Annex C
(informative)

Generic Software Interface for use of models in different software environments

C.1 Description of the approach

It is not possible to describe all relevant models in a generic form. Reasons may be

– the model contains functionality that is not available in the generic models

– the model contains proprietary information that should not be made available for a broad public

– the model is expected to be an exact copy of a real control implementation and consists of the original control source-code of a controller

For the use of such models it may be required

– to use a model in different software environments

– to use models created by different manufacturers that used different software environments

The use of a generic software-interface allows the use of the same model in compiled form both in different software environments and in combination with models from other sources. Depending on the software environment, features like calling sequence, integration algorithm, parameter handling may be handled differently. The concept of the generic interface ensures that a model will function and deliver correct results under such conditions. Some of the core requirements are

– support for both internal solvers (in the model) and external solvers (states and state derivatives provided to the simulation environment)

– support for multiple instances of a model

– optional support for variable step execution of models

– optional support for iterative load flow calculation

– optional support functions for use in graphical user interfaces (input, output and parameter name visibility)

– optional support for parameter change

The generic software interface provides all the functionality to implement models in different simulation environments. The interface description is based on an implementation in C-Code since this is the most common programming language, but there is no restriction on the programming language in which the interface is implemented.

Various simulations in different software environments DLL-models are a good approach to maintain protection of intellectual property and reproducibility of results. An important part of a DLL for various simulation tasks is a flexible interface which is able to handle the requirements of the different simulation environments. The Extended Simulation Environment interface (ESE-interface) meets these requirements. Its data structures and functions are described in this annex.

C.2 Description of the Software interface

C.2.1 Description of data structures

C.2.1.1 General

For communication through the ESE-interface the following data structures are used (C-Code).

C.2.1.2 StaticExtSimEnvCapi

Description: Contains general information about the model

```
typedef struct
{     const uint8_T                     APIRelease[4];              // Release number of the API used during
                                                                    // code generation
      const char_T * const              ModelName;                  // Model name, IEC Version name
      const char_T * const              ModelVersion;               // Model version
      const char_T * const              ModelDescription;           // Model description
      const char_T * const              VersionControlInfo;         // Version control information
      const char_T * const              GeneralInformation;         // General: info like copyright, owner, …
      const char_T * const              ModelCreated;               // Model created on
      const char_T * const              ModelCreator;               // Model created by
      const char_T * const              ModelLastModifiedDate;      // Model last modified on
      const char_T * const              ModelLastModifiedBy;        // Model last modified by
      const char_T * const              ModelModifiedComment;       // Model modified comment
      const char_T * const              ModelModifiedHistory;       // Model modified history
      const char_T * const              CodeGeneratedOn;            // Code generated on
      const char_T * const              IncludedSolver;             // Solver name (can be empty)
      const real64_T                    FixedStepBaseSampleTime;// Base sample time
      const int32_T                     NumInputPorts;              // Number of inputs
      const StaticESEInputSignal * const  InputPortsInfo;           // Pointer to input signal description array
      const int32_T                     NumOutputPorts;             // Number of outputs
      const StaticESEOutputSignal * const OutputPortsInfo;          // Pointer to output signal description
                                                                    // array
      const int32_T                     NumParameters;              // Number of parameters
      const StaticESEParameter * const  ParametersInfo;             // Pointer to parameter description array
      const int32_T                     NumContStates;              // Number of continuous states
      const int32_T                     SizeofMiscStates;           // Size of work variables / misc states
      const uint32_T                    ModelChecksum[4];           // model checksum
      const char_T                      *LastErrorMessage;          // Error string pointer
      const uint8_T                     EMT_RMS_Mode;               // Mode: EMT = 1, RMS = 2,
                                                                    // EMT & RMS = 3,
                                                                    // otherwise: 0
      const uint8_T                     LoadflowFlag;               // Model contains a loadflow function:
                                                                    // 0 = no, 1 = yes
      ESEExtension                      Extension;                  // Provided for extensions

}StaticExtSimEnvCapi;
```

