1 Protocol

The COE protocol is based on a JSON over HTTP protocol using a similar approach as REST. However, unlike REST, the COE keeps state. The format used to describe each command is: URL and any arguments, prefixed with a colon, *e.g.* "*:section*".

1.1 COE Information

Information about the COE is available at:
`http://localhost:8082/`

1.2 The API Command

The command is available at:
`http://localhost:8082/api`

or the following for a PDF version:
`http://localhost:8082/api/pdf`

If successful, the command returns one of two types of content:

Content-Type: application/pdf A PDF version of this document.

Content-Type: text/plain The LaTeX source file of this document.

1.3 The Status Command

The command is available at:
`http://localhost:8082/status/:session`

The command takes the following arguments:

:session Optional session id filtering the returned data array to the single instance with the given session id.

If no session is provided, then the command returns an array of all sessions status similar to the example below.

```
[
{
    "status":"idle",
    "sessionid": "-1"
},
{
    "status":"idle",
    "sessionid": "0"
}
]
```

If a session ID is given, then a single session is returned

```
{
    "status":"idle",
    "sessionid": "-1"
}
```

1.4 The Create Session Command

The command is available at:
`http://localhost:8082/createSession`
The command takes no argument and returns a JSON string containing the session id. An example is presented below, where the sessionId is 12345:

```
{"sessionId":"12345"}
```

1.5 The Attach Session Command

The command is available at:
`ws://localhost:8082/attachSession/:session`
The command takes no arguments and opens a WebSocket (`https://tools.ietf.org/html/rfc6455`). Output data from connected outputs will be live streamed according to the following format:

```
{
    "{fmuName}":{
        "instanceName":{
            "variableName": variableValue
        }
    }
}
```

1.6 The Initialize Command

The command is available at:
`http://localhost:8082/initialize/:session`

The command takes the following argument and requires a JSON payload, Content-Type: application/json:

:session The session ID.

The Data payload:

```
{
    "fmus":{
        "{controllerFmu}":"file://controller.fmu",
        "{tankFmu}":"file://tank.fmu"
    },
    "connections":{
        "{controllerFmu}.crtlIns.valve":["{tankFmu}.tankIns.valve"],
        "{tankFmu}.tankIns.level":["{controllerFmu}.crtlIns.level"]
    },
    "parameters":{
        "{controllerFmu}.crtlIns.maxLevel":8,
        "{controllerFmu}.crtlIns.minLevel":2
    },
    "algorithm":{
        FIXED-STEP-SIZE-CONFIG or VARIABLE-STEP-SIZE-CONFIG
    },
    "livestream":{
        "{controllerFmu}.crtlIns":["local","local2"],
        "{tankFmu}.tankIns":["level"]
    },
    "logVariables":{
        "{tankFmu}.tankIns":["local"]
    },
    "parallelSimulation": false,
    "stabalizationEnabled":false,
    "global_absolute_tolerance": 0.0,
    "global_relative_tolerance":0.01

}
```

`FIXED-STEP-SIZE-CONFIG` The fixed step size configuration contains the following:

```
        "type":"fixed-step",
        "size":0.1
```

`VARIABLE-STEP-SIZE-CONFIG` The variable step size configuration contains the following:

```
        "type":"var-step",
        "size":[1E-6, 1.0],
        "initsize":1E-4,
        "constraints":{
            STEPSIZE-CONSTRAINT*
        }
```

Where the properties are defined as:

`type:"var-step"` selects the variable stepsize calculator (each algorithm has a `type`).

`size:[<minimal stepsize> , <maximal step size>]` defines the stepsize interval as an array of double values.

`initsize:<initial stepsize>` defines the initial stepsize as double value.

