A Novel Proposal on how to Parameterize Models in Dymola Utilizing External Files under Consideration of a Subsequent Model Export using the Functional Mock-Up Interface

Thomas Schmitt[1] Markus Andres[1] Stephan Ziegler[1] Stephan Diehl[1]

[1]3DS GmbH, Germany, {thomas.schmitt, markus.andres, stephan.ziegler, stephan.diehl}@3ds.com

Abstract

This paper introduces a novel proposal on how to parameterize models with data, taking a subsequent model export into account, using e.g. the Functional Mock-Up Interface (FMI). During model export parameters are either assigned with values directly or they are linked to external data-files. If the design of models or libraries is done without considering how data is handled in an exported model, those concepts are often mixed, resulting in an inconsistent data management which is cumbersome or even error prone for the user.

Keywords: *model parameterization, data files, model export, functional mock-up interface, FMI, FMU*

1 Introduction

In 2011 a new model export standard was released by Modelisar: The Functional Mock-Up Interface (FMI) (Blochwitz et al., 2011), (Association, 2015). FMI was immediately accepted and promoted by many tool vendors and Original Equipment Manufacturers (OEMs). Unfortunately, there are a couple of known pitfalls related to the export of models (Bertsch et al., 2014). One especially relevant for an a-causal modeling language like Modelica is related to the change of an a-causal to a causal model. This required adaption can cause higher index problems and/or algebraic loops (Blochwitz et al., 2012). However, this paper shall deal with a topic not yet intensely discussed by the Modelica community but of central importance for industrial use cases: Parameterization of models, considering a subsequent model export and the handling of data in this case.

From our experience library developers should put considerable effort into proper model parameterization when it comes to a subsequent model export. Fortunately, the Modelica language offers several possibilities to parameterize a model, i.e. to assign parameters with values.

In Modelica it is common to specify parameter values in records. The parameterization can either be done by coding values into the record with the Modelica environment or by reading the data from an external file for which the format can vary. Both solutions have their pros and cons and are absolutely justifiable. (Köhler and Banerjee, 2005) shows a case where custom text-based files are used as parameter files, which can be accessed by multiple simulation environments. On the other hand, Modelica-based parameter files (records) are usually more convenient for the user, especially for beginners as they can be edited directly in the Modelica environment.

1.1 Use-Cases of Exported Models

In this paper we will focus on the export of FMUs from Dymola[1], discussing different use-cases in which the FMU is utilized after the export. Depending on the particular use-case the model export underlies different requirements regarding convenient data handling. To our experience the following cases cover most of the applications used in industry today.

1. Parameter values are stored inside the FMU.
2. Parameters are stored in an external data-file. The FMU reads the parameter values during initialization of the simulation.
3. The data-file is stored inside the FMU's `resources` folder, i.e. the FMU reads the parameters during initialization, but no external files are necessary.

Each of those use-cases requires a different implementation in terms of model parameterization.

1.2 User Convenience

From our experience it turns out that enabling all three uses-cases significantly enhances the flexibility of the designed models and especially its exported version e.g. an

[1]Although other ways of exporting like using the `dymosim.exe` or exported source code should behave the same way.

FMU. This enables the same models to be applied as a complete unit coupling models and parameters (use-case 1), as well as using a single model in multiple applications varying parameters by simply replacing data on a file system[2] (use-case 2) or enabling a combination of both (use-case 3). Although covering all use-cases would be the most satisfying solution, it is also valuable to decide for the single most important use-case and implement this one throughout the whole library.

Therefore we are very pleased to present a first proposal within this paper. Until today the presented approach is restricted to scalars and tables up to a dimension of two, but an extension to higher dimensions seems reasonable.

2 A Small Modelica Library

For a better understanding the parameterization of the models and the subsequent model export will be demonstrated using an exemplary Modelica library: The `TableBasedDiodes` library (refer to Figure 1).

