Using 1D & 3D Simulation for Mechatronic System Design

Andrew Dyer
Altair Engineering, Inc.
Troy, USA
adyer@altair.com

Abstract— **Products and processes have become increasingly complex to meet demands on performance, reliability, and cost. As a result, simulation has become a key component to successfully designing and delivering these products to market. Mechatronics--also known as smart or cyber-physical systems—is a marriage of machine to sensors, actuators, and computing power to achieve the system goals which cannot be achieved by a purely mechanical system alone. The Model-Based Development (MBD) process leverages simulation models and can improve the design and delivery while supporting complex products like mechatronics systems.**

Using Altair's suite of tools and partner products, the simulation process from requirements management to functional system analysis, cascading down to component design can be achieved for a Model-Based Development approach. This paper will highlight two of these tools as a part of MBD during the design phase for plant modeling and controls synthesis: solidThinking Activate for 1D system simulation, and MotionSolve for 3D multibody system simulation. As a case study, a 1D simulation model of an active suspension system is explored at different stages of vehicle development including integration with 3D models via co-simulation.

Keywords—1D, 3D, solidThinking Activate, HyperWorks MotionSolve, system simulation, Model-Based Development, Modelica, linear quadratic regulator, LQR, co-simulation, Altair Partner Alliance, Functional Mock-up Interface, CarSim, XLDyn, requirements management, active suspension, mechatronics, cyber-physical systems

I. OUTLINE

This paper is organized as follows:

Section II introduces the Model-Based Development (MBD) process and its benefits for creating and supporting complex mechatronic systems. 1D and 3D system-level modeling tools from Altair that support MBD for mechatronic systems are also summarized.

Section III provides some background on active suspension systems that will be used in several examples to illustrate the use of 1D and 3D simulation tools within MBD.

Section IV develops the functional simulation model of the active suspension within a 1D simulation tool, both the mechanism (plant) as well as the controller model used to explore the design.

Section V reviews usage of a 1D simulation tool within a systems engineering requirements management tool to perform early design studies for products that are composed of many subsystems that interact and may have competing design parameters.

Section VI considers an alternative modeling method using Modelica components, which allows 1D model to be created and shared more easily as well as extend the fidelity by adding more details for the components of the mechatronics system.

Section VII imports the need to couple different simulation tools for full system modeling which may be achieved with a standardized interface called the Functional Mock-up Interface, as well as other features of 1D and 3D tools.

Section VIII describes options for increasing the fidelity of the mechanism (plant) model by coupling 1D with tools like CarSim for vehicle-specific modeling and MotionSolve for more detailed vehicle and general mechanism. An example of MotionSolve + Activate co-simulation is developed to test the active suspension with a high-fidelity 3D model.

Section IX highlights options for creating and validating 1D models based on more detailed 3D models.

II. Introduction – Model-Based Development

Over the last several decades, mechatronic systems have become increasingly prevalent and important to our lives. A key driver of this has been the improvement of the power and cost of embedded controllers and peripherals, which find their way into new products. Mechatronic systems take many forms: they improve safety in vehicles with systems like anti-lock braking and stability control and may be found underfoot in your house in the form of autonomous vacuums. Surgeons use robotics to stabilize their view of the heart for open-heart surgery, and exoskeletons assist people to help with repeated heavy lifting. Drones are now delivering packages.

Vehicles in general are heavily dependent on mechatronics systems to achieve the levels of performance and safety required. E-steering, E-throttle and now E-braking are found in production vehicles. And, these systems are communicating with each to each other for new gains in performance; for example, Mazda's G-Vectoring Control system integrates control of engine, transmission, chassis and body, where sensors monitor steering and throttle position to help the driver more safely and comfortably control the steering behavior while powering through a turn [1].

How can these increasingly complex systems be built within time and budget constraints, while meeting customer requirements for performance, reliability, and quality?

Concurrent design of mechatronics systems – considering all systems – is not yet mainstream. Historically, the software for embedded control systems is developed after the design phase which includes the mechanism and selection of actuators and electronics. This approach may prove costly if the design requirements are not achievable when the completed system is finally tested.

Model-Based Development (MBD) is a process that helps to manage and reduce the risk of creating complex systems. This process relies on simulation models and flows from abstract to the specific, which means that design starts with system-level requirements, and flows from functional system models to detailed system models to component models, and back up to implementation through testing, as shown in the V-diagram in Figure 1.

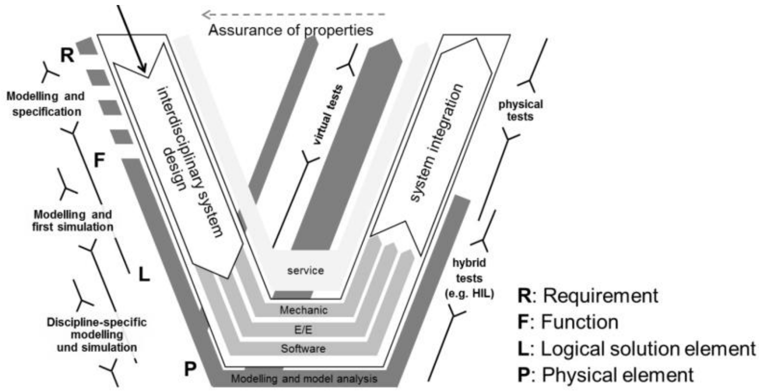

Figure 1: Extended V-model according to Eigner et al.[14]

Throughout this process, simulation assists in driving from the high-level requirements down to component level design. System-level simulation tools help engineers and designers to analyze designs based on different levels of fidelity throughout the process, which is based on the needs of the simulation and the data available at the time.

For example, early in the design process, no component geometry may be available, so functional behavior is captured at this point. Later, as this functional behavior is better defined and more data (e.g., geometry, etc.) is available, detailed models capture the system behavior more accurately as the design comes to form.

While building these models requires investment, this approach pays dividends by providing many benefits critical in meeting design requirements and mitigating problems that are expensive to fix late in the design process:

- Test designs earlier in design process when changes are less expensive
- Perform design studies and optimization with parametric models to best meet requirements and determine the most influential design parameters
- Integrate multi-domain, multi-department systems in an open environment
- Leverage the knowledge of different systems into one model, containing both 1D and 3D models

System-level simulation models can be performed by different tools depending on the preferences of the designers/analysts. Altair offers several tools in the Math and Systems categories to help with Model-Based Development, as modeling approaches and methods vary at different phases of design:

- solidThinking Compose – Math engine, Integrated Development Environment scripting for programmatic models, including OML (an Octave-like language), TCL, and Python
- solidThinking Activate – Block-diagram (1D) multi-domain system level modeling and optimization with system integration via Functional Mock-up Interface
- solidThinking Embed – Block-diagram and state chart (1D) modeling tool with highly efficient code generation for embedded controller hardware, high speed data interface for Hardware in the Loop (HIL) testing
- MotionSolve – 3D Multibody system simulation, with wide array of modeling entities for linear and non-linear plant modeling, open architecture, powerful model creation including CAD import, and co-simulation capabilities

For example, simulation with solidThinking Activate allows a user to model various physical dynamic systems, control systems, actuators and sensors in a convenient block diagram environment, which can also be coupled via co-simulation with MotionSolve for more detailed plant modeling.

Activate is also a platform for multi-domain system integration via Functional Mock-up Interface and support for Modelica libraries (described in more detail later).

Moreover, models in MotionSolve can be linearized for classical control system design, and its powerful non-linear capabilities helps model critical effects ranging from friction, to component flexibility, to contact between colliding bodies. Additional capabilities are available for integration with Fortran/C/Python which may be used to work with legacy models. Finally, integration with other HyperWorks tools allows engineers to perform multi-physics simulation and perform component-level design optimization.

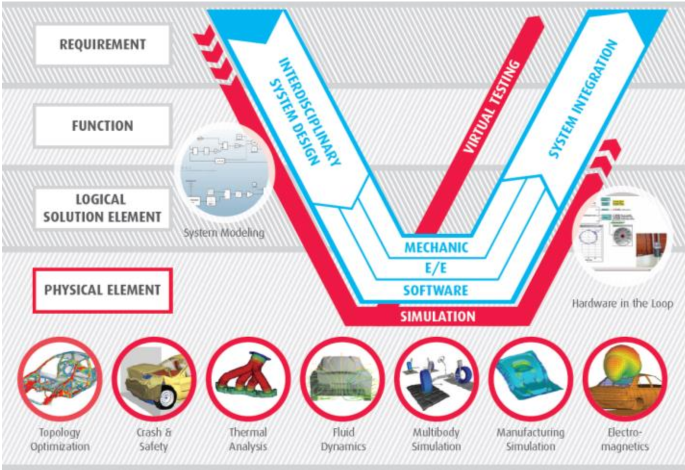

Figure 2: Altair's Simulation Support in the V-Model [15]

Through examples of an active suspension system, this paper will explore how 1D and 3D system models can be leveraged in the Model-Based Development process for varying levels of model fidelity. These examples will range from initial controller design exploration with a simplified Activate model to detailed 3D multibody simulation to test the mechatronics system via Activate + MotionSolve co-simulation.

