An Interface to the FTire Tire Model

Volker Beuter
Kämmerer AG
Wettergasse 18, D-35037 Marburg
v.beuter@kaemmerer-group.com

Abstract

The FTire tire model [2] is a well established model for calculating tire forces and torques, especially if the internal dynamics of the tire and higher frequencies have to be considered. This tire model is available for most multi-body simulation programs like ADAMS, SimPack, RecurDyn and for MATLAB. But up to now it was not available in the Modelica world. As the FTire model is only available as binary libraries it could not be ported to a pure Modelica code, but an interface to the existing implementation had to be written. This paper describes the implementation of `FTire` package making FTire tires available in Modelica (so this paper is rather about interfacing than about tire modeling). The interface involves much more than just a set of Modelica wrappers to C functions: Most callable FTire functions are impure, some having only side effects, no return values. Some considerations have to be taken to ensure the FTire functions are called just often enough. Above this basic Modelica interface layer there is an embedding into the `Modelica.Mechanics.MultiBody` framework with some features beyond FTire itself like non-standard orientation or common definitions for several tires.

The moreover there are some related packages for special interests: The `FTireVDL` demonstrates the compatibility of the FTire interface to the `VehicleDynamics` package [5] from Modelon. Users of the `Visualization` package [4] from DLR-RM can apply the `FTireSimVis` package to animate multi-body models with FTire wheels with the SimVis program, exceeding the animation capabilities of Dymola.

Keywords: FTire tire model; interface; `MultiBody`*; visualization*

1 Introduction

Part of Kämmerer's involvement in the Eurosyslib project [13] was the development of interfaces to existing tire models. As the FTire libraries, headers etc. were publicly available from the Internet we decided to start with the FTire model. In order to prevent duplicate implementations there was an agreement with Modelon not to implement tire models that had already been implemented (as pure Modelica code) into their `VehicleDynamics` library.

There had been plans to implement an interface to the RMOD-K 7 tire model by Prof. Oertel, FH Brandenburg, Germany as well. Mr Oertel provided useful ideas concerning the anticipated problems caused by the fact that integrating a tire model equipped with its own integrator into a simulation environment always means co-simulation. On his suggestion we implemented a two masses oscillator model, where one oscillator is modeled in pure Modelica the other by an external C function solving its differential equation by means of a simple explicit Euler integrator. But finally Mr Oertel decided to implement a Modelica interface to RMOD-K 7 on its own some day. Meanwhile there was no time any more to start interfacing to another tire model so we concentrated on the interface to FTire.

2 The FTire Tire Model

The FTire tire model was developed by Prof. M. Gipser, FH Esslingen, Germany and is now distributed by his company COSIN scientific software, Munich [2].

The FTire tire model is available as a stand alone simulator and for most multi-body simulation programs.

The name "FTire" means *Flexible Ring Tire Model*. A tire is conceptually considered as a flexible ring of masses connected by springs in a certain pattern.

But this work is *not* intended to be an introduction into the FTire model. For the matters concerning here, it is only important that a tire mainly interacts with the rest of the model by means of a 3D force and a 3D

torque applied to a certain point in the model and depends on the motion states (position, orientation, velocity and angular velocity) of the wheel center. All the internal structure of the FTire model we can forget here (with the exception of the meaning of some optional additional outputs). We do not build the internal structure of the FTire Flexible Ring model by means of the `Modelica.Mechanics.MultiBody` package, but use the FTire libraries as a set of black boxes to calculate tire forces and some other auxiliary tasks.

It is also important to note that the Modelica interface to FTire does not fit into a decomposition framework of calculating a contact point first, than determining velocities and slip quantities, calculating normal forces and finally other force components [1]. Firstly FTire does not follow this approach; calculations are not based on a theoretical contact point. And even if FTire did so, it is only possible to embed calculations as a whole.

2.1 The FTire Data Files

All the parameters describing tire road interaction in the FTire tire model are in principle stored in two files:

- All tire parameters are stored in a tire property file (default extension `.tir`).
- The road geometry and properties are stored in a road data file (default extension `.rdf`). Some road data file types also refer to further files containing the road geometry.

This means most parameters concerning FTire tires will not directly be provided in the Modelica model but only those two files containing the parameters.

3 The COSIN Tire Interface (CTI) and the FTire Tools

There are several options to access the FTire tire force calculation from a third party software like Dymola in our case. The most favorite way is the *COSIN Tire Interface* (CTI) [3]. It can be used from both FORTRAN and C code, here we use the C versions of the routines. The CTI is available for several operating systems, but we are currently only concerned with the implementation for 32 Bit Windows. The CTI consists of a header file, a small static library (which will be linked to the `dymosim` executable) and a dynamic link library. The static library calls the FTire calculation routines in the `.dll`.

This is all that is needed to access FTire tires from Dymola in Modelica models. But there are some convenient programs from COSIN related to the FTire tire model also used by this interface. The first one, the animation program COSIN/graphics consists just of one executable file and directly comes with the CTI.