C.2.1.3 InstanceExtSimenvCapi

Description: Contains runtime specific information

```
typedef struct
{           real64_T          *ExternalInputs;          // Input signals, all elements in one long vector
            real64_T          *ExternalOutputs;         // Output signals, all elements in one long vector
            real64_T          *Parameters;              // Parameters as vector
            real64_T          *ContinuousStates;        // We assume a states vector
            real64_T          *StateDerivatives;        // We assume a states derivatives vector
            uint8_T           *MiscStates;              // Work variables / states with unknown content
            const char_T      *LastErrorMessage;        // Error string pointer
            const char_T      *LastGeneralMessage;      // General message
            uint8_T           VerboseLevel;             // Decides how much the code "should talk"
            ESEExtension      Extension;                // Provided for extensions
}InstanceExtSimEnvCapi;
```

C.2.1.4 StaticESEInputSignal

Description: Contains information about an input signal

```
typedef struct
{           const char_T * const      Name;                     // Input signal name
            const char_T * const      BlockPath;                // Path to block in model
            const int32_T             Width;                    // Signal width
}StaticESEInputSignal;
```

C.2.1.5 StaticESEOutputSignal

Description: Contains information about an output signal

```
typedef struct
{           const char_T * const      Name;                     // Output signal name
            const char_T * const      BlockPath;                // Path to block in model
            const int32_T             Width;                    // Signal width
}StaticESEOutputSignal;
```

C.2.1.6 StaticESEParameter

Description: Contains information about model parameters

```
typedef struct
{           const char_T * const        Name;             // Parameter name
            const char_T * const        Description;      // Description
            const char_T * const        Unit;             // Unit
            const real64_T              DefaultValue;     // Default value
            const real64_T              MinValue;         // Minimum value
            const real64_T              MaxValue;         // Maximum value
}StaticESEParameter;
```

C.2.1.7 ESEExtension

Description: Additional memory for later extensions

```
typedef union
{           int8_T           UserInt8_8[8];
            uint8_T          UserUint8_8[8];
            int16_T          UserInt16_4[4];
            uint16_T         UserUint16_4[4];
            int32_T          UserInt32_2[2];
            uint32_T         UserUint32_2[2];
            char_T           UserChar_8[8];
            real32_T         UserReal32_2[2];
            real64_T         UserReal64;
            void             *UserVoidPtr;
}ESEExtension;
```

C.2.2 Functions for communication through the ESE-interface

The following functions control the sequence of the simulation. A typical sequence is shown in Figure C.1.

C.2.2.1 const StaticExtSimEnvCapi* __cdecl Model_GetInfo():

Description: Provides general information about the model

Return value: NULL on error, else pointer to filled StaticExtSimEnvCapi structure

C.2.2.2 InstanceExtSimEnvCapi* __cdecl Model_Instance(uint32_T UseSolverInDLL, real64_T Ta):

Description: Creates instance of the model

UseSolverInDLL: 1: Internal solver, 0: External solver
Ta: >0: sample time; -1: Pre defined sample time
Return value: NULL on error, else pointer to filled InstanceExtSimEnvCapi structure

C.2.2.3 const char_T* __cdecl Model_CheckParameters(InstanceExtSimEnvCapi *pInstanceCapi):

Description: Checks if parameter values are in the correct range

Return value: NULL on error, else string with error description

C.2.2.4 const char_T* __cdecl Model_Loadflow(InstanceExtSimEnvCapi *pInstanceCapi):

Description: Performs a load flow iteration

Return value: Null if no error, else string with error description

C.2.2.5 const char_T* __cdecl Model_Initialize(InstanceExtSimEnvCapi *pInstanceCapi):

Description: Initialises the model

Return value: NULL if no error, else string with error description

C.2.2.6 const char_T* __cdecl Model_Outputs(InstanceExtSimEnvCapi *pInstanceCapi, uint32_T IsMajorTimeStep):

Description: Performs timestep and recalculates outputs,
based on states and inputs updated by ‘Model_Update’

IsMajorTimeStep: 1: Major timestep; 0: Minor timestep (between two integrations)
Return value: Null if no error, else string with error description

C.2.2.7 const char_T* __cdecl Model_Update(InstanceExtSimEnvCapi *pInstanceCapi):

Description: Read inputs and update state variables

Return value: Null if no error, else string with error description

C.2.2.8 const char_T* __cdecl Model_Derivatives(InstanceExtSimEnvCapi *pInstanceCapi):