III. Introduction – Active Suspension

Automotive suspensions provide utility in vehicle design by increasing comfort of passengers and controlling the vehicle response to driver inputs and road disturbances. A more detailed list of design requirements for these systems usually includes ride comfort, durability, handling (performance, safety), and packaging. Optimizing these systems requires compromise on design factors that compete to achieve desired performance. Many different suspension types can be found for passenger vehicles. The majority of these rely on passive systems with spring and damping characteristics that are tuned to meet the objectives of the vehicle. A common design is to use a coil spring and shock combination between control arms and/or the vehicle chassis.

Shocks have a big influence on ride comfort and handling of vehicles as they are a main source of damping in the suspension. Typically, the shocks are hydraulic and control the damping characteristic by forcing the fluid through small holes.

Active or semi-active suspension systems are used to improve ride and handling behavior compared to a passive system. Semi-active suspensions control the damping behavior, while (fully) active versions provide direct actuation on the suspension. This is usually added in parallel with a passive system so that if the active system fails, the passive spring/shock will still support the vehicle. Some modern active suspensions have tunable settings that allow the driver to change the behavior of the suspension [2].

While active suspensions have been around for many years, the technology that helps build these systems has improved to make them perform better and at less cost, and thus are more broadly feasible.

If we consider the high-level topology of a fully-active suspension, a controller is used to compute the actuation force to improve on the passive system performance. Typically, the motion of the wheel is measured via sensors and used by the controller to provide the strategy for input to the actuator. Different strategies have been studied for control, from PID to fuzzy logic, optimal and sliding mode control [3].

The next section describes the details of how this system is modeled and how the controller strategy is developed.

IV. Functional Plant and Controller Design

Early in the design process and before the detailed design is available, simple 1D models can be used to help explore designs. In this section a model in solidThinking Activate illustrates this approach.

The mechatronics system is divided into four main categories:

1. Plant (mechanism or process to be controlled)
2. Controller
3. Sensors
4. Actuators

Initially, to analyze a control strategy in this section, only the first two categories will be considered.

A 1D plant model is often a relatively simple model with limited fidelity, which captures the functional behavior to design control systems. This simple model also has the advantage of being fast, which can provide real-time performance needed for hardware-in-the-loop simulation.

Often these models (typically non-linear in response) are approximated as a linear system – these are simpler models and helps provide insight to the behavior of the system. They also allow the use of linear controls analysis tools to help analyze and design the controller.

Plant Modeling

The plant model in this case is a quarter car ride model of the vehicle including a simplified suspension. This quarter model is a reasonable representation for independent suspension systems with the assumption that the parameters of the system accurately reflect the behavior of the real physical vehicle. It consists of a two-mass spring damper system:

- Sprung Mass – representing the vehicle chassis and a portion of the suspension and driveline components that are supported by the springs
- Unsprung Mass – the wheel/tire and remaining portion of the suspension and driveline components that are not supported by the springs

One spring and damper set is connected in parallel between the sprung and unsprung masses to represent the suspension vertical compliance, and another is between the unsprung mass and ground to represent the tire vertical compliance.

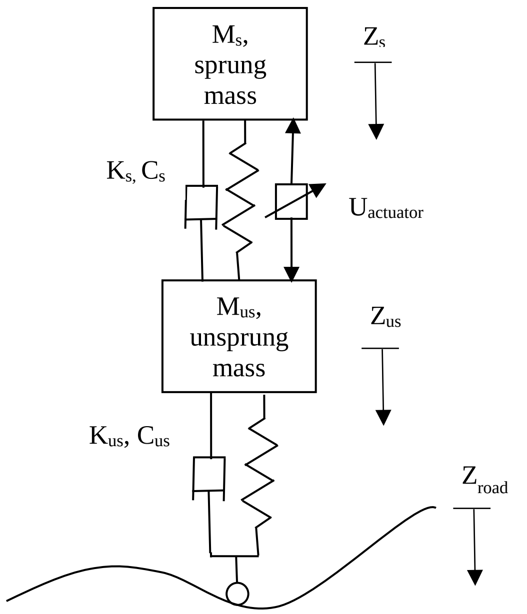

Figure 3: Quarter-car ride model diagram

The equations of motion for the quarter-car ride model are as follows [4]:

$$M_s\ddot{Z}_s + K_s(Z_s - Z_{us}) + C_s(\dot{Z}_s - \dot{Z}_{us}) + U_a = 0$$

$$M_{us}\ddot{Z}_{us} + K_s(Z_{us} - Z_s) + C_s(\dot{Z}_{us} - \dot{Z}_s) + C_{us}(\dot{Z}_{us} - \dot{Z}_{road}) + K_{us}(Z_{us} - Z_{road}) - U_a = 0$$

States:

Tire displacement $(Z_{us} - Z_{road})$
Unsprung mass velocity $(\dot{Z}_{us})$
Suspension stroke $(Z_s - Z_{us})$
Sprung mass velocity $(\dot{Z}_s)$

Dynamic equations such as these can be implemented in different ways within solidThinking Activate, depending on what is needed. For example, because these equations are in linear form, a state-space block may be used to represent the equations of motion for the ride model. In this case, a more general form of equations is used with a *Matrix Expression* block, which allows any general form of matrix equations to be represented and allows for additional terms to be added later, as needed.

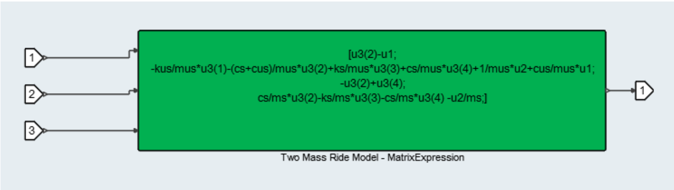

Figure 4: Matrix Expression Block

Inputs to the block shown above are:

1. Road disturbance velocity
2. Actuator force
3. Vehicle states (vector)

Outputs of this block (green, below) are derivative of the vehicle states, which are integrated by the *Integral* block to get the vehicle states, shown below in Figure 5.

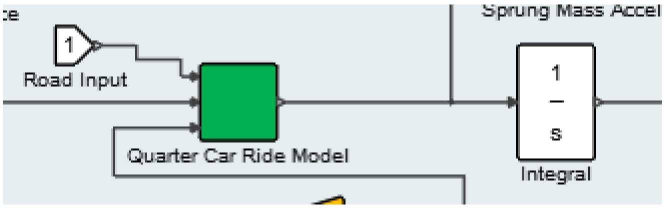

Figure 5: Quarter-car ride model inputs

Controller Modeling

The suspension performance can be evaluated by measuring these outputs in the simulation model:

- Ride: Acceleration of the sprung mass
- Handling: Displacement of the tire
- Packaging: Relative displacement between sprung/unsprung masses

For this model, a Linear Quadratic Regulator (LQR) will be used for control, which is a form of optimal control for state feedback that we can use to balance the requirements for ride, handling, and packaging.

Activate supports a controls toolbox – a library of functions that help to build control systems within command line or scripts within Activate. Using the scripting support, the Controllability condition of the system can be computed to make sure that full-state feedback can be used to implement the LQR control.

Using the function *ctrb()*, we can confirm that the quarter car model is indeed controllable based on the A and B state-space matrices which define the linear system equations.

```
// Passive system equations:
A = [0 1 0 0;
-kus/mus, -(cs+cus)/mus, ks/mus, cs/mus;
0 -1 0 1;
0 cs/ms -ks/ms -cs/ms];

G = [-1 cus/mus 0 0]; //road velocity input

Bu = [0 1/mus 0 -1/ms];//active suspension

Co = ctrb(A,Bu)

print("The controllability matrix is")
print(Co)

//can do the same with this function
print("Is the system controllable?")
print(isctrb(A,Bu))
```

The goal of the LQR is to compute the state-feedback gain matrix that minimizes the cost function, which is a function of weighted states of the system and the input actuation force. The states and input signals are squared to remove sign dependencies giving the cost function a quadratic shape.

Activate supports an LQR function which takes as arguments the matrices for the linear state equations of our quarter car model (A, B) as well as the weights for the states (Q) and actuator force (R). By selecting different values for the weights in Q and R, we can affect the behavior of the system.

The Activate model is parametric, so we can change the design by varying parameters. We can set a string to choose different weights for the LQR function and simulate the different behavior over the different road surfaces.

```
//suspension tuning choice
choice = 'soft'
choice = 'moderate'
choice = 'firm'
```

```
if choice == 'soft' then
print("soft ride")
Q = C`*C // need square matrix
Q(1,1) = 1E3; // weight for tire deflection
Q(2,2)= 10; //velocity of unsprung mass
Q(3,3) = 10; // suspension stroke
Q(4,4) = 1E2; //sprung mass speed
R = 1; // weight for input force
```

Additionally, we can explore adding an observer to the system to eliminate the need to measure all of the states of the system to provide state-feedback control. The Observability of the system can be ascertained with the Controls Toolbox in Activate using the *obsv()* function, and indeed the system is observable based on the A and C state-space matrices.

```
Ob = obsv(A,C_sensors)

print("The observability matrix is:")
print(Ob)
```

An observer makes use of the equation for the plant model to estimate unmeasured states from those states that are measured. As a parallel to the gain for feedback control (K), we want to design a feedback gain for the observer (L) to minimize the error in the estimate for the states that are estimated by the observer. We can use the *place()* function to place the observer poles to be faster than the controller poles to get good performance.

```
L = place(A`,C_sensors`,observer_poles)`
```

The Activate model with full-state feedback is shown in Figure 6 below, with and without an observer to test performance of the observer. Additionally, a *Switch* block is used to quickly toggle topology to test different configurations (with and without active suspension actuator, with or without observer).

