# D4.1.3 - FMI-Extensions for MODRIO

## WP4.1 – Theory, semantics and standardization of multi-mode DAE systems

## WP4 – Systems with multiple operating modes

# MODRIO (11004)

**Version** M39 (3.0)
**Date** 01/04/2016

**Authors**

| | |
|---|---|
| Torsten Blochwitz | ITI GmbH[1] Dresden, Germany |
| Martin Otter | DLR SR, Germany |
| Hilding Elmqvist, Hans Olsson | Dassault Systèmes, Sweden |
| Rüdiger Franke | ABB AG, Germany |
| Markus Friedrich | SIMPACK AG[2], Germany |
| Andreas Junghanns, Jakob Mauss | QTronic, Germany |
| Antoine Viel | Siemens PLM Software, France |

[1] ITI is now ESI ITI GmbH

[2] SIMPACK AG is now Dassault Systems Deutschland GmbH

## Executive summary

The Functional Mockup Interface (FMI) provides a tool-independent way to exchange dynamic models without and with an embedded simulation engine. In the MODRIO project, FMI was used to utilize nonlinear dynamic models in a tool-neutral way especially for WP3, WP4, WP5, and WP6. In these work packages extensions to FMI 1.0 have been developed, as needed for MODRIO.

The development of the theory for multi-mode Differential Algebraic Equations (DAEs) was more difficult than expected (see **[1]**). For that reason the necessary algorithms for multi-mode DAEs, and the respective Modelica and FMI extensions could not be developed in time. Nevertheless important preconditions for FMI extensions related to multi-mode DAEs have been developed. Furthermore several requirements from other work packages have been considered and led to proposals for FMI extensions. These developments are continued within four FMI Working Groups.

A substantial part of FMI improvements as needed in MODRIO has been standardized in the FMI 2.0 specification **[2]** which was released on July 25, 2014. The Modelica Association Project FMI plans to keep this specification stable for the next years and to introduce the remaining MODRIO related and other changes in a backwards compatible way in minor releases.

This deliverable sketches the MODRIO work which has been performed for release of FMI 2.0. Section 3 summarizes the planned extensions for FMI 2.0 which have been developed in this and other MODRIO work packages.

**Executive summary** ..... **2**

**1. Functional Mock-up Interface Version 2.0** ..... **4**

**2. MODRIO related extensions of FMI 2.0** ..... **4**

2.1. Instantiation ..... 4
2.2. Mathematical Model and Semantics of Event Handling ..... 6

**3. FMI Extensions for MODRIO** ..... **7**

3.1. FMI Working Group: Ports & Icons (WP6) ..... 8
3.2. FMI Working Group: DAE Representation and Partial Derivatives w.r.t. Parameters ..... 9
*3.2.1. DAE Support in FMI (WP4, WP5, WP6)* ..... *9*
3.2.1.1. Use Cases ..... 9
3.2.1.2. Mathematical Structure and Transformation ..... 9
3.2.1.3. Available Solvers ..... 10
3.2.1.4. Initialization and Event Handling ..... 11
3.2.1.5. C-API Extensions ..... 11
3.2.1.6. XML Schema Extensions ..... 11
*3.2.2. Partial Derivatives with Respect to Parameters (WP5)* ..... *12*
3.3. FMI Working Group: Array Variables (WP6, WP4) ..... 12
3.4. FMI Working Group: Clocks & Hybrid Co-Simulation (WP3, WP4, WP5, WP6) ..... 13

**4. References** ..... **13**

# 1. Functional Mock-up Interface Version 2.0

As one result of the ITEA2 project MODELISAR the specifications for FMI for Model Exchange **[3]** and FMI for Co-Simulation **[4]** were released in 2010 as separate documents. Afterwards the FMI development group started to work on FMI 2.0 which combines and unifies both specifications, clarifies ambiguities and adds new features. With the start of the MODRIO project, this work was intensified.

After an intensive development period with public releases of three intermediate versions and a test period which included cross checks between prototypic tool implementations, FMI 2.0 for Model Exchange and Co-Simulation **[2]** was released in July 2014.

Many clarifications have been made based on practical experiences with FMI 1.0 implementations and usage, for example:

- Instantiation (an FMU[3] can now contain both implementations, Model Exchange and Co-Simulation, in parallel).
- Clearer classification of variables.
- Mathematical model of model exchange and co-simulation improved.
- Semantics of event handling improved.
- More precise definition of allowed calling sequences.

Some of the extensions that have been introduced:

- Parameters can be changed during simulation.
- The complete FMU state can be saved, restored and serialized, which enables sophisticated co-simulation master algorithms amongst others.
- The structure of the partial derivatives can be given with respect to states and inputs (to support large systems).
- Directional derivatives can be computed with respect to states and inputs.
- Algebraic loops over FMUs are now supported in all modes (initialization, event, continuous-time) for Model Exchange allowing improved initialization, for example.
- Improved handling of physical units.

# 2. MODRIO related extensions of FMI 2.0

The following FMI extensions have been made considering the MODRIO requirements from WP 4 (multi-mode DAEs) and WP 3 (state estimation).