The moreover there is a whole suite of additional programs, collectively called the FTire tools. Here we especially need the FTire/editor: In principle a `.tir` file is a text file which can be edited with any text editor. But there are two reasons why it is advisable to use FTire/editor:

- FTire/editor provides all appropriate selections. Key words do not have to be memorized and erroneous inputs are reduced.
- In order to speed up simulations FTire does not directly use the input data provided by the user, but does some preprocessing. This preprocessed data is binary and is appended encoded as printable characters at the end of the tire data file. FTire/editor automatically executes a new preprocessing of these data if needed. This excludes simulations based on out-dated preprocessed data.

For FTire version 2010-4 the structure of the downloadable release has been changed. Now the FTire/tools are always included and its version fits to the CTI version. (Earlier versions had release numbers instead of release quarters. The version before 2010-4 was 2.11)

The functions in the CTI generally only provide access to *time dependent* inputs. Most of the *parameters* (in the Modelica sense, i.e. not changing during a simulation) can only be provided in a tire property or road data file. This determined the design of the Modelica interface to FTire: A Modelica model with FTire tires primary refers to `.tir` and `.rdf` files. Most tire parameters have to be changed in the tire property file by using FTire/editor. We do not write any `.tir` and `.rdf` with data entered into a Dymola GUI.

The CTI routines serve several purposes:

- There are data reading functions for the road data and tire property files.
- There are functions which actually do the calculation of the tire forces and torques depending on the current tire positions, velocity, orientation and angular velocity. In principally this is just one function, but there are variants returning the

forces at the (rotating) wheel center or the (non-rotating) hub or with a user defined road geometry. With this minimal set of routines tire simulations can be done already. But in some situations more routines are needed:

- A routine can apply reaction forces to the road part. This is often needed in case the road is not the absolute ground but some moving part, e.g. in test rigs.
- There are routines providing time-dependent inputs in case they are not parameters from the tire property files, e.g. an inflation pressure decreasing during simulation.
- Moreover there are reporting routines returning other calculated quantities than the tire forces and torques. These outputs are either directly returned or are written to special output files. These routines can be used for debugging or visualization purposes, but are also used for the embedding into the `MultiBody` framework, especially for visualization.

4 The Basic Modelica Interface to the CTI

The lowest level of the Modelica interface to FTire is a set of Modelica functions calling the corresponding CTI routines – directly or indirectly – as external C functions. There is one Modelica function for nearly every function in the CTI. The current version has been written for version 2010-4 of the CTI.

4.1 Calling CTI Routines as External C Functions

Most CTI functions can be directly called from Modelica as external functions. The name of each of the Modelica interface functions is like the corresponding CTI function, but in some cases the argument lists and return values are modified due to Modelica requirements:

- The input argument providing a state is also used as return value. This is done so that this function need not only be used in an empty function call but in an ordinary equation. This is useful, even if the returned variable is just a dummy not used any further: Being used in a real equation apparently ensures that the function is called often enough. (If used as an empty function call, no calls are performed because from Dymola's point of view nothing depends on it.)
- Some of the function calls are formally not time dependent (although they are to return time-varying quantities) and Dymola does not evaluate the function over time it seems when the original argument list is used: The returned array is constantly the zero array. By introducing a time dependent variable (time itself) the function is evaluated as expected.
- A similar situation occurs at CTI functions where an input in general is time dependent, but it may also be a constant. Therefore an additional optional input for the simulation time (not passed to the CTI function) is introduced in order ensure a time dependency to prevent Dymola from optimizing away the function call.

Some of the functions here do not have a return value. Their only purpose are their side effects, like turning on or off verbosity. Usually they are only needed once in the beginning of the simulation in order to set a certain mode. A call to them ought to be made as an empty function call in an `initial algorithm` section.

4.2 C Wrapper functions around CTI routines with function pointer arguments

In some CTI functions added in the last versions there are pointers to functions in their argument list. One example for such a function a a tire force calculation routine for a custom road model. Here a pointer to a road evaluating function is one function argument. (The road evaluation function returns among others the height z at a given location (x,y) on the road at a time t.)

Functional input arguments to functions are not supported in Modelica before version 3.2 and therefore not available at the time of writing this article. In order to support these CTI functions in the Modelica – FTire interface yet, C wrapper functions have been written for them. Such a wrapper has the same argument list like the CTI function in question, only the argument for passing the pointer to the user function (like the road evaluation function) is left out. The wrapper function only calls the CTI functions with the passed arguments and a pointer to an implementation of the user subroutine. The wrapper function is interfaced as an external C function in Modelica in the usual way. By this method also the CTI functions with function

pointers can be supported in the interface but currently the user functions have to be provided in C.

```
// wrapper around CTI function

#include "cti.h"
#include "UserRoadModel.c"

 void ComputeForcesWithExtRoad(
  int ti, double t, double* r,
  double* a, double* v, double* w, int mode,
  double* f, double* m, int* ier)
{
  ctiComputeForcesWithExtRoad(
     ti, t, r, a, v, w,
     UserRoadModel, mode,  f, m, ier);
}
```

In principle this is all what is needed to include FTire tires into Modelica models. There are some test models where motion states of the tire are calculated and directly provided to the force calculation functions, though.