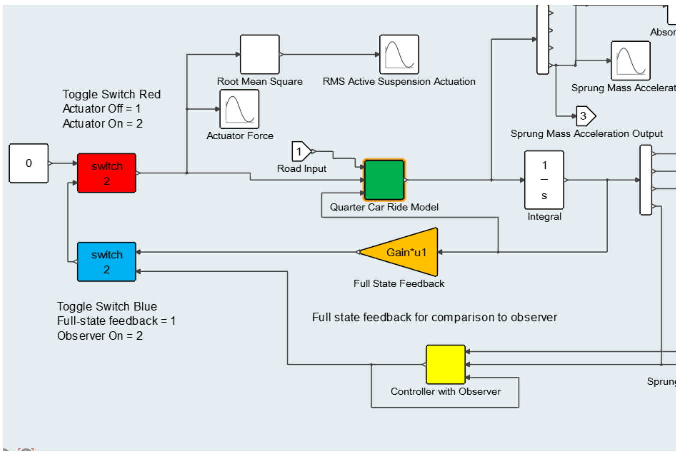

Figure 6: Feedback Controller Added, with Switch blocks to control model topology

In Figure 7, you can see that the model has options for different roads, provided as a velocity disturbance to the quarter car model – two types of single bumps and a rough road, and again a switch block is used to choose which road input is active.

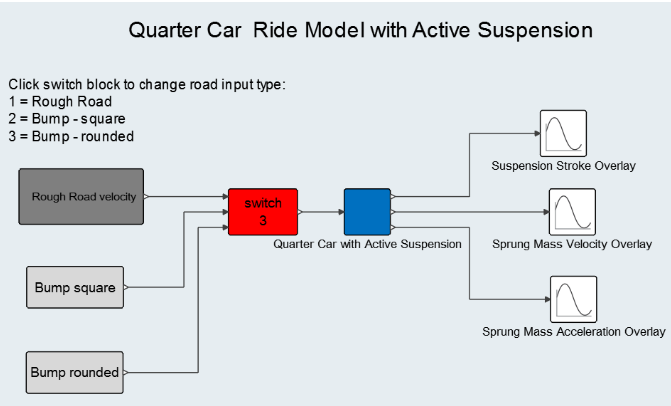

Figure 7: Top-level of model with road inputs

Single Bump

The road model first explored is a single bump, with displacement computed as follows [5]:

$$Z_{road} = Amplitude\,\frac{(1 - \cos(frequency * \pi * time))}{2}$$

Where:

$$frequency = \frac{1}{duration\ of\ bump}$$

The passive system will be compared to three designs for the controller – "soft", "moderate", and "firm", which have different weighting on the states for the LQR controller.

Looking at the suspension stroke (packaging) vs. sprung mass acceleration (ride), for this particular design that the "soft" suspension tuning has the largest peak suspension stroke with the smallest peak acceleration (Figures 8, 9):

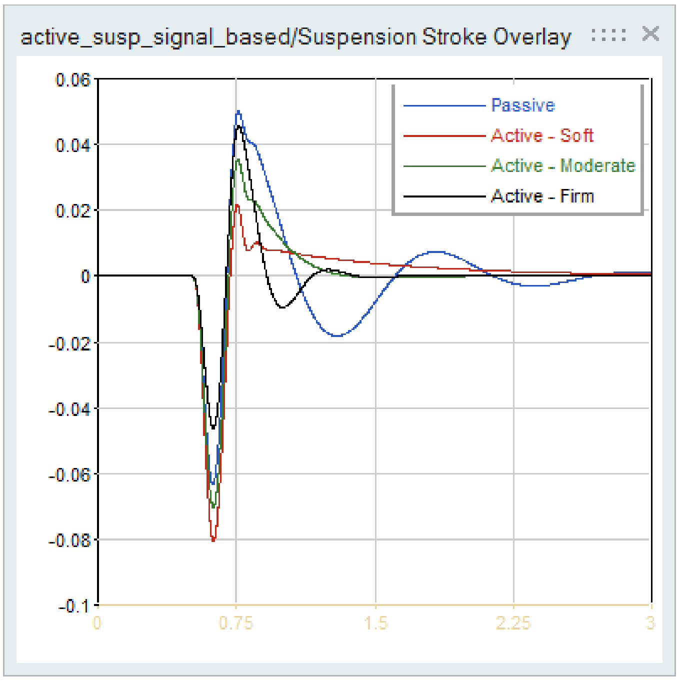

Figure 8: Single Bump Comparison – Suspension Stroke

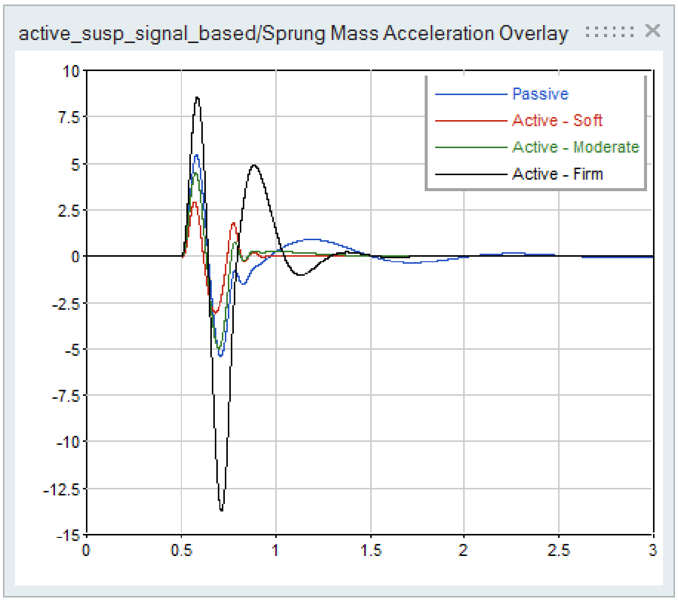

Figure 9: Single Bump Comparison – Sprung Mass Acceleration

The observer may also be evaluated and compared for the single bump test (Figure 10).

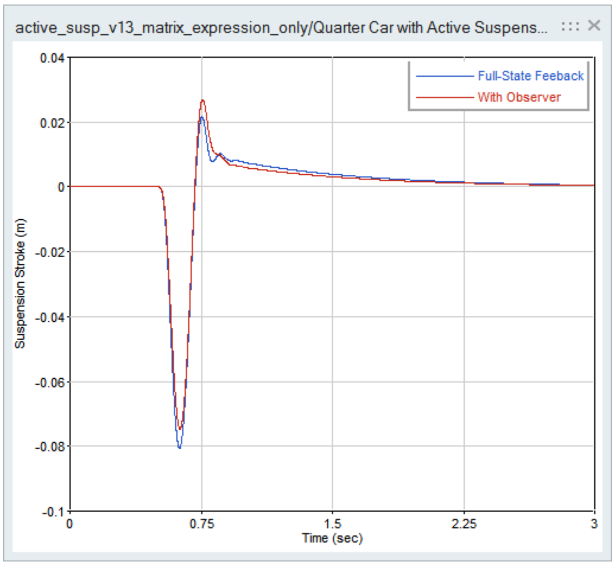

Figure 10: Suspension Stroke with/without observer

Rough Road

Evaluating the vehicle response to rough road inputs is another important consideration for design. The rough road input is modeled with the following equation [6]:

$$Road\ Velocity\ Input = \sqrt{\frac{\sigma^2 \alpha V}{\pi}} random(N)$$

…where symbols sigma and alpha are parameters that depend on the road surface, V = vehicle velocity, and the function random(N) provides N pseudo-random numbers. This is implemented in the Activate model as a superblock and is modeled similarly as follows:

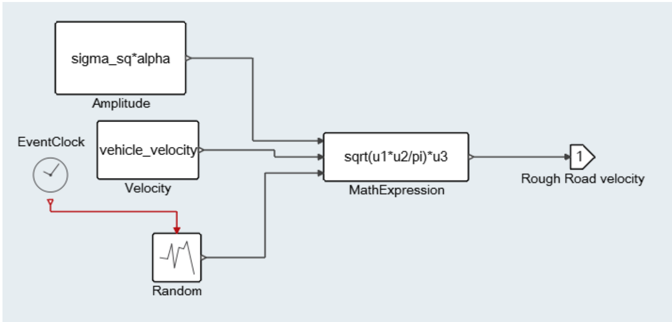

Figure 11: Rough road model

In this model values for sigma and alpha are chosen to represent a "dirt road".

Root Mean Square (RMS) and Absorbed Power computations are used in the model to quantify response to the rough road input. Shown below in Figure 12 is a comparison of a passive suspension to the different settings for the active suspension, and you can see that the active suspension can significantly reduce the overall RMS acceleration and thus improve ride comfort. You can also see that the "Firm" setting increases the RMS acceleration as it weights other requirements more like reducing suspension stroke.