## 2.1. Instantiation

For instantiation of an FMU the following function is to be called by the environment:

```
fmi2Component fmi2Instantiate(fmi2String  instanceName,
                              fmi2Type    fmuType,
                              fmi2String  fmuGUID,
                              fmi2String  fmuResourceLocation,
                              const fmi2CallbackFunctions* functions,
                              fmi2Boolean                  visible,
                              fmi2Boolean                  loggingOn);
```

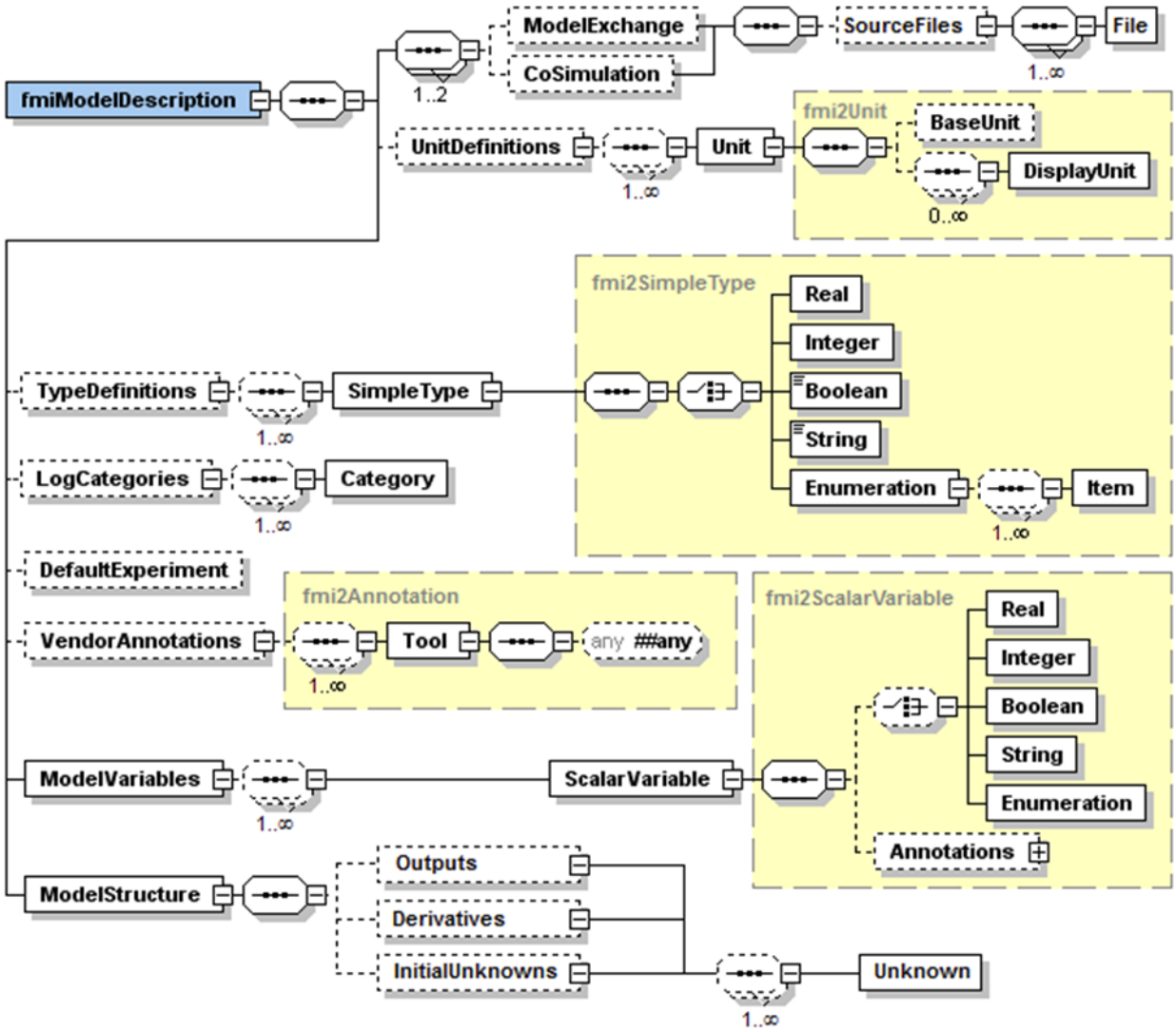
[3] FMU - Functional Mock-up Unit, is a software component that implements the FMI interface

`fmu2Type` is defined as:

```
typedef enum {fmi2ModelExchange,
              fmi2CoSimulation
             }fmi2Type;
```

The value of this argument defines which of the provided implementations of the FMU is used. In this way one FMU can contain different implementations (currently `ModelExchange` and `CoSimulation`) in parallel. Additionally this design allows for adding new FMI kinds without disturbing the existing ones.

Which kind is provided by the FMU is stored in the respective `modelDescription.xml` file. The following figure represents the top level parts of the XML-schema definition graphically (from **[2]**).