Parallel processing versions of the CTI functions for calculating tire forces are not yet supported (cf. section 10.3).

Currently no external objects are used in interfacing to FTire, but needed actions like reading tire and road data files are done explicitly. Using a constructor function of an external object may be a cleaner way to do so.

5 The FTire Wheel Model in the `MultiBody` Framework

But of course this is no convenient modeling of a tire from the user's point of view. A wheel ought to be a component which can be connected to other `MultiBody` components just by a `connect` equation to a `Frame` connector. In principle this means the tire position and orientation from the `Frame` connector and its time derivatives are passed to the force calculation function. The force and torque variables of the frame connector in turn are equalled to the returned forces and torques. But there are some complicating factors:

5.1 Adjustment of the tire mass

When calculating tire forces FTire only considers the share of the tire mass which is *not* connected rigidly to the rim. The remaining share has to be considered explicitly by the simulation program when calculating inertia forces. This share of mass and inertia is reported by some CTI function. The moreover from the user's point of view when modeling a vehicle a wheel with a tire ought to be handled together as simple as possible. Therefore the main component of the `FTire` package is not a *tire* but a *wheel* model also containing a wheel mass. The share of the tire mass and inertia considered fixed to the rim (and therefore not accounted for by FTire) is automatically added to the user provided wheel mass.

5.2 Road Orientation

In FTire itself the orientation of the road is fixed: Global z is pointing up, global x is pointing forward. (This is called the **FTire initial frame**.) This orientation would impose a rather strict restriction on modeling vehicles with the `FTire` package. The moreover it does not match the default orientation used in the `MultiBody` package where global y is pointing up. It was a design goal that with the `FTire` package the road (and therefore the tire) can be oriented arbitrarily so that any existing model can be equipped with FTire tires without the need to re-orient the complete model.

A road orientation can be specified easily by two direction vectors `forward` and up. (The defaults `forward = {1, 0, 0}` and `up = {0, 0, 1}` mean that the usual FTire road orientation is used; `up = {0, 1, 0}` means the usual `MultiBody` orientation.) These direction vectors determine a rotation object from the world to the road frame. FTire expects the tire motion states (position, orientation, velocity and angular velocity) resolved in the FTire initial frame. The `Frame` connector provides the position and orientation resolved in the world frame and the angular velocity in the local frame (the translational velocity can be derived from the position). So first the angular velocity is resolved in the world frame by means of the orientation object of the connecting frame, then all quantities are resolved in the FTire initial frame using the orientation object from the world to the road frame. The force calculation function returns forces and torques resolved in the FTire initial frame. So here they are first resolved in the world frame and then in the local frame, because this is what has to be provided to the connecting `Frame`. All these calculations and transformations are capsulated into a `FTireForce` model.

5.3 Tire Orientation

In other multi body programs the orientation of joints and also tires is only determined by the orientation of the connecting frames. But in the `MultiBody` package the joint axes are defined by some direction vec-

tor(s) (resolved in the local frame). So we do not define the tire spin axis directly by the local frame (like other multi body programs do, taking the local z-axis) but by direction vectors `spin` and `tire_up`. (The vector `tire_up` is only needed for uniquely determining the wheel orientation by providing a reference for the rotation angle.) The defaults `spin = {0, 1, 0}` and `tire_up = {0, 0, 1}` mean that the wheel rotates around its local y-axis. When the connector frame is not rotated relative to the world frame the tire can roll along the global x-axis. In case of the usual `MultiBody` orientation (global y is pointing upwards, i.e. road orientation vector `up = {0, 1, 0}`), `spin = {0, 0, 1}` ought to be used.

Internally the tire orientation is modeled by a `FixedRotation` component. With the default tire direction vectors its orientation is the null rotation.

5.4 Common Tire and Road Properties

In FTire itself all tire instances in a model are completely independent, i.e. they all have their own tire property and road data file definitions. But except for some test rigs and special situations all wheels of a vehicle will run on the same road. This road will have the same orientation for all tires and they in turn will have the same orientation. The moreover in many cases there will be the same tire for all instances, i.e. the same tire property file.

In order to prevent the need for providing these property files and the orientation vectors four or more times for a vehicle a common properties model using the `inner / outer` mechanism has been introduced: Each tire model accesses a common property component as `outer` object. In case a top level model containing an FTire wheel model does not contain such a common properties component a default component is used. The values from a common properties object can be overwritten by a wheel model: In case a tire property or road data file string is provided (i.e. it is not empty) or direction vectors are specified (i.e. they are not zero vectors) they are used overwriting the values from the common properties object. In this way it is possible to make exceptions, e.g. providing road data and tire property file by way of the common property object, but using a different tire property file for one wheel in order to model a defect tire.