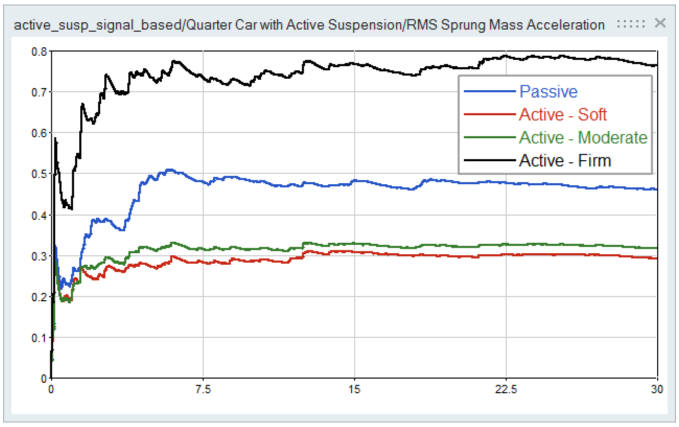

Figure 12: Root Mean Square Sprung Mass Acceleration

This simple model enables different control strategies to be evaluated, e.g., active vs. semi-active vs. passive in order to understand bounds on performance. Other aspects of design that can be explored include discovering design sensitivities, both mechanical and logical, and effects of delays to the control system. As more data is available and the design progresses, the evaluation of sensors and actuators can be added, including effects of noise, sensitivity, accuracy, performance, etc.

V. SYSTEM REQUIREMENTS MANAGEMENT

Functional system models are used early in the design process to explore the feasibility of different control systems and the effects of changes to the plant model, like this active suspension example. However, the active suspension is only one of many systems on a vehicle that need to be designed and the vehicle as a whole will have many competing or coupled design parameters. In order to design the complete vehicle system, the other requirements should be consider and linked to either simulation, test or some other form of evaluation data that can be used to consider design alternatives.

The Altair Partner Product XLDyn [15] is a systems engineering tool to balance system level requirements, helping answer questions early in development through an add-in tool for Microsoft Excel® ™ .

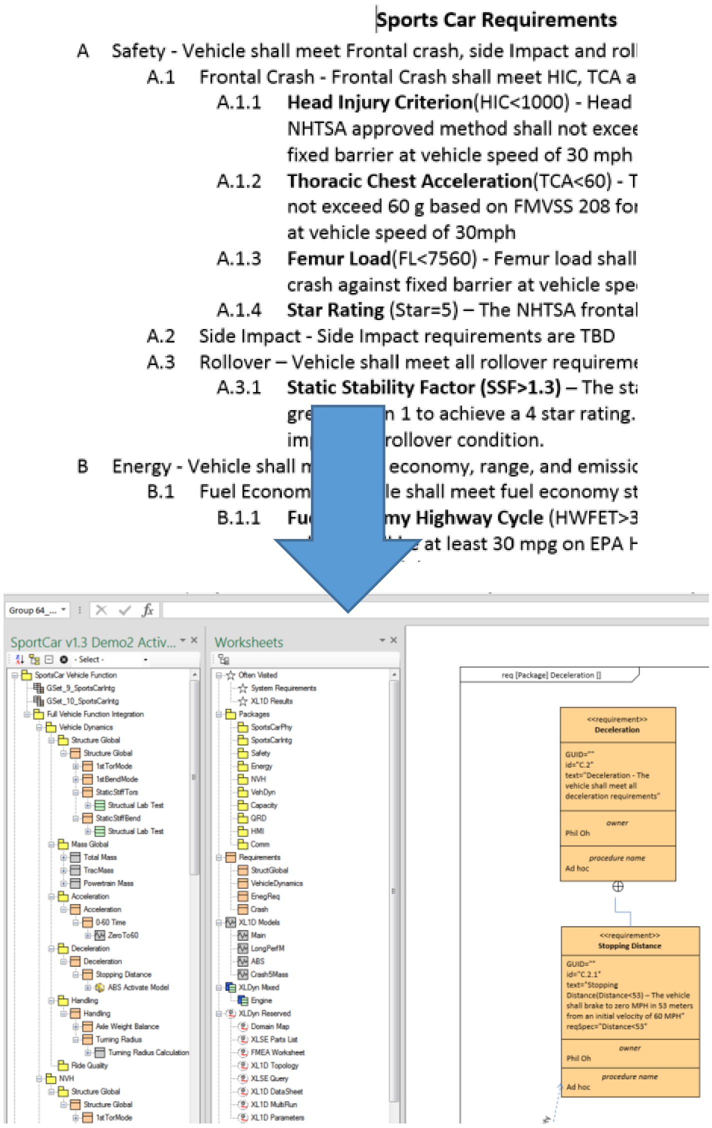

Figure 13: XLDyn creates SysML models from requirements documents for dynamic verification and tracking

XLDyn has both a requirements management feature set built into XLSE, as well as a simplified system modeling tool called XL1D. XLSE can create SysML-based system models of requirements, populated by data in formatted documents, which may be derived from an enterprise data system (Figure 13 above).

XLDyn is also dynamically linked to Activate models, which may be run directly from an XLDyn spreadsheet. For example, an anti-lock braking system (ABS) shown below is modeled in Activate and linked to XLDyn, where any of the design parameters in the Activate model can be changed and resulting design evaluated by XLDyn directly. (Figures 14, 15 below):

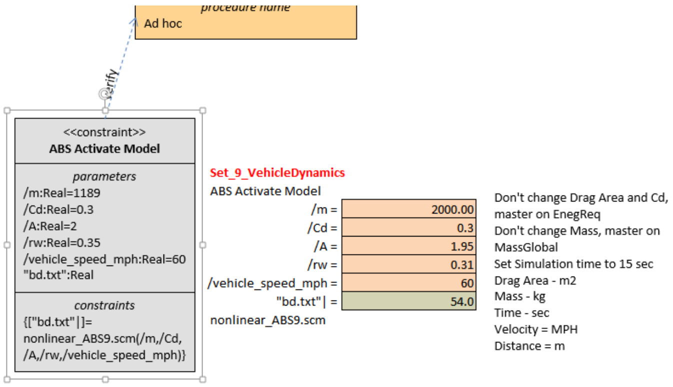

Figure 14: XLDyn integration with Activate

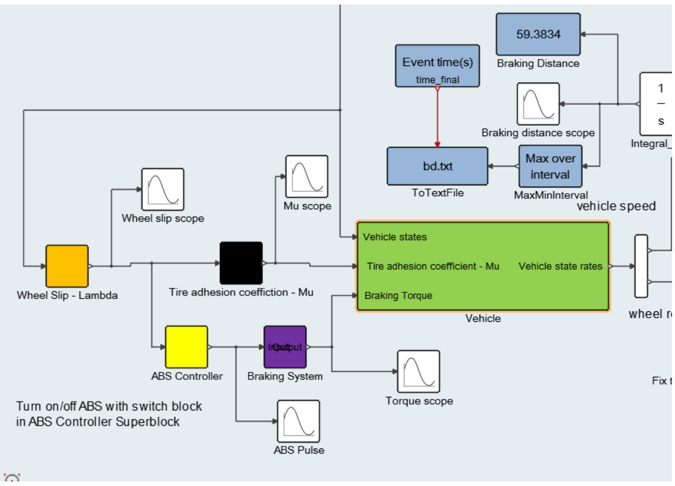

Figure 15: Anti-lock Brake System (ABS) used for systems design in XLDyn

Common design parameters can be varied to see the impact on the entire system, including built-in Monte Carlo analysis. Requirements are verified with Measure of Effectiveness (MoE's) via all available results -- simulation, observation, test, etc. With complete system models and coupled parameters, you can verify design requirements and provide project status to management.

VI. MODELICA-BASED MODELING

As the design process evolves and functional design is established (e.g., via systems design iteration with XLDyn described previously), more detailed models emerge to help realize the design.

An active suspension often implements actuation via hydraulics, and this behavior should be modeled as it typically is non-linear (e.g., Donahue) [7].

While non-linear equations for hydraulics and other effects may be modeled via blocks like the *Matrix Expression* and others, as we used earlier for the suspension ride model, alternatively, the system can represented by dedicated physical components, which represent portions of the real system, like valves, pipe, pressure source, etc. These are implemented via Modelica which has a number of important benefits, described next.

Modelica Modeling

Up to this point, we have modeled the system based on traditional signal-based blocks which are causal – meaning they have a predefined input and output, as in a gain block:

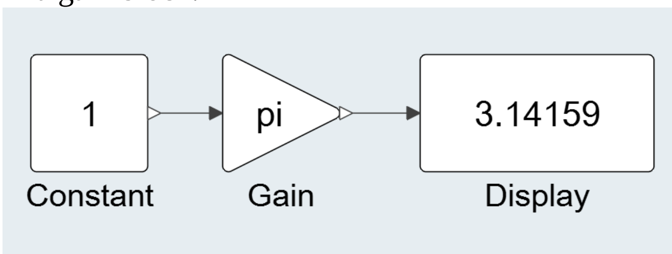

Figure 16: Signal based blocks

This is a very effect method of modeling for control systems where the control algorithms often fit naturally within this scheme.

Another method of modeling is based on the Modelica language (www.modelica.org), which is an object-oriented method design to build component that represent physical entities (e.g., springs, resistors, pipes, etc.) that comprise models. These components are equation-based and can be built into libraries that support different domains. These physical blocks are acausal – they don't have predefined inputs and outputs generally – and a compiler symbolically sorts the assembled system equations and determine the causality as part of the solution process. This makes building models and changing them much easier, as an engineer does not need to reformulate the equations for a signal-based block approach when adding fidelity to models as the design process evolves.