`constraints:` defines the stepsize constraints as follows:

```
            "id":{
                type:"[zerocrossing|boundeddifference|samplingrate|
                    fmumaxstepsize]",
                ...
            }
```

The `id` is a string that is used to identify the constraint e.g. in the log. All constraints have a single common name-value pair with name `type`. The value of `type` specifies the type of the constraint; the other name-value pairs of the constraint depend on the value of `type`. The `zerocrossing` is described in Section 1.6.1, `boundeddifference` in Section 1.6.2, and `samplingrate` in Section 1.6.3 `fmumaxstepsize` in Section 1.6.4.

The JSON payload contains the following entries:

fmus A list of the location of the FMUs.

connections A map of connections, output to input.

parameters A map from parameter to value.

algorithm Step size algorithm configuration:

fixed-step A fixed step size algorithm is available requiring a step size to be specified.

livestream These scalar variables will be livestreamed via websockets. They must have a causality of either local or output.

logVariables These scalar variables will be logged. They must have a causality of either local or output.

parallelSimulation Optional boolean condition specifying if the COE should parallelize certain parts of the simulation. This condition is or'ed together with all `simulation.parallelise.*` properties.

stabalizationEnabled This enabled stabilization mode in the COE. Currently it makes use of *successive substitution* retrying a maximum of 5 steps. It uses the `global_absolute_tolerance` and `global_relative_tolerance` to decide if the signals are stable or another stabilization step is needed. The implementation uses the *Numpy.isclose* function. When this option is enabled the cyclic check is disabled and a warning is for any FMU that is in a cycle that does not support get and set state from FMI.

global_absolute_tolerance The global absolute tolerance used for stabilization.

global_relative_tolerance The global relative tolerance used for stabilization.

The command returns the following response on success:

```
    {
        "status":"initialized",
        "sessionid": "1234",
        "avaliableLogLevels":{
            "{8c4e810f-3df3-4a00-8276-176fa3c9f001}.tank":[
            {
                "name":"logAll",
                "description":"Description of this loggin level"
            },
            {
                "name":"logError",
                "description":null
            }],
            "{8c4e810f-3df3-4a00-8276-176fa3c9f000}.controller":[
            {
                "name":"logAll",
            }]
        }
    }
```

The `avaliableLogLevels` value will be specific to the FMUs given in the initial payload. The `sessionid` is the ID that must be supplied in any subsequent calls.

1.6.1 Definition of a Zero Crossing Constraint

A constraint of `"type":"zerocrossing"` is defined by

```
    "id":{
        "type":"zerocrossing",
        "ports":[
        "<guid>.<instance>.<outport>",
        "<guid>.<instance>.<outport>"
        ],
        "order":[1|2],
        "abstol":<double>,
        "safety":<double>
    }
```

where the second entry in the `ports` list and the attributes `order`, `abstol` and `safety` are optional. The name-value pairs have the following meaning.

- `ports:` Defines the zero crossing function f as an array of strings of size 1 or 2. If one output port is provided, then f is the value of that output port. If two output ports are provided, then f is the difference between the values of the first and second output ports. Any other size of the string array is not supported.

- `order:` This name-value pair is optional; it specifies the extrapolation order that is used to predict a zero crossing (see Section 2.3.1). First and second order extrapolation are supported. The default is second order extrapolation.

- `abstol:` This name-value pair is optional; it specifies the absolute tolerance. The stepsize calculator attempts to adjust the stepsize such that at a time instant t_{ZC} the absolute value of the zero crossing function f is smaller or equal to the absolute tolerance, $|f(t_{ZC})| \leq abstol$. The default value for the absolute tolerance is 10^{-3}.

- `safety:` This name-value pair is optional; it adjusts the conservatism of the zero crossing prediction. The neutral default value is 0.0. If the variable stepsize calculator fails to resolve a zero crossing of a particular co-simulation within the absolute tolerance (and the minimal stepsize is not the limiting factor), then the value for `safety` can be increased for more conservatism in the zero crossing prediction. Negative values for less conservatism are mathematically possible, but should probably not be used.