Which kind is provided is indicated by the elements `ModelExchange` and/or `CoSimulation` which are children of the top level element `fmiModelDescription`. Additional properties for each kind are stored as attributes of these elements.

As shown below, for adding new features new C-functions are to be defined and the state machines of

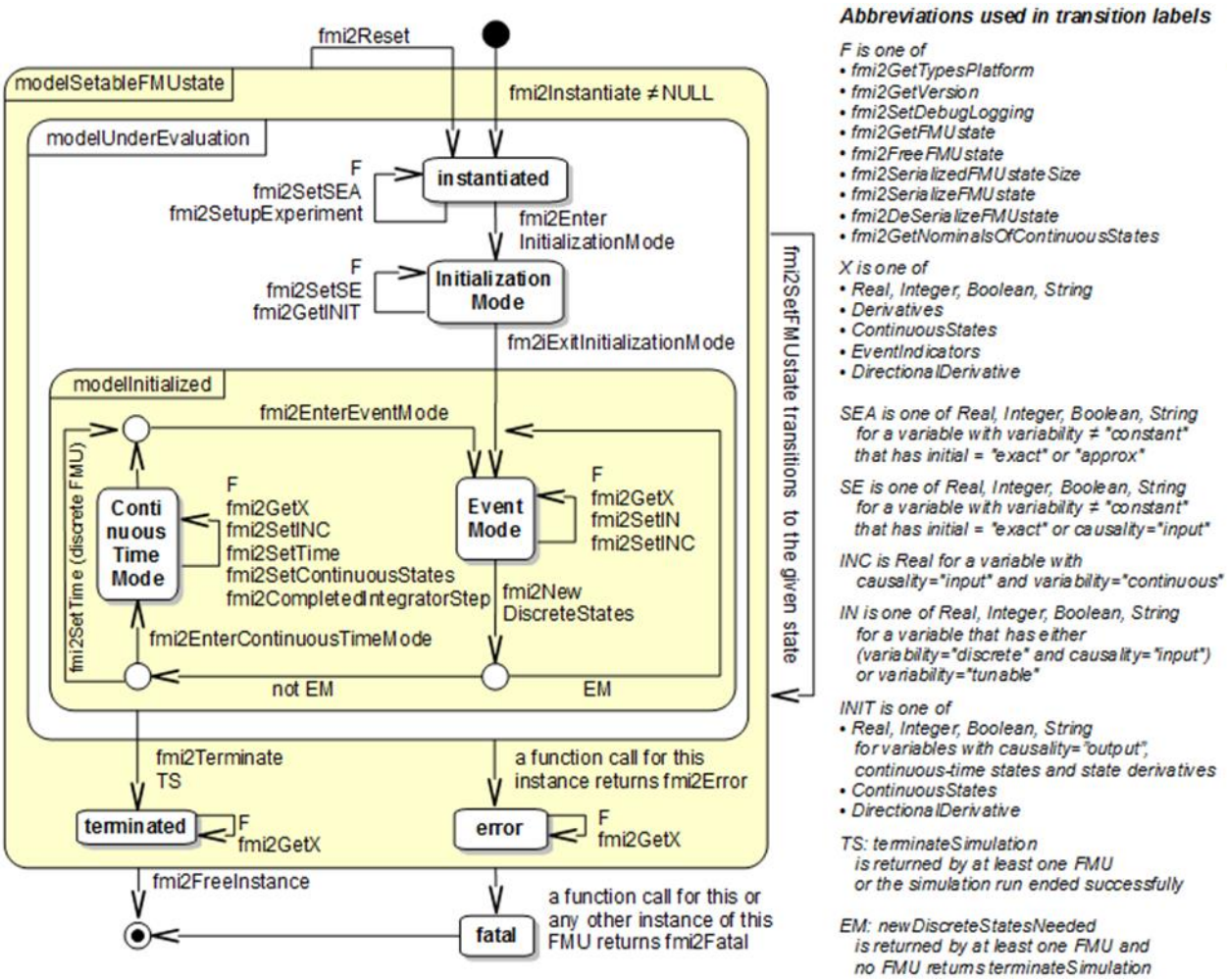
the allowed functions calls might have to be adapted. In order not to disturb the existing FMI kinds, this can be done in new ones, for example `MultiModeDAE` or `Observer`. The only connection points between existing FMI types are the `fmu2Type` structure and the XML-schema definition have to be extended. Both can be done keeping existing 2.0 FMUs compatible to FMI 2.1. In this way the specification work for these aspects can be done independent from other more general FMI extensions.

Additionally another aspect can be covered in that way. The Steering Committee of the FMI Modelica Association Project (which makes the decisions about new features and releases) consists of tool vendors and industrial users with different interests. Not all of them are MODRIO project partners. The new design allows a modularization of the FMI. A tool does not have to support all kinds of FMI, but the ones it is designed for. This will ease the negotiation process for adding features in the FMI Steering Committee.