In case all property files and direction vectors are directly provided at all wheels the common property object is not used at all. When there is only one wheel in the model (like in a tire test rig) this is the easier method.

5.5 Road Part Model

In FTire the road does not have to be associated to the absolute ground but tire contact may be calculated towards some movable road part. The `FTire` interface package supports this feature by a model `RoadPart`. Like the wheel models it possesses a usual `MultiBody Frame` connector. The road part model consists of a `Body` component to model its mass and inertia, a road surface visualizer object and a `RoadForce` component.

The `RoadForce` model queries the road part motion quantities (position, velocity, orientation and angular velocity) and provides it to the FTire kernel by means of a CTI function. When tire forces for the associated wheel are calculated this not done based on the motion relative to ground but on the relative motion to its road part. Therefore each road part needs to have a unique ID corresponding to a wheel. (When the same road part is to be used for more than one wheel a `Body` component (and a road visualizer if needed) can be used together with one `RoadForce` component for each of the wheels running on that part.) The reaction forces from the tire contact are applied to the road part. (Because of the tire inertia this forces do not equal the forces to the wheel with opposite signs.)

Here again it has to be considered that FTire itself has a fixed orientation of the road whereas the Modelica model using the `FTire` interface package may use some other orientation. Thus the motion states of the road part resolved in the world frame are transformed into the FTire initial frame before passing to the FTire kernel. Conversely the CTI function returns the reaction forces also in the FTire initial frame. They are first transformed to the world frame before applying them to the frame connector of the `RoadForce` component resolved in this frame.

6 Tire and Road Visualization

For a tire model based on the theoretical contact point concept a simple cylinder (with some mark to indicate rotation) is all what is needed for tire visualization seen from a technical point of view. The unloaded radius and the tire width (and possibly the radius of the wheel rim to indicate too deep penetration of the tire into the road) is all that is needed. Everything else is a matter of nice animations, but does not bring much new insights into the tire behavior.

Things are different when the internal dynamics of the tire is also considered and the tire (and possibly also road) deformation during simulation is also avail-

able. In this case the animation of the deformed tire is a good plausibility check on the tire simulation.

The CTI provides tire shape information by means of the function `ctiPutNodePositions`. The name may be a bit misleading here. It does *not* deliver the position of one of the internal belt elements constituting the flexible ring modeling the tire. Instead, for any point on the tire surface uniquely characterized by an circumferential angle (scaled to the range $[0,1]$) and a value for the position along the cross section (also in the range $[0,1]$) the 3D position of that point is calculated. The position is returned in several coordinate systems. We only use the position in the FTire initial frame.

When the `ctiPutNodePositions` function is evaluated on a regular grid over $[0,1] \times [0,1]$ this constitutes a representation of the tire surface. It is important to note that the returned positions are variables, not just parameters. The calculated grid is a representation of the dynamic tire deformations during a simulation. Here it has to be considered that the FTire initial frame is not necessarily the `MultiBody` world frame, so the surface coordinates have to be resolved in the world frame using the rotation object from the world frame to the FTire initial frame.

Things are similar for the road shape representation: There is a CTI function `ctiEvaluateRoadHeight` calculating the current height of a location (x,y) of the road. Again with a grid of equally spaced evaluation points on a range $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$ the road surface is determined. Usually the surface of a road is static, it is calculated only at initialization time. By a Boolean parameter it can be specified that the road surface is (possibly) time-varying and has to be evaluated at every simulation step. Interestingly the formulations for the surface grid values are the same except for that in the time-varying case the simulation time variable `time` is passed to the road height evaluation function, in the default case the constant 0. Dymola infers that in the second case all the inputs to this function are constants or parameters and therefore it does not need to be evaluated again during continuous integration.

There are two ways these surfaces can be visualized: The model is animated in the Dymola animation window, tire and road are visualized by surface visualizers from Kämmerer's `Visualizers` package. Alternatively the `Visualization` package with the external viewer SimVis from DLR-RM can be used like described in the last subsection of this section.

6.1 Kämmerer's `Visualizers` package

The Modelica Standard Library (MSL) contains visualizer models for animation of geometrical primitives like boxes, cylinders and spheres in Dymola. All these visualizers are based on the model `MB.Visualizers.Advanced.Shape` (respectively `ModelicaServices.Animation.Shape` in Modelica 3.1). Unfortunately this model is not able to animate so called polylines (connected sequences of straight lines) or surfaces. But in the Dymola distribution with the MSL come some tiny Modelica models for visualizers, also for polylines and surfaces. From these ideas a package for visualization of (possibly) time-varying geometric primitives has been developed. In particular it contains several visualizer models for surfaces[1] and a simplified version of the standard visualizer form the MSL applicable in some situations.