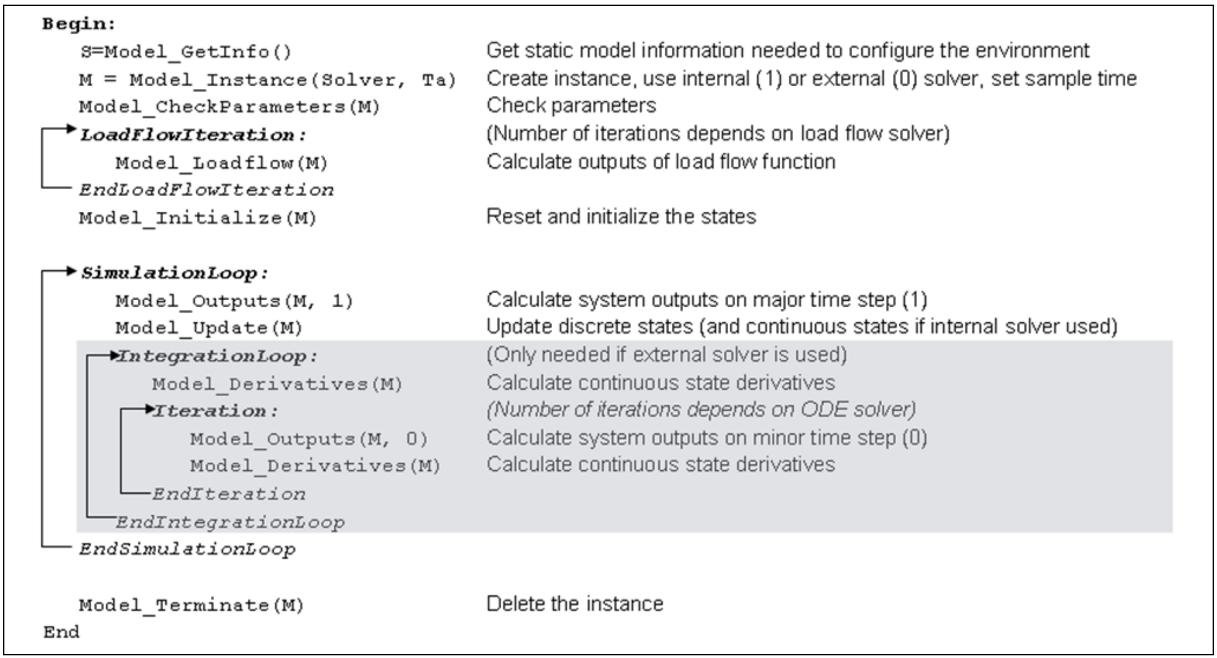
Description: Calculate derivatives of state variables (only needed if external solver is used)

Return value: Null if no error, else string with error description

C.2.2.9 const char_T* __cdecl Model_Terminate(InstanceExtSimEnvCapi *pInstanceCapi):

Description: Delete model instance and deallocate memory

Return value: Null if no error, else String with error description



```
Begin:
   S=Model_GetInfo()                   Get static model information needed to configure the environment
   M = Model_Instance(Solver, Ta)      Create instance, use internal (1) or external (0) solver, set sample time
   Model_CheckParameters(M)            Check parameters
   LoadFlowIteration:                  (Number of iterations depends on load flow solver)
      Model_Loadflow(M)                Calculate outputs of load flow function
   EndLoadFlowIteration
   Model_Initialize(M)                 Reset and initialize the states

   SimulationLoop:
      Model_Outputs(M, 1)              Calculate system outputs on major time step (1)
      Model_Update(M)                  Update discrete states (and continuous states if internal solver used)
      IntegrationLoop:                 (Only needed if external solver is used)
         Model_Derivatives(M)          Calculate continuous state derivatives
         Iteration:                    (Number of iterations depends on ODE solver)
            Model_Outputs(M, 0)        Calculate system outputs on minor time step (0)
            Model_Derivatives(M)       Calculate continuous state derivatives
         EndIteration
      EndIntegrationLoop
   EndSimulationLoop

   Model_Terminate(M)                  Delete the instance
End
```

IEC

Figure C.1 – Sequence of Simulation on use of ESE-interface

C.2.3 Inputs, Outputs, Parameters

The following restrictions apply for the interface:

- Floating Point values
- Scalars or vectors (no matrices, structures or busses)
- Real values (not complex)
- Inputs and Outputs sample-based (not frame-based)