If in worst case the Steering Committee decides not to include the proposed extensions into a future FMI 2.x version, a working group can still develop its own FMI kind and reference to the official FMI specification for the parts which are reused. Tool vendors are able to provide FMUs which are compatible to FMI and provide additional implementations in parallel, which simplifies tool implementations.

## 2.2. Mathematical Model and Semantics of Event Handling

In FMI 1.0 it was not well defined in which mode (initialization, continuous time, event) an FMU is and under which condition a transition to a different mode occurs. In FMI 2.0 this semantics was completely redefined. The transitions between modes are now defined by introduction of dedicated functions (`fmi2EnterInitializationMode`, `fmi2ExitInitializationMode`, `fmi2EnterContinuousTimeMode`, `fmi2EnterEventMode`…). This semantics is represented by the following state machine (from **[2]**):




The mathematical description defines which equations are evaluated in which mode by the FMU (see **[2]**, page 75, Table 1).

The new semantics of event handling considers the planned FMI extension for handling of clocked (synchronous) systems according to current Modelica 3.3. It is expected that these extensions can be added in a backwards compatible way. A proposal for these extensions is available, but needs to be refined and adapted.

# 3. FMI Extensions for MODRIO

Meanwhile the Modelica Association Project FMI founded the following four working groups:

- Ports & Icons
- DAE Representation & Directional Derivatives
- Array Variables
- Clocks & Hybrid Co-Simulation

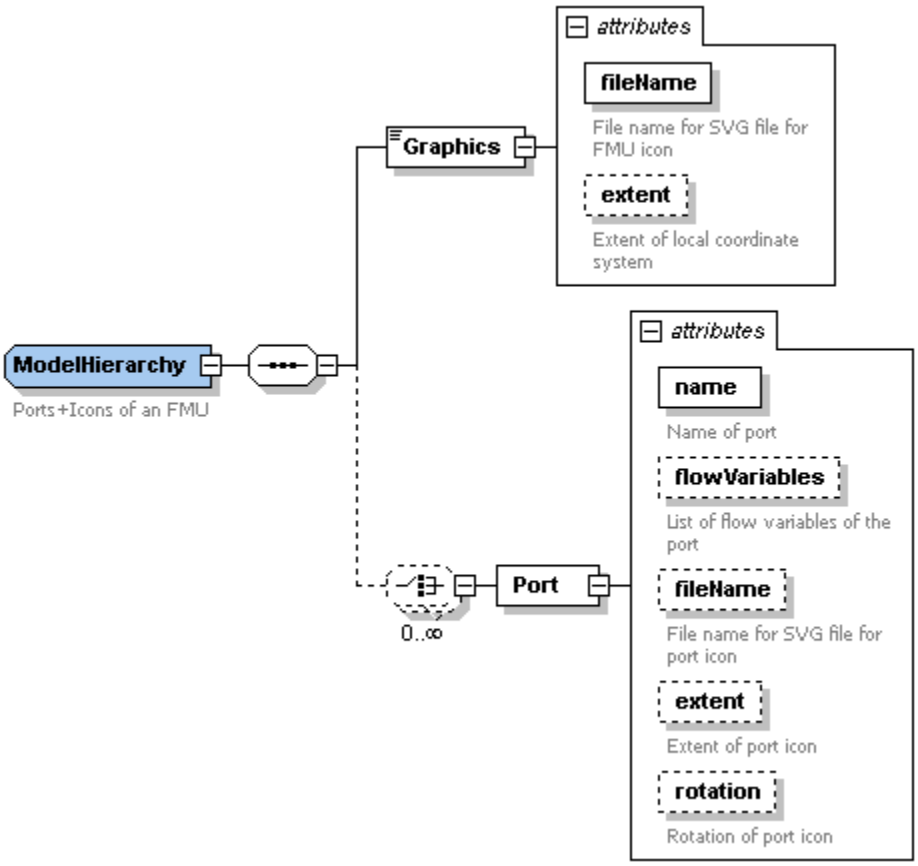
These working groups develop FMI Change Proposals which have to pass certain quality gates before the FMI Steering Committee decides about inclusion to a certain FMI release. This development process is defined within the FMI Change Process Rules **[5]**. Since several partners with different interests are involved it took more time than expected to find solutions which are accepted by all partners. For that reasons the extensions, developed in MODRIO work packages WP3, WP4, WP5, and WP6 are not yet part of a new FMI release. But the results of the MODRIO work have been fed directly to the FMI Working Groups that continue the work after the MODRIO project. The following sections summarize these outcomes.

## 3.1. FMI Working Group: Ports & Icons (WP6)

The information about connectors/ports, especially physical/acausal connectors such as electrical pins, and bus connectors, must be stored in the FMU, in order that a re-import with this information becomes possible. Furthermore, the information about the computational causality in the connector variables must be stored as well. In addition to that, graphical information for the FMU itself, and for the ports need to be stored.

Both extensions allow the export and reimport parts of Modelica models without loss of information in order to support separated compilation techniques for larger Modelica models (WP6).