The surface geometry is defined by a matrix of 3D positions, i.e. by an `input SI.Position[3] grid[m, n]` where `m` and `n` are Integer parameters. The color and reflectance of the surface can be provided by inputs like at the usual MSL visualizer for geometrical primitives. (This means there is an overall color for the surface. Different colors for a surface can only be realized by splitting up a surface into several sub-surfaces.)

The most simple surface model is `WorldSurface` where the `grid` input defines the surface in world coordinates. The model `Surface` additionally has a position input `r` and an orientation object `R` constituting a frame. Here the surface is defined in this frame. (Compared to the usual MSL `Shape` model here we abstained from additionally defining direction vectors defined in that frame to further orient the surface.) The model `FixedSurface` has a frame connector. It uses a protected `Surface` component as a sub-model where the position and orientation inputs are simply the ones from the enclosing `FixedSurface` model. The sub-model is conditionally disabled dependent on an `animation` flag. This is the same approach as with the MSL `FixedShape` model. But there is also a version `FixedSurface2` where there is no `Surface` sub-model, but the model is extended from it, simply modifying the position and orientation inputs. In using this method there is no possibility to switch off animation by means of an `animation` flag. But on the

[1] A similar visualizer is now available in the latest version 3.2 of the MSL with Dymola specific implementation in the `ModelicaServices` package to be used in Dymola 7.5. In contrast Kämmerer's `Visualizers` package can also be used even with the MSL 2.2.2 and Dymola 6.1!

other hand if this is not needed, this prevents a duplication of model parameter and variables to two model levels.

The polyline model family is implemented in an analogous way. There is also an alternative `FixedShape` model extended from the `Shape` model instead of using it as a subcomponent.

The `Visualizers` package is used in the `FTire` package for road and tire visualization.

6.2 Tire and Road Visualization with the `Visualizers` package

The road visualizer model consists of a `FixedSurface2` component where the surface is the grid with the calculated heights. This surface is firstly oriented in the FTire initial frame. In order to account for the correct road orientation the surface visualizer is connected to the outside connector via a `FixedRotation` component with the orientation calculated from the road orientation vectors and zero translation.

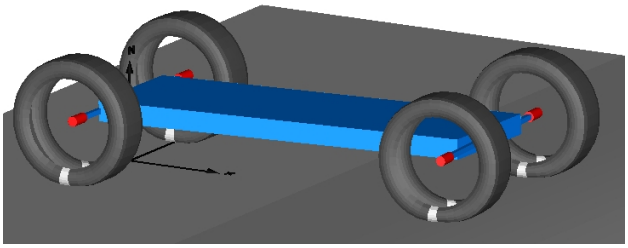

Figure 1: Tire and road visualization using Kämmerer's `Visualizers` package

The common properties and the `RoadPart` models already contain a `RoadVisualizer` object. Therefore a user only has to include a road visualizer into a model explicitly when none of these models is used, i.e. if the road is belonging to the absolute ground and tire property and road data files are specified at the wheel components.

Using this dynamic tire visualization directly within Dymola supersedes to use a separate visualization of the tire with the COSIN/graphics program. Although calculating the dynamic tire shape costs CPU time it is still faster than separate animation.

6.3 Using the `Visualization` Package instead

DLR-RM has implemented a commercial Modelica package "`Visualization`" [4] for advanced visualization of multi body models. It does not use the built-in visualization capacities of a Modelica environment

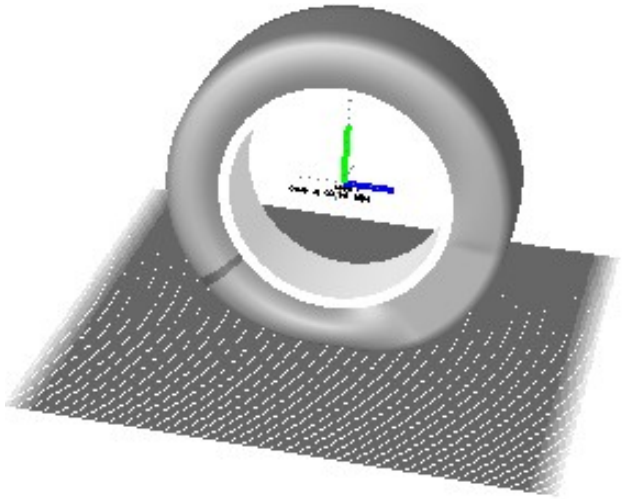

Figure 2: Tire animated with FTire animation tool COSIN/graphics

like Dymola but animates a simulation in a separate program called SimVis. It provides a lot of additional animation features like atmospheric effects, lights and cameras [7]. In particular it also contains visualizer for arbitrary, time-varying surfaces. So this package seemed to be suitable for tire animations.

