronischer Systeme. SIMVEC, Baden-Baden, 2008

[4] Stork, A.; Thole, C.-A.; Klimenko, S.; Nikitin, I.; Nikitina, L.; Astakhov, Y.: Simulated Reality in Automotive Design. International Conference on Cyberworlds, Hannover, 2007