The proposed solution in **[6]** extends the FMI xml-model description by the new element `<ModelHierarchy>`:




Under this element all ports are listed and the following data are stored:

- Graphical representation and position of ports
- Links to FMI input and output variables that represent the port variables.
- Definition of the flow variables of the port.

In addition to that the proposal **[7]** was provided that includes detailed internal information about the internal hierarchical graph structure of the causality inside the FMU. This allows for semantic checks of networks of FMU regarding solvability.

## 3.2. FMI Working Group: DAE Representation and Partial Derivatives w.r.t. Parameters

### 3.2.1. DAE Support in FMI (WP4, WP5, WP6)

Differential Algebraic Equation (DAE) support within the Functional-Mockup-Interface (FMI) is a major step of the ongoing enhancements of FMI. This topic is handled by the FMI working group “DAE and partial derivatives with respect to parameters” see **[8]**, **[9]** and **[10]**.

After showing several use cases in the next section, the following sections give an overview of the possible mathematical definitions and translations of DAEs. The last sections show the current status of the integration of DAEs in the FMI standard on the level of the Extensible Markup Language (XML) model description file as well as the C interface.

#### 3.2.1.1. Use Cases

DAE support in FMI has several use cases. Even for models without mode changing characteristics there are several use cases for DAEs:

- Avoid the requirement of index reduction inside of FMUs:
  The current FMI standard requires everything to be transformed to an Ordinary Differential Equation (ODE). This reduction may reduce the accuracy and introduces drift.
- Avoid local nonlinear equation solvers inside of FMUs:
  Using DAEs nonlinear algebraic equations can be solved by the global integrator instead of locally. This may improve accuracy and avoid problems with different local and global error tolerances.
- Preserve the sparseness of DAE systems:
  By the translation of a DAE to an ODE the possible sparse structure of the DAE may be lost. This may introduce a large computational overhead of an ODE in contrast to a DAE.
- Allow connections between constraint FMUs:
  Connecting ODE FMUs with internal, locally solved, algebraic loops may lead to a singular system for the coupled equations. Hence, connecting such FMUs is not possible at all. This is solved by DAEs.

#### 3.2.1.2. Mathematical Structure and Transformation

The mathematical structure of a general DAE system is given by the following implicit equation

$$\mathbf{0} = \boldsymbol{F}(\dot{\boldsymbol{x}}, \boldsymbol{x}, \boldsymbol{s}, t)$$

This equation can be reformulated, in case of an index 1 DAE, in semi explicit form

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{s}, t) \quad \text{with } \boldsymbol{g}^1 \text{ solvable for } \boldsymbol{s}$$

$$\mathbf{0} = \boldsymbol{g}^1(\boldsymbol{x}, \boldsymbol{s}, t) \qquad \det\left(\frac{\partial \boldsymbol{g}^1}{\partial \boldsymbol{s}}\right) \neq 0$$

Where **x** are the continuous states, **s** the algebraic states and t the independent variable, usually the time. **f** represents the calculation function for the derivative of the differential states and **g** is the constraint function (the algebraic equations). This formulation is used in FMI.

For higher index DAEs beside the index 1 constraint $\mathbf{g}^1$, being a hidden constraint in this case, additional higher index constraints exist:

$$\begin{aligned}
\mathbf{0} &= \boldsymbol{g}^2(\boldsymbol{x}, \boldsymbol{s}, t) \\
\mathbf{0} &= \boldsymbol{g}^3(\boldsymbol{x}, \boldsymbol{s}, t) \\
&\dots \\
&\text{with } \left\{ \frac{dg_i^k}{dt} \right\} \subseteq \left\{ g_i^{k-1} \right\},\ k > 1
\end{aligned}$$

The transformation from any DAE representation to the above FMI representation is a task specific to the FMU exporting tool. The physical modelling of many domains directly lead to an index 1 or higher index formulation analog to the above FMI representation. In this case no special sophisticated transformations must be done.

But even special, higher index, representations can be reformulated to the above notation. For example a constraint multi-body system in GEAR-GUPTA-LEIMKUHLER stabilized form

$$\begin{aligned}
\dot{\boldsymbol{q}} &= \boldsymbol{u} + \boldsymbol{G}^T \boldsymbol{\mu} \\
\boldsymbol{M}\dot{\boldsymbol{u}} &= \boldsymbol{h}(\boldsymbol{q}, \boldsymbol{u}, t) + \boldsymbol{G}^T \boldsymbol{\lambda} \\
\mathbf{0} &= \boldsymbol{g}(\boldsymbol{q}, t) \\
\mathbf{0} &= \boldsymbol{G}\boldsymbol{u} + \frac{\partial \boldsymbol{g}}{\partial t}
\end{aligned}$$