In order to investigate this a separate package `FTireSimVis` (using the `FTire` package) has been implemented. Currently it contains independent tire and road visualizer models based on the `Visualization` package and an extended wheel model using this tire visualizer and also an extended version of the common properties object with the road visualizer for SimVis. In some later release of the `FTire` and `FTireSimVis` packages there ought to be partial visualizer models for tire and road in the base package. The tire and road visualizers in the `FTire` and `FTireSimVis` packages will be extended from these partial visualizers. All models in the `FTire` containing visualizers will declare them as `replaceable`. The `FTireSimVis` versions of the models will only have to `redeclare` the visualizers.

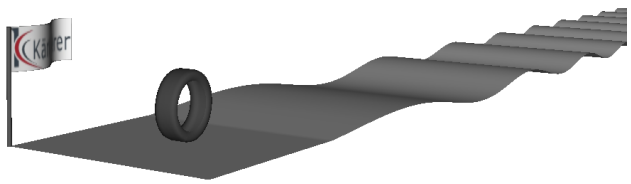

Figure 3: Tire and road animated with SimVis viewer

7 Embedding into the Vehicle Dynamics Library from Modelon

Modelon developed and maintains a commercial package `VehicleDynamics`, also called VDL (for vehicle dynamics library) [5]. It provides a set of tire models completely implemented in Modelica. But it did not include any tire models which involve interfacing to external libraries.[2] So it is quite a natural question, if the Modelica FTire interface discussed here fits into the VDL framework.

To answer this question a new package `FTireVDL` has been created, because it is not relevant to the basic interface and the `FTire` package has to stay independent from the VDL. The `FTireVDL` package mirrors the structure of sub-packages as far as needed. There are sub-packages for several layers of vehicle parts, like vehicle, chassis and wheel.

The implementation of the FTire wheel for the VDL package extending from the partial wheel interface model `Conventional` is quite simple: An FTire wheel component from the basic `FTire` package had to be connected to the outside connector. As the VDL uses another connector (containing besides the usual `MultiBody frame` also a rotational `flange` to describe the rotation) a `MultiBodyMount` component had to be placed in between. Additionally most of the summary variables could be filled by the TYDEX output variables of the FTire wheel. When some quantity is not available the summary variable is constantly zero.

As the FTire wheel has its own visualization, the tire visualizer from the VDL is not needed. Some of its nice features, like showing contact of the tire to the ground by changing the color of the tire or force vectors (but in this case located in the wheel center, as there is no contact point at FTire) could be added easily in some further version.

Regarding test models there are versions of the `"GettingStarted"` Sedan car. To this purpose there is a version of its chassis equipped with FTire wheels and also a vehicle model version using this chassis in turn; all the other components are used like in the VDL `SedanTEKBakker` vehicle. (So if there, in the chassis and in the `GettingStarted` experiment model the components had been redeclared replaceable, no new FTire model versions for the chassis and the vehicle had been required and we could simple extend the experiment model and do the required redeclarations.) There is also a simple tire test rig model. Both experiments are available in two versions: In one version the required tire property and road data files are directly provided in the wheel component. An explicit road visualizer component is used to animate the road. The other variant uses an FTire common properties object to specify tire and road. Here we can use the road visualizer contained in that object.

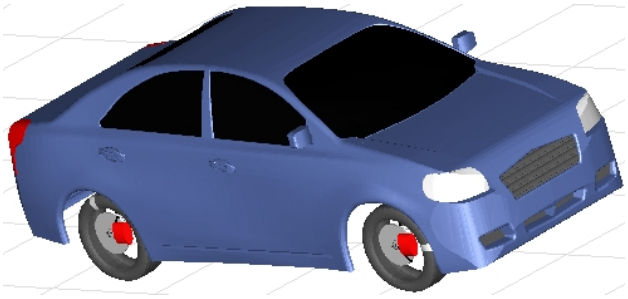

Figure 4: Sedan car from the VDL equipped with `FTire` wheels

A remark on the used integrator method: Usually FTire preferably works when the solver of the calling simulation tool uses a fixed step size integrator. This also holds for the `FTire` package in Dymola: We had the best results with the RKFix4 Integrator. Surprisingly when running the examples in the `FTireVDL` package the simulation fails due to numerical instabilities of FTire itself. (The corresponding VDL models with one of its own tire models works fine with a fixed step size integrator.) This phenomenon deserves further investigation.

In contrast to the `FTire` package the current `FTireVDL` package is not to be seen as a package ready to use, but as a demonstration that the integration of FTire tires into the VDL framework using Kämmerer's `FTire` is possible. As the VDL is encrypted and some of its models are not readable on the text layer, a full integration will only be possible in close cooperation with Modelon. The VDL contains a road builder. It ought to be enlarged with the facility to create road data files.

8 The FTire Interface in other Modelica Environments

The `FTire` package and all related packages have been developed and tested with several Dymola releases. In order to use them with other Modelica environments there are several central demands to that environments. With decreasing importance these are:

[2] Meanwhile there is an interface to the Delft tire model [6].