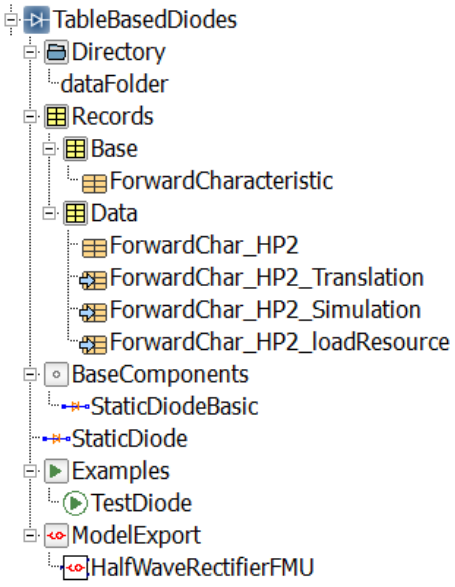

Figure 1: Package structure of the Modelica library

The packages and the models will be explained in the following sections.

2.1 The Utilized Model

The model illustrated in Figure 2, which is located in `BaseComponents.StaticDiodeBasic`, will be used to demonstrate both the parameterization and the export of the model.

[2]Or alternatively modifying the string pointing to the file.

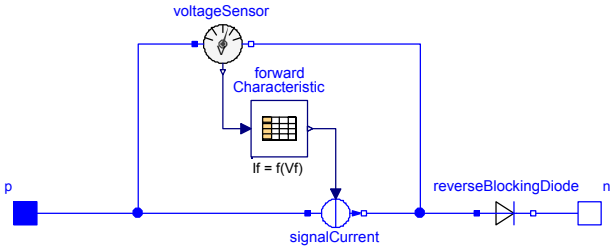

Figure 2: Static model of a diode: $I_f = f(V_f)$

2.2 Model Parameters

To parameterize the model, the diode's forward characteristic, i.e. $I_f = f(V_f)$ is implemented using the one-dimensional table of the MSL (`CombiTable1D`). Since the reverse characteristic is not always provided in datasheets an additional reverse blocking diode (ideal diode of the MSL) is used to ensure that no current will flow in reverse direction. The parameters of this diode are two scalars describing the on-state resistance `Ron` and the off-state conductance `Goff`.

2.3 Record Structure

To provide different data sets, the parameters are usually declared in records. We propose to provide a partial record, i.e. `Records.Base.ForwardCharacteristic` that contains the parameter declarations. This (base) record shall then be extended to assign the parameters with values via modifiers.

Listing 1: Base record of the static diode model

```
partial record ForwardCharacteristic
  extends Modelica.Icons.Record;
  import SI = Modelica.SIunits;

  parameter String Filename = "noFile";
  parameter Boolean tableOnFile = false;

  parameter SI.Current forward[:,:] = fill(0.0
      , 0, 2) "diode's forward characteristic
      i = f(v)";
  parameter SI.Resistance Ron(min = 0, start =
       1e-5) "closed diode resistance";
  parameter SI.Conductance Goff(min = 0, start
       = 1e-5) "opened diode conductance";
end ForwardCharacteristic;
```

2.4 Combining Model and Data

A composition of the diode model shown in Figure 2 and the record in Listing 1 will result in the model depicted in Figure 3.

The parameters of `BaseComponents.StaticDiodeBasic` will be assigned with values stored in the data record via dot-notation. The corresponding code is illustrated in Listing 2.

DOI 10.3384/ecp1511823

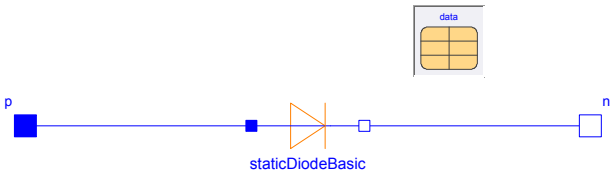

Figure 3: Model and data are combined

Listing 2: Text layer of the static diode model shown in Figure 3

```
model StaticDiode
 ...
  replaceable
      Records.Base.ForwardCharacteristic data
      annotation(choicesAllMatching);

  BaseComponents.StaticDiodeBasic
      staticDiodeBasic(
    tableOnFile=data.tableOnFile,
    table=data.forward,
    tableName="forwardChar",
    fileName=data.Filename,
    Ron=data.Ron,
    Goff=data.Goff);
...
end StaticDiode;
```

In Figure 3 a partial record is instantiated, i.e. a replaceable model is introduced. If we add the annotation `choicesAllMatching` a drop-down menu appears in the model's parameter window (refer to Figure 4).