can be transformed to the above FMI representation by the following equations

$$\begin{aligned}
\boldsymbol{f} &= \begin{bmatrix} \boldsymbol{u} + \boldsymbol{G}^T \boldsymbol{\mu} \\ \boldsymbol{M}^{-1}(\boldsymbol{h} + \boldsymbol{G}^T \boldsymbol{\lambda}) \end{bmatrix} \\
\boldsymbol{g}^1 &= \begin{bmatrix} \boldsymbol{G}\boldsymbol{u} + \boldsymbol{G}\boldsymbol{G}^T \boldsymbol{\mu} + \frac{\partial \boldsymbol{g}}{\partial t} \\ \boldsymbol{G}(\boldsymbol{M}^{-1}\boldsymbol{h} + \boldsymbol{M}^{-1}\boldsymbol{G}^T \boldsymbol{\lambda}) + \boldsymbol{g}^* \end{bmatrix} \\
\boldsymbol{g}^2 &= \begin{bmatrix} \boldsymbol{g} \\ \boldsymbol{G}\boldsymbol{u} + \frac{\partial \boldsymbol{g}}{\partial t} \end{bmatrix} \\
&\text{with } \boldsymbol{x} = [\boldsymbol{q}^T, \boldsymbol{u}^T]^T,\ \boldsymbol{s} = [\boldsymbol{\lambda}^T, \boldsymbol{\mu}^T]^T
\end{aligned}$$

Analog transformations can be done for other DAE representations. For example the typical representation of DAEs in *Modelica* based tools is generated using the PANTELIDES algorithm **[11]**. Using slight adaptions to this algorithm one can translate *Modelica* based models to the representation used by FMI.

If higher index DAEs must be solved using index 1 solvers, index reduction techniques can be applied. Such index reduction normally comes along with "drift" on the hidden (higher index) constraints. However, the FMI representation of DAEs allows us to apply drift correction in such a case. The required C interface functions (`fmi2completedIntegratorStep` and `fmi2EventUpdate`) for such corrections are already included in FMI 2.0. Please see the slides in **[9]** for more details.

The same applies for other stabilization techniques like BAUMGARTE stabilization or over determined DAE solvers, e.g. ODASSL: the FMI representation of DAEs allows the use of such solvers or techniques. See again **[9]** for more details.

#### 3.2.1.3. Available Solvers

To solve DAEs many freely available codes exist and are used in commercial tools. These codes can mainly be categorized by the representation of the DAE, being the input function to the solvers: implicit, semi-explicit or "mass-matrix-form". Apart from this many solvers are restricted to a specific DAE index. Since the last section has already covered the translation between different DAE representations the following enumeration will just list different available solvers sorted by the DAE

index and language:

- DAE index 1 solvers
  - *Fortran*: DASSL, DASPK, …
  - *MATLAB*: ode15i, ode15s, ode23t, ode23, ode23s, ode23tb, ode45
  - (many other index 1 DAE solvers available)
- For semi-explicit DAEs of index >= 1:
  - *Fortran*: RADAU5, MEBDF, ODASSL, …
  - (much less higher index DAE solvers available)

#### 3.2.1.4. Initialization and Event Handling

Initialization is a major topic for DAE systems. One reason for this is the fact that many DAE solvers are not able to start the integration if the initial state is not fully consistent up to some numerical boundaries. However, FMI 2.0 already contains a quite sophisticated mechanism for calculating initial conditions. This includes the use of special initial equations which are only used during the initialization phase but not during the integration or event handling phase. This initialization mechanism of FMUs is sufficient for DAE initialization.

However, the event handling mode of FMUs containing DAEs must be extended. This topic is work in progress within the FMI DAE working group.

#### 3.2.1.5. C-API Extensions

Extensions to the FMI C Application Programing Interface (API) are not required for the DAE support. Getting and setting of the new variables, algebraic states and residues, can be done using the existing `fmi2SetReal` and `fmi2GetReal` function calls. This is only possible if all value references of these variables are known. As seen in the next section this is true since residues and algebraic variables are listed in the `ModelStructure` section of the model description XML file. Using the variable index one can get the value reference of each variable.

#### 3.2.1.6. XML Schema Extensions

For each algebraic state and each residue variable a `ScalarVariable` element is added to the `ModelVariables` XML element, just like for any other variable.

```
<ModelVariables>
  ...
  <!-- an algebraic variable -->
  <ScalarVariable causality="local" variability="continuous"
                  initial="approx" ...>
    <Real .../>
  </ScalarVariable>
  <!-- a residue variable -->
  <ScalarVariable causality="local" variability="continuous"
                  initial="calculated" ...>
    <Real .../>
  </ScalarVariable>
  ...
</ModelVariables>
```

The `causality` attribute for both is `local`, and the `variability` attribute for both is `continuous`. The `initial` attribute for algebraic variables is `approx` and for residues `calculated`. This definition is analog to the definition of continuous states and state derivatives in FMI 2.0.