In this regard, the quarter car ride model can be alternatively represented by these Modelica-based (physical) components for a mechanical system in Activate, as shown below:

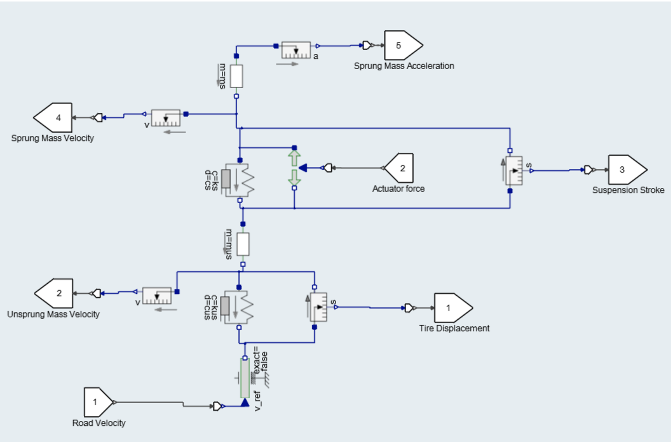

Figure 17: Modelica implementation of ride model

One benefit to this method in that this assembled model more closely represents the topology of the real system, as in Figure 17 from top to bottom, you can see a component for sprung mass, spring/damper for the passive suspension, actuator force, unsprung mass, and the a second spring/damper to represent the tire. Models are more easily built, modified and shared. Other components shown in the model in Figure 17 are added to measure desired outputs – displacements, velocities, accelerations, etc.

This physical model can be used to replace the portion of the model we had used to derive the equations for the Suspension Ride Model via the Matrix Expression block, and you can see it replaced via a green superblock (a hierarchical composite block) shown in the next figure.

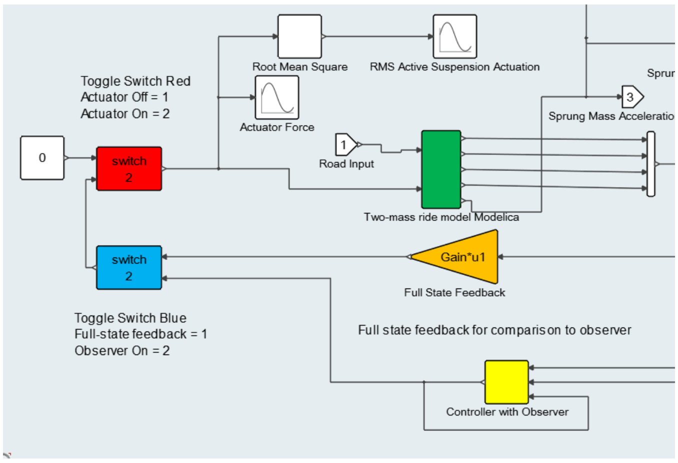

Figure 18: Superblock with Modelica components replaces traditional signal based block

This model can also be copied quickly and compared to the original model for verification, as both signal-based and Modelica (physical) blocks can be mixed in Activate.

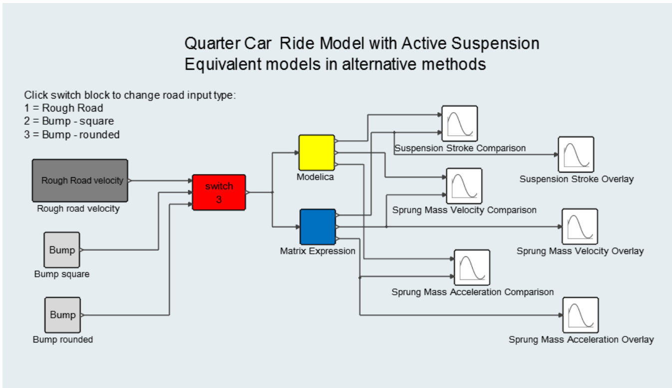

Figure 19: Signal-based and Physical-based modeling

Various domains are supported in Modelica-based libraries, including mechanical, electrical/electronic, hydraulics, thermal, and many others, both open-source and commercial.

This has advantages of creating and sharing models; as an open-source language, you can create and share your own libraries. Many libraries from various domains are available, and you can modify open source libraries to fit your needs.

VII. Multi-Domain System Integration

One important aspect of Model-Based Development is creating the entire system model, while sharing and communicating freely with all of the stakeholders that contribute to this model in the design process. This includes not only the early modeling for systems engineering and requirements management (e.g. with XLDyn), but also to detailed simulation modeling that may be combined to analyze each part the full system. Both Activate and MotionSolve support an open architecture to be able to achieve this.

A key component to enable full-system modeling is the Functional Mock-up Interface (FMI). FMI supports simulation coupling with any other tool that implements this open-source standard, including solidThinking Activate as well as almost 100 other tools at the time of this writing. Support for FMI available to HyperWorks customers through the Altair Partner Alliance, which includes CarSim, MapleSim, and DSH+ (www.fmi-standard.org). FMI enables tools to be shared and can even hide proprietary data in compiled models.

In addition, both Activate and MotionSolve support user subroutines. Activate supports code written in C, its own scripting language (HML/OML), and Modelica (code written in this standard). MotionSolve supports FORTRAN, C/C++, and Python.

Flexibility in providing interfaces with other tools is important to highlight with MotionSolve. Besides coupling with Activate, MotionSolve also co-simulates with AcuSolve for Computational Fluid Dynamics, with applications including tank sloshing, wind turbines, and active grill shutters.

MotionSolve includes other native features that implement control systems both linear and non-linear, including Laplace Transfer Functions, State-space equations, general non-linear differential equations and algebraic equations.

VIII. Increased Fidelity Plant Modeling

As more information is available to build higher-fidelity models, multibody system simulation can be used to improve the accuracy of the simulation. Detailed 3D multibody models can be used to both to validate the quarter-car ride model in the 1D simulation Activate as well as simulate the plant model directly via co-simulation, where Activate still models the rest of the mechatronics system for which is it better suited– sensors, actuators, and controls.

CarSim [16] is one solution for simulating vehicles for applications like control system analysis with models in Activate. CarSim is a software tool from Mechanical Simulation Corporation that helps create and analyze simplified vehicle models for real-time handling performance and Advanced Driver Assistance Systems. Starting with version 2016.1, Carsim supports the Functional Mock-up Interface (FMI; www.fmi-standard.org) co-simulation by including the capability to automatically generate a Functional Mock-up Unit (FMU) for co-simulation, which is a representation of the model and its interface for simulation.

FMI allows vehicle models from CarSim to be co-simulated with Activate for coupled system analysis. For example, we can model an anti-lock brake system or active damper in Activate with a vehicle model in CarSim. An example of development of an active suspension with CarSim can be found in [3].

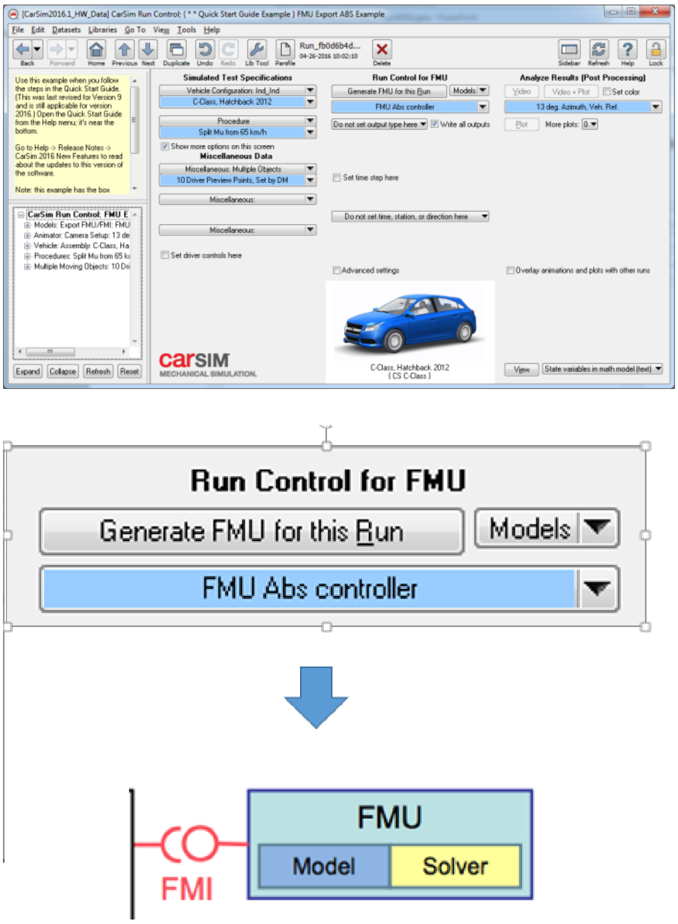

Figure 20: Functional Mock-up Interface in CarSim

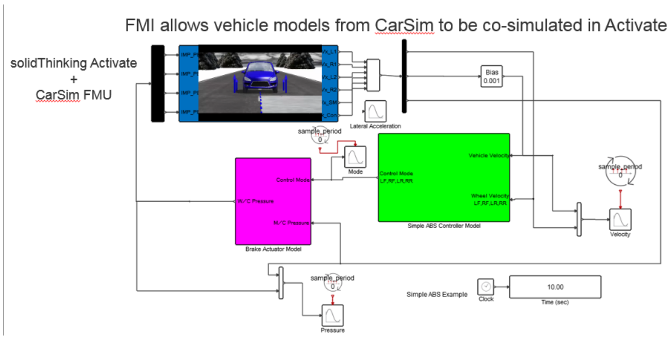

Figure 21: Activate ABS system co-simulation with CarSim

CarSim provides an effective representation of the vehicle suspension behavior which provides very fast simulation, but does not model detailed suspension components themselves, so some design tasks like extracting the loads on the suspension components, are not available.