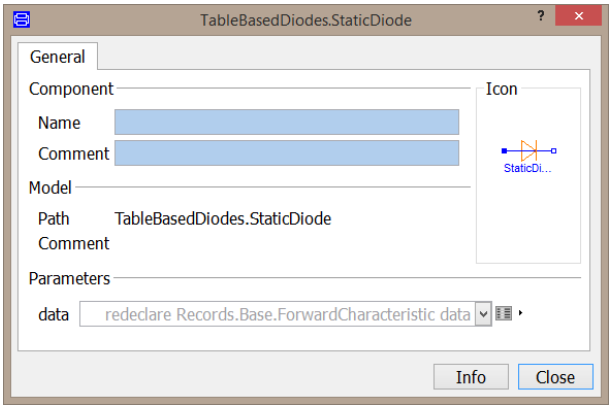

Figure 4: Parameter window of the static diode model

Now, every record that extends the partial (base) record can be selected in the `data` menu. This will be important to be able to easily distinguish the use-cases.

2.5 Getting Data from Files

If data-files are used, it is possible to influence when the parameters of the model are assigned with values specified in the data-file. Assigning can either happen during model translation[3] or at the beginning of the simulation, i.e. during initialization. Assigning parameters during translation means that all values from the data-file are written into the model's code directly. Therefore the file from which the data was read during translation is not needed anymore when the model is simulated. In contrast to that, assigning parameters during initialization of the simulation needs the file available to start the simulation.

Both of the mentioned possibilities can be favorable depending on the scenario the model is used in. Therefore, four implementations will be demonstrated in Section 3 demonstrating how to influence when the parameterization happens. First we will discuss how to handle paths to files efficiently within Modelica.

2.6 File Handling

Typically the data-files are put into a folder located in the libraries root directory, e.g. `TableBasedDiodes` as shown in Figure 1. The data folder is called `Data`. It is common to provide the relative path to this folder inside the Modelica library by introducing, e.g. the package `Directory` shown in Listing 3.

Listing 3: Directory Package

```
package Directory
  constant String dataFolder =
  Modelica.Utilities.Files.loadResource("
      modelica://TableBasedDiodes/") + "Data/"
      ;
end Directory;
```

3 Parameterization of the Model

In the following chapter, four different implementations are introduced to cover the proposed use-cases. Those will be discussed in the following sections.

3.1 Implementation 1: Parameters are Assigned in the Record Directly

A very common case for Modelica library developers is to specify the value of a parameter directly inside the record. This generates an exported model (FMU) that does not need any data-file. This is particularly useful when the user just wants to change only few parameters - preferably scalar values - but not the whole parameter set containing big tables.

Listing 4 illustrates the parameters of the diode's forward characteristic which can be found in the datasheet of Infineon's Hybrid Pack 2.

Listing 4: Data stored in the record

```
record ForwardChar_HP2
"forward characteristic FS800R07A2E3"
  extends Base.ForwardCharacteristic(
  forward = [0,0; 0.5,0.01;
    0.743,7.749; 1.007,109.203;
```

[3] Generating an FMU is a functionality similar to translating a model, resulting in a lot of common properties.

```
    1.126,207.838; 1.226,309.291;
    1.306,407.926; 1.379,509.379;
    1.440,608.014; 1.502,709.467;
    1.555,808.102; 1.609,909.555;
    1.663, 1008.190; 1.713,1109.643;
    1.755,1208.278; 1.805,1309.731;
    1.847, 1408.366; 1.893,1509.82],
  Ron = 1e-5,
  Goff = 1e-5);
end ForwardChar_HP2;
```

One possible drawback when specifying data directly inside the record is that it is rather inconvenient to modify arrays and matrices. Hence, data is often provided inside data-files, i.e. `mat`-files, `csv`-files, `hdf5`-files or any other file formats or even user-specific file formats.

3.2 Implementation 2: Read Data from File during Model Translation

To read data from `csv` or `mat` files Dymola provides functions within the `DataFiles` package. The function `readMATmatrix()` needs two arguments, the filename and the variable name stored inside the file. The `scalar()` function is needed to convert the `(1x1)`-matrix into a scalar value. In Listing 5 the data is read from the file `DiodeFS800R07A2E3.mat`. When the model is translated the values of the data-file are stored inside the record, i.e. the data-file is not needed to simulate the model.