Beside the pure definition of the new DAE variables also the dependencies of these must be known, for examples for getting the sparse JACOBIAN pattern for partial or directional derivatives. This is done by adding the new XML element `Constraints` to the element `ModelStructure`. A typical

`Constraints` element will look like this:

```
<ModelStructure>
  ...
  <Constraints>
    <Constraint algebraicStateIndex="4">
      <Unknown index="10" dependencies="..." dependenciesKind=".."/>
      <Unknown index="13" dependencies="..." dependenciesKind=".."/>
      <Unknown index="14" dependencies="..." dependenciesKind=".."/>
    </Constraint>
    <Constraint algebraicStateIndex="6">
      <Unknown index="23" dependencies="..." dependenciesKind=".."/>
    </Constraint>
    ...
  </Constraints>
  ...
<ModelStructure>
```

This defines a DAE with two constraints and two algebraic variables. The first constraint is of index 3 since it has three `Unknown` elements and the corresponding algebraic state has the variable index 4. The second constraint is of index 1 since it has only one `Unknown` element and the corresponding algebraic variable has the variable index 6. The `Unknown` XML elements are just like the already available `Unknown` elements from the `ModelStructure` XML section: they define the variable index of the corresponding variable as well as its dependencies and kind of dependencies.

It should be noted that it is currently under discussion in the FMI DAE working group whether DAEs of higher index should be included in the next FMI release or not. If only index 1 DAEs are supported the model description XML file will be different to the above one.

### 3.2.2. Partial Derivatives with Respect to Parameters (WP5)

FMI 2.0 already supports directional derivatives of outputs and derivatives w.r.t. inputs and continuous states. For non-linear model predictive control applications and certain optimization algorithms partial derivatives w.r.t. parameters are needed (see for example **[12]**). The idea is to reuse the FMI 2.0 functions for getting directional derivatives (`fmi2GetDirectionalDerivative`) and include parameters by allowing to add them as $\mathbf{v}_{known}$ to the xml-model description element `<dependencies>` (see page 61 of **[2]**).

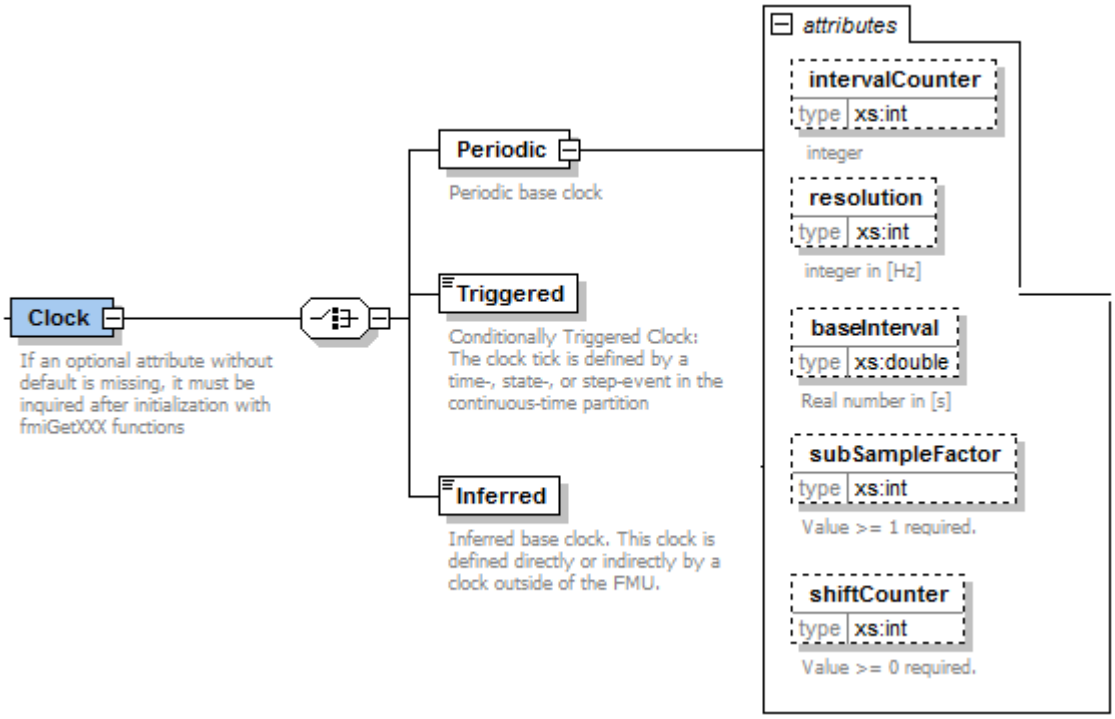
## 3.3. FMI Working Group: Array Variables (WP6, WP4)

The MODRIO demands regarding FMU import loss of information requires the support of hierarchical data structures. The structured variable naming convention of FMI 2.0 (**[2]** section “2.2.9 Variable Naming Conventions”) can be used to support most of the use cases for exchange of structured data. Extensions of FMI C-functions for an efficient support on API level extremely complicates the interface. For that reason the work on this feature has been postponed by the FMI group. However the efficient support of variables of higher dimensions and of parameters with variables dimension enhances the usability of FMI a lot. For that reason the FMI Working Group “Array Variables” currently concentrates on this feature. The idea is to add the list element `<Dimensions>` to the definition of non-scalar variables that contain the default dimensions of multi-dimensional variables. Each element of `<Dimensions>` has a `valueReference` attribute that allows setting and getting of dimensions (as far as it is allowed) using the existing `fmi2Get/SetInteger` functions. For the efficient getting and setting of array variables additional API functions are defined.