This is where a 3D solution like MotionSolve is very effective. MotionSolve is Altair's very powerful, flexible, and robust multibody dynamics solver. MotionSolve models are created in various ways -- either from atomic components (e.g., rigid bodies, graphics, joints, forces, etc.), from importing CAD to automatically generate mass properties, or via libraries of mechanical systems, as with the vehicle library that contains various design templates for suspensions, vehicles, analyses, and reports.

MotionSolve supports a large range of components to create varying levels of fidelity in mechanical system models. This ranges from simple linear and non-linear spring dampers, to effects like component flexibility via Component Mode Synthesis for linear flexible or for non-linear flexibility based on the Absolute Nodal Coordinate Formulation (ANCF). Non-linear bushings, including frequency, amplitude and load-dependency are supported, and LuGre friction can be easily added to enhance the model accuracy. Detailed tire and road models to capture the effects of non-linear tire behavior are also integrated into MotionSolve.

In this process of using MotionSolve, once the mechanism is defined (for example, based on the CAD or FE components to define your parts), MotionSolve creates the equations of motion for you, so you have no need to derive these.

Now, Activate and MotionSolve can be co-simulated, where the vehicle controller can be tuned further, actuators verified, test repeatedly and safely, and minimize risk with the real prototype. In addition, more accurate system loads can be generated by including the detailed control system with the mechanism.

With these loads, you can perform durability studies with FEMFAT and nCode DesignLife in the Altair Partner Alliance, compute stress, and use loads for topology optimization in OptiStruct [8].

Packaging, and placement of sensors and the system vibration are all possible with the 3D multibody model.

MotionSolve models are parametric for easy design changes and design study/optimization with HyperStudy. MotionSolve results can also be used to generate the data required to populate CarSim models.

This next example describes the salient modeling for the co-simulation with MotionSolve and Activate for the active suspension implemented in a full-vehicle passenger car in MotionSolve.

Activate-MotionSolve Co-Simulation

The MotionSolve model is a 3D multibody model built within the pre-processing interface MotionView. MotionView supports a vehicle library to help quickly build and analyze half- and full-vehicle models, with vehicle specific components including tires, bushings, bump stops, rebound stops, etc., as well as many standard vehicle analyses and accompanying reports. The vehicle library is open source and may be customized.

Baseline Vehicle Description

The vehicle discussed here is a passenger car – a Ford Taurus based on an open-source model from National Highway Traffic Safety Administration [9]. The model is composed of both rigid and flexible bodies (based on Component Mode Synthesis, CMS), which contribute a large portion of the number of degrees of freedom in this model (1201 DOF). FTire (Cosin)[10] is used for the wheels which captures the deformation of the tire as it traverses obstacles.

Forces are added to the rear suspension (only) between the suspension and the body to apply the active suspension forces.

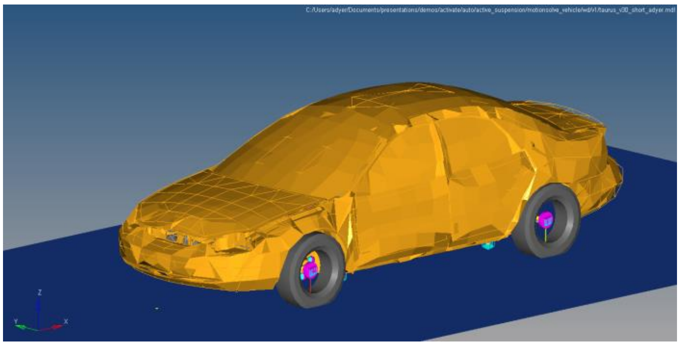

Figure 22: NHSTA Ford Taurus in MotionView (courtesy Jiamin Guan, Mike White, Altair)

Flexible Bodies include the vehicle body, front subframe, steering column, and front lower link of the rear quadlink suspension.

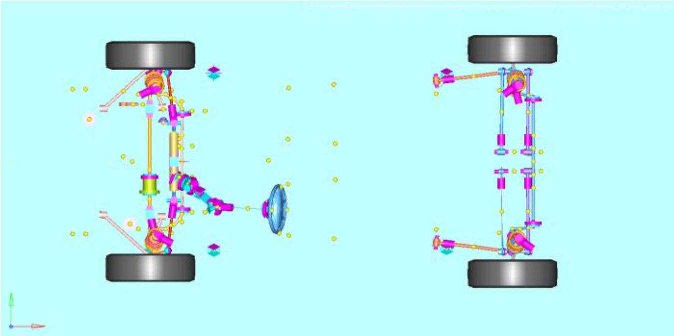

Figure 23: Front MacPherson (left) and Rear Quadlink Suspension (right)

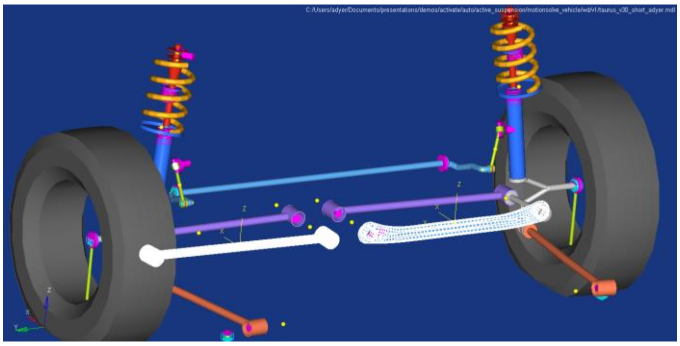

Figure 24: Rear Quadlink Suspension– rigid control arm (left in image) vs. flexible control arm (right)

Simulation Event – Road Bump

For the first event, the vehicle will travel at 20 miles per hour over a smooth road until it traverses a bump with both axles.

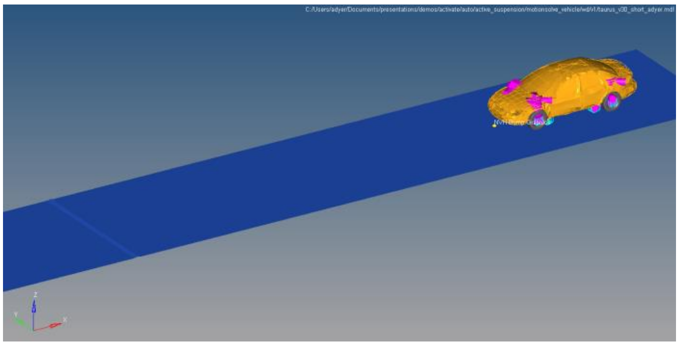

Figure 25: Highway bump Simulation Event in MotionSolve

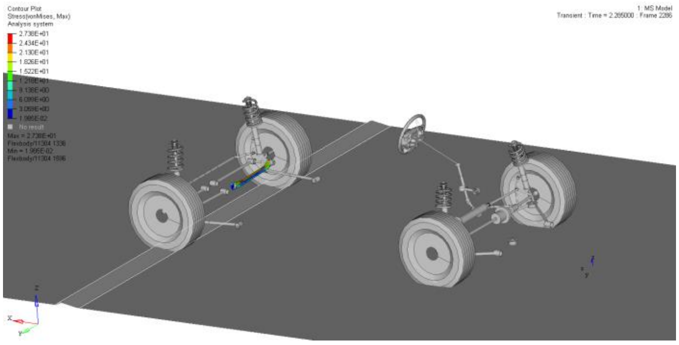

Figure 26: Simulation event ,body hidden

Modifications for Co-simulation with Activate and MotionSolve

In order to enable co-simulation, input and output runtime variables are created and used in the MotionSolve model. The interfacing input variables from Activate to MotionSolve are stored in an array. The MotionSolve model requires a solver array of type *Plant Input*, which has a list of solver (runtime) variables including:

- Active Suspension Force, left and right

These forces will be computed by the Activate model and populated during the co-simulation between Activate and MotionSolve.

Similarly, the MotionSolve model requires a solver array of type *Plant Output* to store outputs from MotionSolve to Activate, which has a list of solver (runtime) variables including:

- Suspension Stroke, Left and right
- Sprung Mass Velocity and Acceleration

MotionSolve will compute these values during the course of the co-simulation and send these over to Activate to compute the active suspension forces.

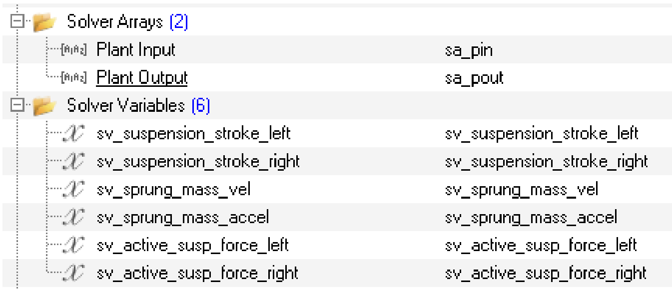

Figure 27: Interfacing entities – shown in the MotionSolve model

Finally, in order to use the actuator force computations from the Activate model, force entities are added in the MotionSolve model and reference the Activate suspension force variables shown above in order to apply them for co-simulation.

Activate Model Changes for Co-Simulation with MotionSolve

The plant model, previously represented by a *Matrix Expression* block or Modelica components, is now replaced by a *MS Signals* block which references the MotionSolve vehicle model for co-simulation. The inputs and outputs from this block are created automatically by Activate based on the *Plant Input* and *Plant Output* arrays found in the MotionSolve model.