Listing 5: Data read from file during model translation

```
record ForwardChar_HP2_Translation
"forward characteristic FS800R07A2E3 from file
     during Translation"
  extends Base.ForwardCharacteristic(
    forward = DataFiles.readMATmatrix(
        Directory.dataFolder + "
        DiodeFS800R07A2E3.mat", "forwardChar")
        ,
    Ron = scalar(DataFiles.readMATmatrix(
        Directory.dataFolder + "
        DiodeFS800R07A2E3.mat", "Ron")),
    Goff = scalar(DataFiles.readMATmatrix(
        Directory.dataFolder + "
        DiodeFS800R07A2E3.mat", "Goff")));
end ForwardChar_HP2_Translation;
```

In case of a subsequent model export the exported model will behave exactly the same as the one covered in implementation 1. The major difference is, that in this implementation parameter values are read from a file instead of Modelica code during translation. This can be convenient, as tools specialized on data manipulation can be used to generate the data file and they can be independent of the simulation environment as e.g. in (Köhler and Banerjee, 2005).

3.3 Implementation 3: Read Data from File during Model Initialization

This implementation becomes favorable if the user wants to exchange whole data sets of one and the same model. This can be the case when the user wants to generate only one FMU of a model, using different sets of parameters based on data files.

If the record shown in Listing 5 is modified to the record illustrated in Listing 6 the data will not be saved in the model. This is due to two major differences in the implementation.

For the scalar values Dymola's Modelica compiler assumes that the parameter `Filename` replacing the constant String in Section 3.2 is intended to be changed, and is therefore kept as a parameter in the compiled model. This makes it possible to change the `Filename` after model compilation.

For the table values, the ability of the MSL's `CombiTables` is used, which enables the user to decide if an external file or a table from Dymola shall be used. In this case no internal table is used as shown in Section 3.2, but a reference to a file instead. Therefore the table only receives the path to the file containing the data. During simulation the functions within the table itself will access the data directly from the file specified as `Filename`. As the data is not written to the model, the size of the tables within the file are not determined during compilation and the sizes of the tables within the datafile can change without modifying the model itself.

Listing 6: Data read from file during simulation

```
record ForwardChar_HP2_Simulation
"forward characteristic FS800R07A2E3 from file
     during Simulation"
  extends Base.ForwardCharacteristic(
    Filename = Directory.dataFolder + "
        DiodeFS800R07A2E3.mat",
    tableOnFile = true,
    Ron = scalar(DataFiles.readMATmatrix(
        Filename, "Ron")),
    Goff = scalar(DataFiles.readMATmatrix(
        Filename, "Goff")));
end ForwardChar_HP2_Simulation;
```

As one can see in Listing 6, the parameter `forward` (shown in Listing 5) was removed and the parameter `tableOnFile` was set to `true` to enable the functionality described above.

String Parameters in Dymola

If the record `ForwardChar_HP2_Simulation` is chosen (Listing 6), the parameter values are not stored in the FMU. They will be read automatically from the datafile during model initialization. The generated FMU will solely contain the string parameter `Filename` specifying the path to the data-file. Dymola users have to set the following flag to ensure that string parameters appear inside the FMU:

DOI 10.3384/ecp1511823

```
Advanced.AllowStringParameters = true
```

Now one can simply change the to path to the data-file by changing the string parameter.

3.4 Implementation 4: Read Data from File during Model Initialization with Data in the FMU

In many cases it is desired to put the data-files into the FMU since many users don't want to separate model and data when it comes to model export to avoid potential sources of errors. One very common error in that regard is, that data-files are not found as they are not being passed on with the FMU or their (relative) path on the hard drive changed.

Therefore it makes sense to store the data-file in the FMU's `resources` folder. To do so only a slight modification of implementation 3 (Listing 6) is necessary. The resulting code is shown in (Listing 7).

Listing 7: Record used to store the data-file in the FMU's resources folder

```
record ForwardChar_HP2_loadResource
"forward characteristic FS800R07A2E3 from file
     load Resource"
  extends Base.ForwardCharacteristic(
    Filename =
    Modelica.Utilities.Files.loadResource
    (Directory.dataFolder + "
        DiodeFS800R07A2E3.mat"),
    tableOnFile = true,
    Ron = scalar(DataFiles.readMATmatrix(
        Filename, "Ron")),
    Goff = scalar(DataFiles.readMATmatrix(
        Filename, "Goff")));
end ForwardChar_HP2_loadResource;
```

The only difference to the record shown in Listing 6 is the `loadResource()` function applied with the filename as parameter.