## 3.4. FMI Working Group: Clocks & Hybrid Co-Simulation (WP3, WP4, WP5, WP6)

When exporting a Modelica model in the FMI 2.0 format, information about clocks and clocked variables is not supported and therefore lost. When re-importing such an FMI, this information is lost as well. Supporting clocks and clocked variables in FMI will allow describing sampled-data systems and especially, precise synchronization of clocks. Several different draft proposals from ITI, DLR and DS are available. In **[13]** these proposals have been combined and slightly simplified. The approach was motivated by a demonstration of a predictive load set point optimization.

The main idea is to introduce new `<clock>` elements to the FMI model description:




By new API functions `fmi2SetClock` clocks can be activated, using `fmi2GetClock` internally activated clocks can be inquired when the FMU is in `EventMode`. The interval time triggered clocks can be set or retrieved by `fmi2SetInterval` and set with `fmi2GetInterval`. The new element `<DiscreteStates>` under `<ModelStructure>` defines the discrete state variables and their dependencies on other variables and on clocks.

In this FMI working group MODRIO partners cooperate with members of the Horizon 2020 project INTO-CPS.

# 4. References

[1] A. Benveniste, T. Bourke, B. Caillaud, M. Pouzet: *Semantics of multi-mode DAE systems.* MODRIO deliverable D4.1.1: https://modrio.org/svn/MODRIO/trunk/WP4 Multiple Modes/WP4.1 Theory and Semantics/D4.1.1/D4.1.1_M24_Semantics_of_multi-mode_dae_systems.docx

[2] Modelica Association: *FMI for Model Exchange and Co-Simulation*, Version 2.0, July 25th, 2014. https://www.fmi-standard.org/downloads

[3] MODELISAR Consortium: *Functional Mock-up Interface for Model Exchange.* https://svn.modelica.org/fmi/branches/public/specifications/v1.0/FMI_for_ModelExchange_v1.0.pdf

[4] MODELISAR Consortium: *Functional Mock-up Interface for Co-Simulation.* https://svn.modelica.org/fmi/branches/public/specifications/v1.0/FMI_for_CoSimulation_v1.0.pdf

[5] Modelica Association: *FMI Development Process And Communication Policy*. https://svn.fmi-standard.org/fmi/branches/public/docs/DevProcess/FMI_DevelopmentProcess_1.0.pdf

[6] H. Elmqvist (Dassault Systemés): *Initial version of FMI change proposal FMI with Acausal Ports and Icons.* Internal document of FMI Working group: https://svn.fmi-standard.org/fmi/trunk/WorkingGroups/01_Port_Icons/Meetings/2015-08-25-InitialWebMeeting/

[7] Sébastien Furic (Siemens Industry Software): *Proposal for Extending the FMI with Physical Modelling Capabilities.* Presentation at FMI Design Meeting December, 15th, 2015, Oberpfaffenhofen: https://svn.fmi-standard.org/fmi/trunk/WorkingGroups/01_Port_Icons/Meetings/2015-12-14-FMI_Design_Meeting_DLR/Proposal_for_Extending_the_FMI_with_Physical_Modelling_Capabilities.pdf

[8] FMI DAE Working Group web meeting, https://trac.fmi-standard.org/browser/trunk/WorkingGroups/02_DAE_PartialDerivatives/Meetings/2015-09-01-InitialWebMeeting, Sep. 2015

[9] FMI DAE Working Group meeting, https://trac.fmi-standard.org/browser/trunk/WorkingGroups/02_DAE_PartialDerivatives/Meetings/2015-09-24-FMI_Design_Meeting_Velizy, Velizy, France, Sep. 2015

[10] FMI DAE Working Group web meeting, https://trac.fmi-standard.org/browser/trunk/WorkingGroups/02_DAE_PartialDerivatives/Meetings/2016-03-17-FMI-WebMeeting, Mar. 2016

[11] Sven Erik Mattsson, Martin Otter, Hilding Elmqvist: *Multi-Mode DAE Systems with Varying Index*, Modelica Conference, 2015: https://modelica.org/events/modelica2015/proceedings/html/submissions/ecp1511889_MattssonOtterElmqvist.pdf

[12] Andreas Pfeiffer, Johan Akesson: *D5.1.2 - Development of an FMI-based standardized interface for executing NMPC within virtual environments.* MODRIO deliverable: https://modrio.org/svn/MODRIO/trunk/Deliverables ITEA2/Project_Review_3/WP5/D5.1.2/D5.1.2_M36_FMIInterfaceNMPC.pdf

[13] Rüdiger Franke: *Discrete states and time events in FMI - Exploit Modelica Synchronous Features.* Presentation at FMI Design Meeting December, 15th, 2015, Oberpfaffenhofen: https://svn.fmi-standard.org/fmi/trunk/WorkingGroups/04_Hybrid_CoSimulation/2015-12-14-FMI_Design_Meeting_DLR/DiscreteStates_TimeEvents.pdf