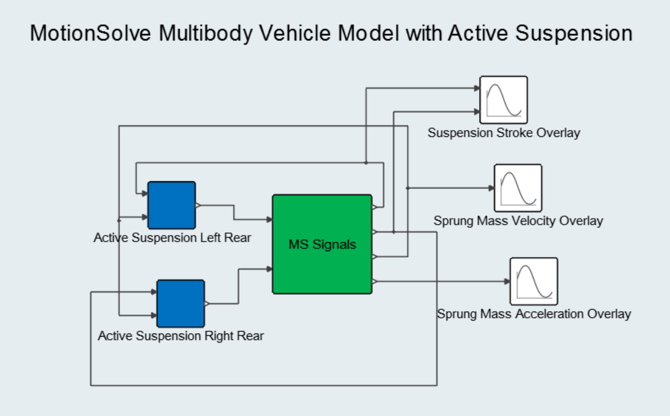

Figure 28: Activate active suspension model for co-simulation with MotionSolve

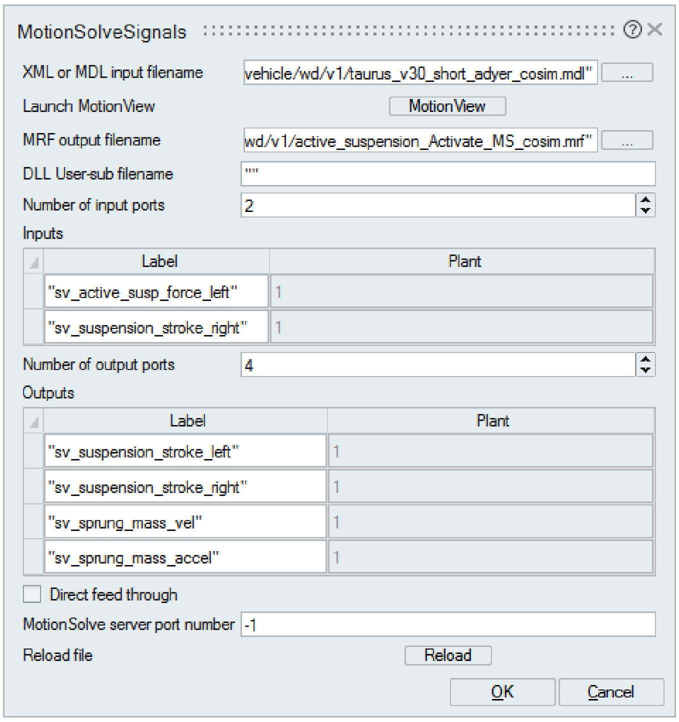

Figure 29 Dialog in Activate showing inputs/outputs

The active suspension force modeling in Activate must be rearranged to accommodate the new model topology, but the *Controller with Observer* block (yellow) remains unchanged – this computes the feedback gains based on the linear quarter-car ride model and uses an observer to estimate unmeasured states.

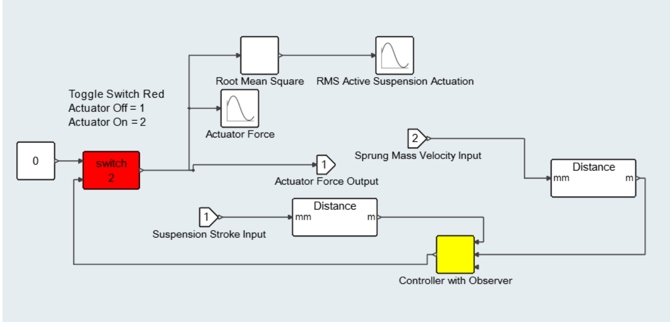

Figure 30: Active Suspension Actuator Force for Co-Simulation with MotionSolve

Co-Simulation Results

The active suspension at "moderate" setting in the Activate script reduces vehicle accelerations by approximately one-third (Figure 31), while it requires more suspension stroke to achieve this (Figure 32).

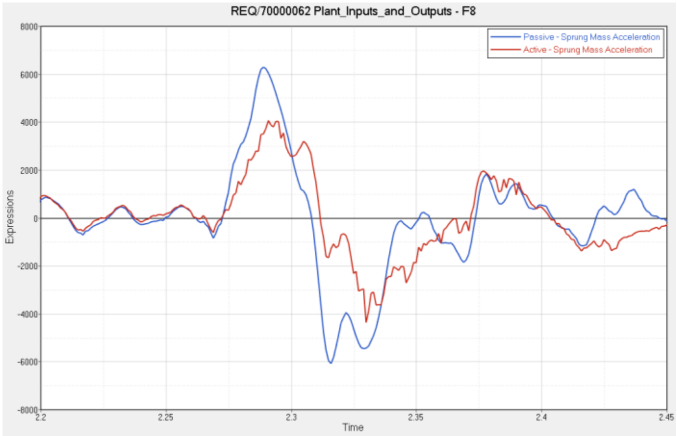

Figure 31: Sprung Mass Acceleration – Passive vs. Active

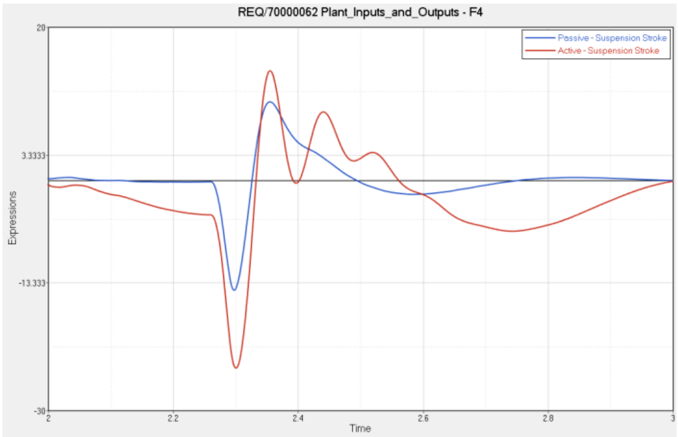

Figure 32: Suspension Stroke – Passive vs. Active

Peak stress in the rear front lower control arm reduces ~8% with the active suspension with minimal change in stress contours (Figures 33, 34). The flexible components do not make a significant change in the results in this test since the deformations are not large, but may be important for other road inputs that provide larger forces to the vehicle.

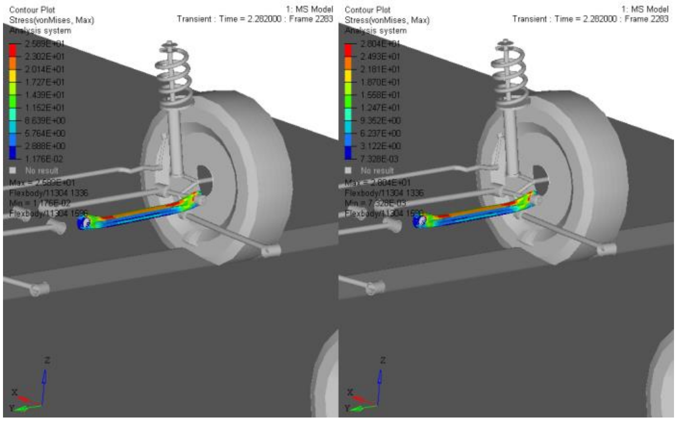

Figure 33: Stress on LCA - Active Left (Max 2.59– Passive Right Max 2.80)

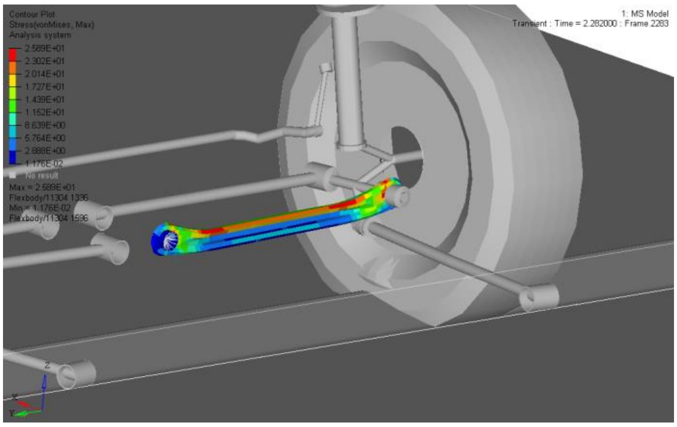

Figure 34: Stress on LCA – Active, Closer View

The body of this vehicle is also made flexible by CMS and the deformation contour can be reviewed as part of the simulation:

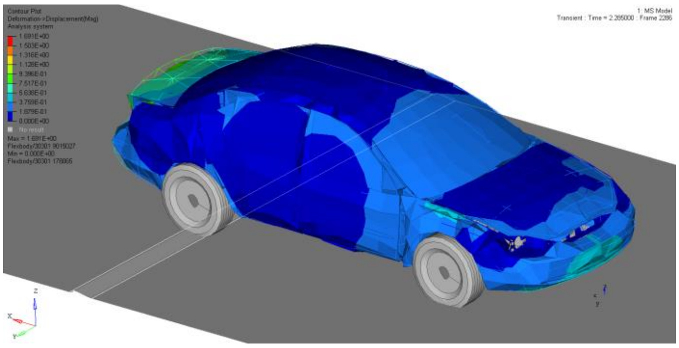

Figure 35: Deformation of Vehicle Body - Front

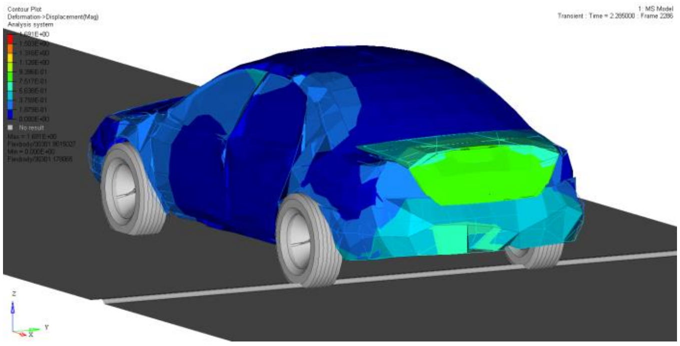

Figure 36: Deformation of Vehicle Body - Rear

Simulation Event - Pothole

For the second event, the vehicle will travel at 20 miles per hour over a smooth road while traversing a 200 mm deep pothole. The rest of the setup of the simulation is identical as for the prior bump event.