Dymola and the FMU's Resources Folder

To make Dymola copy the file to the FMU, the option *Copy resources to FMU* in the *Dymola Simulation Setup* has to be activated. With this flag set and the `loadResources` function in use, the resulting FMU will contain a `resources` folder including the data-file.

As mentioned before this implementation enhances usability, as the user does not have to care about the location of the data files. Still this reduces flexibility as it is not possible anymore to simply change the path to the data file by changing the string parameter introduced in Listing 6. To change the data-file currently the user has to extract the FMU[4], change the data file and compress it again, which is obviously more effort than just changing a parameter.

[4]Which is just a renamed .zip file.

4 Model Export via FMI

For the model export via FMI we use the model `ModelExport.HalfWaveRectifierFMU` depicted in Figure 5.

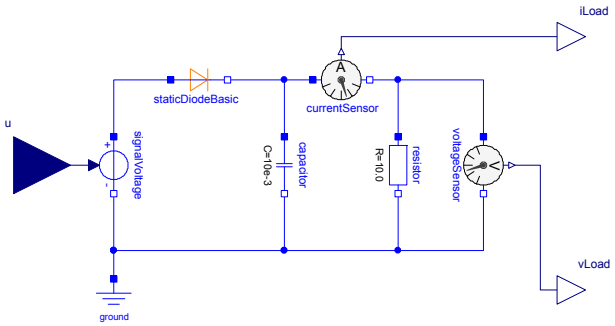

Figure 5: Model used to generate FMUs

We will export the model using the records introduced in the former sections (which can be found in the `Records.Data` package shown in Figure 1). Changing the record by selecting an entry in the pull-down menu will obviously only change the behavior of the `staticDiodeBasic`. The parameters of the `capacitor` and the `resistor` are not affected by the settings of the diode.

Within the drop-down menu that appears when opening the parameter window of the `staticDiodeBasic`, one of the four implementations presented in Section 3 can be chosen. Those are:

1. `ForwardChar_HP2`
2. `ForwardChar_HP2_Translation`
3. `ForwardChar_HP2_Simulation`
4. `ForwardChar_HP2_loadResource`

The identifiers shown in Figure 6 are determined by the model description defined in respective implementations in Listings 4, 5, 6 and 7.

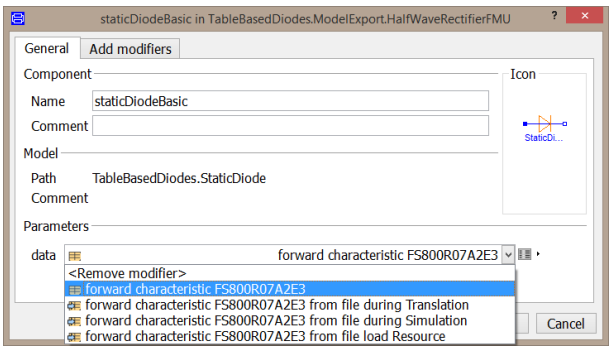

Figure 6: Parameter window of the diode model

By selecting an entry in the pull-down the behavior of both, the model itself in Dymola and the exported FMU

are changed. We can now link the implementations to the use-cases described in Section 1.1.

1. Selecting entry 1 or 2 will both result in a use-case 1 model. The difference will be that entry 1 will rely on data directly stored in the Modelica code and entry 2 will read data from an external file when the model is translated. Still this will not affect the behavior of the generated FMU.

2. Choosing entry 3 will create a use-case 2 model, i.e. load parameters from a data-file which is located in the `Data` folder with an optional string parameter to change the path to the file (see Section 3.3).

3. Compared to the last point selecting entry 4 will only change the behavior in case of FMU generation representing use-case 3. The data files specified by the `loadResources` function will be copied to the FMU's `resources` folder.