- Ability to call C functions by way of the `external` keyword.

  Without this feature no interfacing is possible at all. If at least it is possible to call FORTRAN subroutines the interface can be changed so that the FORTRAN version of the CTI is used instead of the C version.

- (At least partial) support of the `Modelica.Mechanics.MultiBody` package. The FTire interface packages themselves do not contain kinematic loops, so this is no demand to the Modelica environment, but of course this would impose severe limitations to the vehicle models to use the tires.

  Without support of the `MultiBody` package it is still possible to use the primary interface functions to the CTI functions. It could be possible to build some simple vehicle models by means of equations and to use FTire tires by directly calling these interface functions. For some special applications like longitudinal dynamics also an embedding of this basic interface into the `Modelica.Mechanics.Translational` package is conceivable.

- Ability to visualize surfaces like done in the `Visualizers` package. This means support of the undocumented "magic" functions `PackShape` and `PackMaterial` and the form numbers causing subsequent output variables having a meaning for animation (like used in earlier versions of the MSL, now moved to the Dymola version of the `ModelicaServices` package since MSL 3.1).

  If a Modelica environment at least recognizes these functions (with some idle dummy implementation) the visualizers should not cause problems, but the models stay without tire or road visualization. If the surface visualizer model does not even compile it can be replaced by a dummy implementation easily.

9 Future Work Using upcoming Modelica Features

Since about version CTI 2.9 FTire provides the feature of using own tire or road models to incorporate into FTire [3]. The CTI subroutines for doing so contain pointers to functions in their argument lists. The currently used method of using C wrapper functions (cf. section 4.2) has the drawback that the user subroutines have to be provided as C functions too.

With the upcoming feature of using functions as input argument to functions [8] it will be possible to write these user subroutines directly as Modelica functions and no C code will be needed any more.

10 Some Remarks on Modelica

But even with Modelica 3.2 there are still some issues and limitations for further development:

10.1 No Way of passing Information from Model to Function

Writing a *function* to evaluate the current height at a position on a road is appropriate for some special solid road geometry. But when it comes to write an elaborate soft soil road model in Modelica it is quite natural to use differential equations for doing so. This cannot be done in a function but only in a *model*. On the other hand a *function* has to be passed to the CTI function. Due to the requirement that Modelica functions are always pure (i.e. functions in the mathematical sense) there is no way of passing information from a model to a function. So even with Modelica 3.2 it will not be possible to use a user road model defined by a Modelica model in the CTI function for force calculation with user defined road.

10.2 Passing information for Co-Simulation

One aspect of the Modelica philosophy is that the user ought to build physically sound models in the Modelica equations, perhaps provide function derivations or inverses by means of annotations, but let the integrator do his business in solving these equations.

This is quite ok as long as pure Modelica is concerned. But this concept becomes questionable when there is a co-simulation: FTire does not simply calculate forces and torques based on the wheel center motion states as input quantities, but does its own integration (of its own internal model of the tire as a system of elements connected by springs forming the flexible ring).

When FTire is called from a variable step size integrator it will happen that an integration step fails and it will be repeated with some smaller time step. This means time goes back for the force calculation function. The FTire force calculation functions provide an input `MODE` for the "job control", i.e. here you can

inform FTire if the wheel input states are already accepted by the calling integrator or if they are not yet accepted. Currently we are only left with the option to call the function regardless if the the states are accepted or not. The `FTire` package works well with fixed step size integrators (like it is recommend for FTire usage anyway). But it can be expected that the performance with variable step size integrators could be improved if there were any means to communicate such information from the calling integrator to the FTire integrator somehow.

10.3 Multithreading

As far as we know currently no Modelica environment supports multi-threading, i.e. the parallel execution of parts of code and there are no means in Modelica to organize such parallelization. As long as just pure Modelica is concerned it might be argued that is a matter of the Modelica environment to see if parallelization is possible based on the equations (possibly in the flattened model). At least regarding external functions this approach seams to be not feasible: The environment needs to know if a function is thread save (can be parallelized without the risk of wrong results e.g. due to overwritten memory) and on the other hand if it is worth to parallelize the execution because it will take considerable time.

The latest versions of the CTI also provide versions of the force calculating functions for multi-core architectures. The forces of each tire instance are calculated by a separate thread. By this means the forces at all tires of a vehicle are calculated in parallel. The force calculation routine is split in two: There is one routine to pass the wheel center states and to trigger the force calculations in one separate thread for each instance. A second function fetches the results of the instances. In an *algorithmic* programming style for the simulation of the complete vehicle both functions can be used easily: In the algorithm for a single integrator iteration after the (preliminary) calculation of the wheel center states for all wheels there is first one loop triggering the force calculation for *all* tires and second another loop to fetch the results once the calculation is finished. In contrast in the Modelica setting, where external functions are called in some equation based model it seems likely that the force fetching function for the first instance is called before the calculation triggering function for the other instances, i.e. in fact there is no parallelization. This question needs further investigation. Probably there will arise the need to tell the Modelica environment (by way of some annotation) that such a pair of external function belongs together, that the second one delivers the results of the calculations triggered by the first one. [3]

10.4 Automatic `inner` components

The common properties model also contains the road visualizer as a sub-component. It can be turned on and off by some Boolean parameter. Using this instance of the road visualizer is sensible in all circumstances except for the rare case that the road is not fixed to the ground but to some other part (i.e. a `RoadPart` component is used). Considering this the default value for the road animation ought to be `true`.