1.6.2 Definition of a Bounded Difference Constraint

A constraint of `"type":"boundeddifference"` is defined by

```
        "id":{
            "type":"boundeddifference",
            "ports":[
            "<guid>.<instance>.<outport>"
            ,"<guid>.<instance>.<outport>"
            ,"<guid>.<instance>.<outport>"
            ...
            ]
            ,"abstol":<double>
            ,"reltol":<double>
            ,"safety":<double>
            ,"skipDiscrete":<boolean>
        }
```

where entries after the first in the `ports` list and the attributes `abstol`, `reltol`, `safety` and `skipDiscrete` are optional. The name-value pairs have the following meaning.

- `ports:` Defines a set of values whose minimal and maximal value shall have a bounded difference. The set of values is defined by a non-empty array of strings. If one output port is provided, then the set of values comprises that output port's current value and its previous value. If at least two output ports are provided, then the set of values comprises the output ports' current values.

- `abstol`: This name-value pair is optional; it specifies the absolute tolerance. The stepsize calculator attempts to adjust the stepsize such that the absolute difference between the minimal and maximal value is smaller than the value of `abstol`. The default value for the absolute tolerance is 10^{-3}.

- `reltol`: This name-value pair is optional; it specifies the relative tolerance. The stepsize calculator attempts to adjust the stepsize such that the relative difference between the minimal and maximal value is smaller than the value of `reltol`. The default value for the relative tolerance is 10^{-2}.

- `safety`: This name-value pair is optional; it adjusts the conservatism of the algorithm that selects the next stepsize. The neutral default value is 0.0. If the variable stepsize calculator fails to keep the difference bounded (and the minimal stepsize is not the limiting factor), then the value of `safety` can be increased for more conservatism in the stepsize selection algorithm. Small negative values above $\alpha_{RISKY} - 1$, i.e. per default above -0.4 (see Table 3), are possible for less conservatism. Negative values below or equal to $\alpha_{RISKY} - 1$ lead to undefined behavior of the difference bin assignment algorithm (see Section 2.4).

- `skipDiscrete`: This optional name-value pair is by default set to `true`, i.e. the skipping over previous stepsizes that were limited by discrete constraints (see Section 2.7.3) is by default enabled. It may be disabled by setting this value to `false`.

1.6.3 Definition of a Sampling Rate Constraint

A constraint of `"type":"samplingrate"` is defined by

```
        "id":{
            "type":"samplingrate",
            "base":<integer>,
            "rate":<integer>,
            "startTime":<integer>
        }
```

with the following name-value pairs.

- `base`: Defines the exponent of 10 of the time base in seconds.

- `rate`: Defines the sample rate in multiples of 10^{base}.

- `startTime`: Defines the occurrence of the first sample hit in multiples of 10^{base}.

1.6.4 Definition of a FMU Max Step Size Constraint

A constraint of `"type":"fmumaxstepsize"` is defined by

```
        "id":{
            "type":"fmumaxstepsize"
        }
```

The constraint limits the step size to the minimum of the step size returned by `getMaxStepSize` from all instances that support the function.

1.7 The Simulate Command

The command is available at:
`http://localhost:8082/simulate/:session`

The command takes the following arguments and requires a JSON payload, Content-Type: application/json:

:session The session ID.

The Data payload:

```
        {
            "startTime":0.0,
            "endTime":10.1,
            "logLevels": {
                "{8c4e810f-3df3-4a00-8276-176fa3c9f001}.tank":
                ["logAll", "logError"],
                "{8c4e810f-3df3-4a00-8276-176fa3c9f000}.tank":
                ["logError"]
            }
        }
```

The payload contains the start and end time interval plus the log levels.
The command returns the following response on success:

```
        [
        {
            "status":"Finished",
            "sessionid": "1234"
        }
        ]
```

1.8 The Stop Simulation Command

This command sets a flag such that the simulation related to a given sessionID is stopped on completion of its current step.

The command is available at:
`http://localhost:8082/stopsimulation/:session`

The command takes the following arguments:

:session The session ID.