5 Tables with Dimensions Greater than Two

It turns out that the implementation is relatively straightforward for scalar parameters and tables with two independent variables, with simple scalar parameters and tables being based on the MSL's `CombiTables`[5]. Still, today it is not possible - at least with reasonable effort - to provide Modelica libraries covering every use-case as soon as tables with a dimension greater than two are necessary. As tables are of essential importance in the industrial use, some of the problems regarding parameter handling with table-based models will be highlighted now, focusing on an arbitrary number of dimensions.

Implementations of tables with dimensions greater than two (n-dimensional) are provided e.g. by Dymola within the `DataFiles` package `TableND` and by 3DS GmbH's `SimDevTools` (`NDTable`).

However, Dymola's n-dimensional table offers no possibility to enter tables directly or read data from a file during model initialization. Thus, not all use-cases can be covered.

Providing proprietary solutions like the `SimDevTools` fails until today, since Dymola requires to pre-compile functions before the translation of the model (e.g. for the determination of the table size). This has to be done manually until today, resulting in a unacceptable inconvenience for the user.

Mixing different table types implicates a number of disadvantages. Regarding the possible use cases, for a model that includes both, e.g. MSL's `CombiTables` for dimensions up to two and `DataFiles`'s `TableND` for higher dimensions, solely use-case 1 can be covered since the Dymola table reads the values from a data-file during model translation and in turn stores them inside the model. Additionally those tables behave differently when it comes to interpolation and extrapolation, which is not directly related to model parameterization, but is a major drawback during simulation and debugging. Table 1 shows a summary of the features of the table implementations available today.

In order to enhance table-based modeling which is of central importance in system simulation within an industrial environment, we want to encourage the Modelica community to put even more effort into this topic. Especially into:

- Extending the tables functionality such that it is possible to use data with more than two independent dimensions by default.

- Providing the possibility to import additional data formats, e.g. `hdf5`, ideally in a user-expendable fashion for arbitrary data formats.

6 Compatibility with Other Modelica Simulation Environments

The implementations shown in Section 3 which are required to enable the different use-cases shown in Section 1.1 were created using Dymola and its `DataFiles` package as well as the MSL. Unfortunately, we were not able to test how other Modelica environments (refer to `https://modelica.org/tools`) treat the implementations, which would be important as the resulting behavior is likely to be tool-dependent.

For the Modelica community it would be highly favorable to have a solution like Dymola's `DataFiles` package available in the MSL. This would enable the user to read data from external files[6] independent of the simulation environment. Ideally it should be extendable to enable customer specific or future data formats.

7 Conclusion

The paper presents three use-cases of model parameterization and four implementations which cover all three use-cases, specifically aimed at a subsequent model export. Additionally it is shown how to implement records with the possibility to choose how an exported model shall behave, by selecting a set of parameters. This property can be set by changing the set of parameters using Modelica based functionality.

This approach is currently only possible for parameters which are scalars, 1D and 2D tables. For higher dimensions it has been pointed out why this is currently not possible.

[5] Which only offer tables up to a dimension of two.

[6] Arrays as well as scalars

DOI 10.3384/ecp1511823

Library	Dims	Formats	Interplation	Extrapolation	Data Source	Parametrization
MSL CombiTables	1-2	txt, mat, csv (import)	hold, linear, smooth first derivative	linear	Modelica/files	translation, initialization
DataFiles	1-n	mat v4, csv	linear	hold	files	translation
SimDevTools	1-32	sdf (hdf5)	hold, linear, Akima	no, hold, linear	files	initialization

Table 1: Table implementations covered in the paper.

References

Modelica Association. `http://fmi-standard.org/`, 2015.

Christian Bertsch, Elmar Ahle, and Ulrich Schulmeister. The functional mockup interface - seen from an industrial perspective. In *Proceedings of the 10th International Modelica Conference*, 2014.

T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, D. Neumerkel, H. Olsson, J.-V. Peetz, and S. Wolf. The functional mockup interface for tool independent exchange of simulation models. In *Proceedings of the 8th International Modelica Conference*, 2011.

T. Blochwitz, M. Otter, J. Akesson M., Arnold 4, C. Clauß, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, H. Olsson, and A. Viel. Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. In *Proceedings of the 9th International Modelica Conference*, 2012.

J. Köhler and A. Banerjee. Usage of modelica for transmission simulation in zf. In *Proceedings of the 4th International Modelica Conference*, 2005.