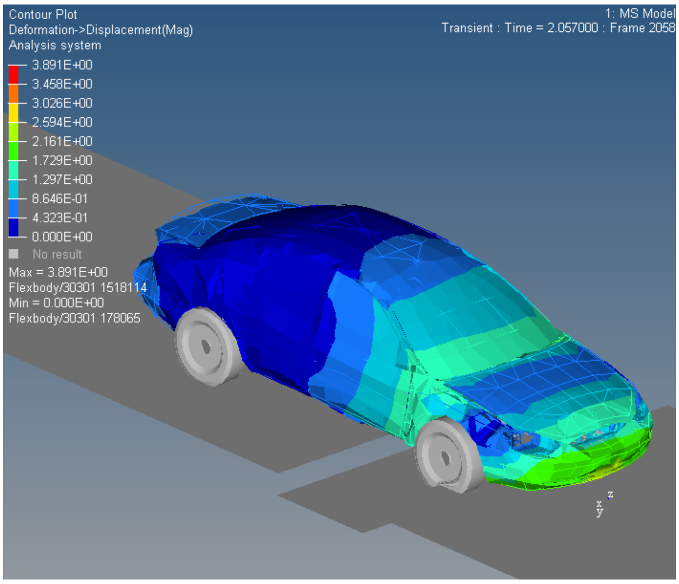

Figure 37: Pothole Event – Body Deformation

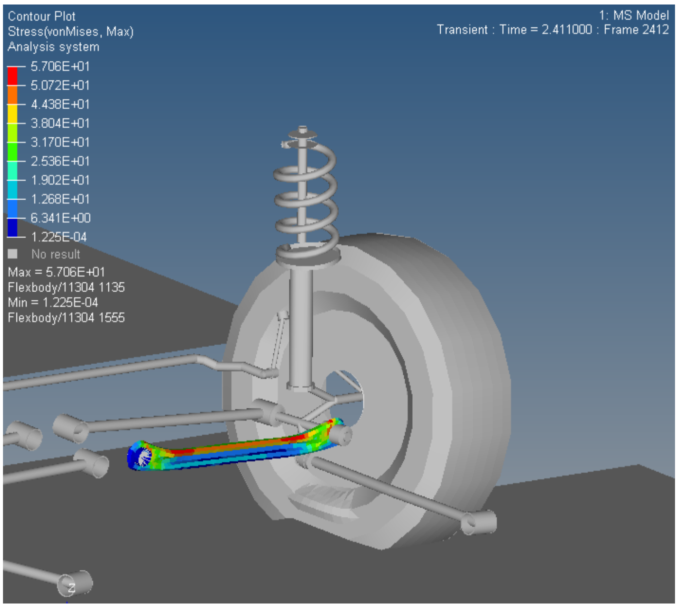

Figure 38: Stress in Rear LCA – Pothole Event

In this case, the loads to the vehicle are more significant and we can see that the flexible bodies have an effect on the behavior of the sprung mass acceleration response in Figure 39.

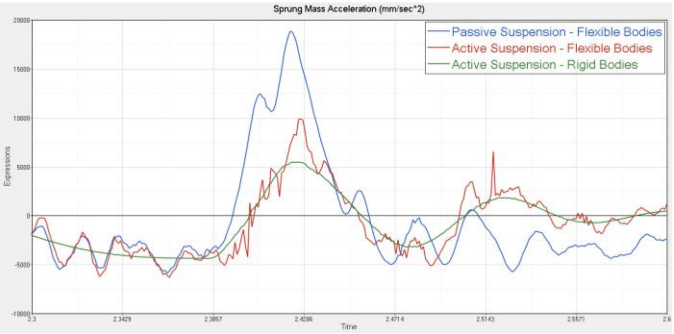

Figure 38: Sprung Mass Acceleration – Pothole Event

IX. Using 3D to Build 1D

Up to this point, we have not discussed the accuracy of the 1D model used in the early design phase. How do we know the simplified linear model is accurate given that suspension models are non-linear? Validation to reliable test data is required in order to remove assumptions that the models used are producing accurate results.

Physical tests have been historically used for performance validation. However, as more reliance on simulation is built up, the role shifts to that of final validation with a series of intermediate tests to build confidence in simulation [8].

MotionSolve 3D models may be employed to assist with validation processes. For the 1D active suspension model, we can consider how the quarter-car ride model captures the behavior compared to a full 3D model with detailed components and connections modeled. With either test data or a higher fidelity 3D model from MotionSolve, a parameter identification analysis may be done with a design study tool like HyperStudy in order to tune the parameters in the quarter-car model (mass, stiffness, damping) to match results [11].

We may also consider how well the simplified linear 1D model itself generally captures the system behavior, as the parameter identification for the 1D quarter car ride model may or may not achieve desirable accuracy. For example, the kinematic structure of suspension may have an effect on behavior not captured in the quarter-car model [12].

Other options for generating 1D models include using MotionSolve to generate reduced-order models by linearizing the model at an operating point [12], or non-linear models considered as in [13]. A design study tool like HyperStudy may also be used to build surrogate models from MotionSolve results based on curve fitting to derive equations for inclusion in Activate.

X. Conclusion

The Model-Based Development process can improve the design, delivery and support complex products like mechatronics systems. This paper has highlighted 1D and 3D system modeling tools from Altair that allow different levels of modeling fidelity to be employed, including some examples of active suspension based on LQR controller with an observer. Early in the design stage, simpler functional models are used where little detail is known to investigate design requirements and discover key design parameters that affect them.

As the design matures from these 1D models, 3D models can be built to realize the design and create more accurate plant models for controller design and tuning. Control systems may be modeled more naturally in a signal-based simulation tools, and Modelica and Functional-Mock-up Interface help to build and share multi-domain plant and actuator models for full system integration. MotionSolve multibody 3D models can be employed to validate and build 1D models. Integration of these tools and others enables the Model-Based Design process to bring together the multi-domain teams needed to build the next mechatronics (cyber-physical) systems.

References

1- http://www.nydailynews.com/autos/street-smarts/mazda-g-vectoring-control-article-1.2844144

2- http://www.tenneco.com/overview/rc_advanced_suspension/

3- "Modeling and Control of a Nonlinear Active Suspension Using Multi-Body Dynamics System Software" M. Fahezal Ismaila*, Y. M. Samb, S. Sudinb, K. Pengc, M. Khairi Aripind

4- "Design and Development of PID Controller-Based Active Suspension System for Automobiles, PID Controller Design Approaches - Theory, Tuning and Application to Frontier Areas", Senthilkumar Mouleeswaran (2012). Dr. Marialena Vagia (Ed.), ISBN: 978-953-51-0405-6, InTech, Available from: http://www.intechopen.com/books/pid-controller-design-approaches-theory-tuning-and-application-to-frontierareas/ design-and-development-of-active-suspension-system-using-pid-controller

5- PI/PISMC Control of Hydraulically Actuated Active Suspension System", Yahaya Md. Sam* and Khisbullah Hudha

6- "Entropy Based Classification of Road Profiles", Baslamisli, Onel

7- "Implementation of an Active Suspension and Preview Controller for Improved Ride Comfort.", Donahue, M. D. (2001) University of California at Berkeley: MSc. Theses.

8- http://insider.altairhyperworks.com/assess-component-strength-durability/

http://insider.altairhyperworks.com/analyzing-durability/

9- https://www.nhtsa.gov/crash-simulation-vehicle-models

10- https://www.cosin.eu/

11- "Sprung Mass Identification of Suspension in a Simplified Model" December 2016 · SAE Technical Papers, Gang TangJinning LiChao DingYunqing Zhang

12- "Reduced-order modelling and parameter estimation for a quarter-car suspension system", Proceedings of the Institution of Mechanical Engineers Part D Journal of Automobile Engineering 214(8):851-864 · September 2000

13- "Nonlinear Two-Dimensional Modeling of a McPherson Suspension for Kinematics and Dynamics Simulation", Mar 2012 · International Journal of Vehicle Design

14- “System lifecycle management: Initial approach for a sustainable product development process based on methods of model based systems engineering. In: Product Lifecycle Management for a Global Market.” Springer, pp. 287–300, Eigner,M., Dickopf,T., Apostolov,H., Schaefer,P., Faißt,K.-G., Keßler,A.,2014.

15- http://www.altairhyperworks.com/partner/xldyn

16- https://www.carsim.com/