But it is a (generally nice) feature of the Modelica `inner / outer` mechanism that it is *not* required to define an `inner` component explicitly in a top level model using components referencing on this model as an `outer` component: An implicit component is used in such a case. In the case of the common properties model here a component would have to be defined explicitly just to turn off road animation when it is not desired. To prevent this the default value for the road animation is `false`.

What were useful here is some function to determine *in* an `inner` model if an instance of it is the automatically created or an explicit one. With this feature it would be possible to set the value of the road animation parameter to `true` only if the component is explicitly defined.

11 Conclusions

The `FTire` package makes the FTire tire model available for the Modelica world. There are even some features supported like dynamic tire and road shape animation not yet available at the embedding in other multi-body programs. On the other hand there are many issues for further improvements like supporting custom road and tire models in a convenient way or parallelization of the tire force calculation for several tires. The usability in other Modelica environments has to be tested.

The original plan to develop also interfaces to other tire models was not addressed anymore within the Eurosyslib project due to lack of time but are subject to further work. Although any interface to the FTire tire

[3] Recently there have been considerations on co-simulation in a general frame in the Modelisar project [9]. It is not yet clear if the current CTI implementation is compatible to this FMI approach and in case it is, if this method of co-simulation is appropriate to interface a set of external function to Modelica.

model does not fit into the decomposition framework of a contact point based tire model [1] some other common framework for interfacing to other tire model is imaginable: Any tire model providing some API will have comparable tasks like reading data files or doing the force calculation. So above the CTI specific interface level a new tire model independent interface level may be established branching to the CTI routines or the counter-parts of some other tire model. In this case it ought to be possible to implement the embedding into the `MultiBody` framework independently from the FTire model but only accessing the tire model intermediate interface level.

12 Acknowledgements

The `FTire` (FTire interface for Dymola), `FTireVDL` (FTire tires in the `VehicleDynamics` package framework) and the `FTireSimVis` (FTire tires and roads visualized with the SimVis program) packages and the used geometry visualization package `Visualizers` have been developed as part of the ITEA2 **Eurosyslib** project (WP 8.5).

Earlier versions of the `FTire` interface package have been intensively tested in a diploma thesis [10] on a model of Kämmerer's side car vehicle "mython" [11] and at Dassault Aviation [12] with aircraft models. The package benefitted much from the reported bugs and suggestions for improvements.

References

[1] ANDRES, Markus, ZIMMER, Dirk and CELLIER, François E.: Object-Oriented Decomposition of Tire Characteristics Based on Semi-Empirical Models. Proceedings of the 7^{th} Modelica Conference, Como, Italy, 2009

[2] The FTire homepage: `www.cosin.eu/prod_FTire`

[3] The CTI reference document: `www.cosin.eu/res/cti.pdf`

[4] The `Visualization` package product page: `www.bausch-gall.de/vi1.htm`

[5] The `VehicleDynamics` package product flyer: `www.modelon.se/DATAUPLOAD/File/Flyer_dymola_VDL_Car.pdf`

[6] DRENTH, Edo, GÄFVERT, Magnus: Modelica Delft-Tyre Interface. Proceedings of the 8^{th} Modelica Conference, Dresden, Germany, 2011

[7] BELLMANN, Tobias: Interactive Simulations and Advanced Visualizazion with Modelica. Proceedings of the 7^{th} Modelica Conference, Como, Italy, 2009

[8] Modelica 3.2 Language Specification, 12.4.2, Modelica Association, March 2010, `www.modelica.org/documents/ ModelicaSpec32.pdf`

[9] Functional Mock-Up Interface for Co-Simulation. Modelisar (07006). Document version 1.0, October 12^{th}, 2010

[10] ZAPF, Stefan: Aufbau und Validierung des Gesamtfahrzeug-Mehrkörpersimulationsmodells mit einem Hochfrequenzreifenmodell im Programmsystem Dymola, diploma thesis, Amberg, Germany, 2009.

[11] The "mython" at Kämmerer's homepage: `www.kaemmerer-group.com/mython/`

[12] THOMAS, Eric and LAPEYRE, Arnaud: DTG 121069 FTire Model-Evaluation Report *(unpublished)*, 2010

[13] `www.itea2.org/public/project_leaflets/EUROSYSLIB_profile_oct-07.pdf`