1.9 The Result Command

The command is available at:
`http://localhost:8082/result/:session/:type`

The command takes the following arguments:

:session The session ID.

:type Optional parameter. Possible parameters are: plain/zip and default is plain.

The command returns the following response on success. A response supports two return formats selected by the `:type` argument and indicated using the content type:

Content-Type: application/zip Returns a zip file containing the initialization data + start data + the result obtained during the simulation

Content-Type: text/plain Returns the result obtained during the simulation as text. The result is a CSV formatted string with: time, stepsize, and all outputs at that time

1.10 The Destroy Command

The command is available at:
`http://localhost:8082/destroy/:session`

The command takes the following arguments:

:session The session ID.

The command destroys a session and releases all resources bound to the session on success full termination.

1.11 Example

This sections exemplifies how to use the COE in terms of creating a configuration file and invoking the COE. Two FMUs are used in the example below: A sine FMU generating a sine wave and an integrate FMU that integrates the sine wave.

First the configuration file will be presented and described in section 1.11.1. Afterwards the HTTP requests required to run a simulation will be demonstrated and explained in section 1.11.2.

1.11.1 Configuration file

The configuration file is in JSON format, and the full file is shown below:

```
{
    "fmus":{
        "{integrate}":"./integrate",
        "{sine}":"./sine"
    },
    "connections":{
        "{sine}.sine.output":["{integrate}.inst1.input","{integrate}.inst2.
            input"],
        "{integrate}.inst2.output":["{integrate}.inst3.input"]
    },
    "parameters":{
        "{sine}.sine.amplitude":1
    },
    "algorithm":{
        "type":"fixed-step",
        "size":0.1
    },
    "livestream":{
        "{sine}.sine":["output"],
        "{integrate}.inst2":["output"]
    },
    "logVariables":{
        "{integrate}.inst3":["output"]
    }
}
```

FMUs JSON object

The `"fmus"` JSON object contains two name/value pairs: `"{integrate}":"./integrate"` and `"{sine}":"./sine"`. Note the `,` separating the pairs. `{integrate}` and `{sine}` are user-defined identifiers for the respective FMUs, they must not contain additional "{}", and they must be consistent throughout the configuration file, when referring to the respective FMUs. The COE accepts two different kind of paths: a path to a directory as in this example, or path to a zip-file with the extension ".fmu". If the integrate FMU was packaged in a zip-file with the extension ".fmu", the path would be `"./integrate.fmu"` instead of `"./integrate"`. The paths must be relative to the "COE.jar" file, which means the directories in this example are in the same directory as the "COE.jar" file.

Connections JSON object

The `"connections"` json object defines the relations between the FMUs. In this case there a two name/value pairs:

```
"{sine}.sine.output":["{integrate}.inst1.input","{integrate}.inst2.input"],
"{integrate}.inst2.output":["{integrate}.inst3.input"]
```

Once again, note the `,` separating the pairs. To address the FMU variables, the following format is used: `fmuReference.instancename.variablename`. This introduces the concept of "instances". Some FMUs can be used to create multiple instances. In the example above the integrate FMU is used to instantiate three instances called: `inst1`, `inst2`, and `inst3`. Therefore `"{integrate}.inst2.output"` means: The variable `output` of instance `inst2` in the FMU `{integrate}`.

1.11.2 Requests

2 Variable Stepsize Calculation

Three of the four constraint types (Zero Crossing, Bounded Difference, and Sampling Rate) are defined in a JSON file that is posted to the COE with the initialize command (see Section 1.6), and one (FMU-requested) is requested by the simulated FMUs.
After initialization, the variable stepsize calculator holds a set of constraint handlers. Each handler is responsible for one constraint. When asked for the next stepsize by the COE, the variable stepsize calculator asks each handler for the next stepsize and returns the minimum of these values.

2.1 Interface with the Master Algorithm

The variable stepsize calculator is called by the master algorithm before each `doStep`. It is given by the master algorithm the current time, the previous stepsize, the current output values, and the (estimated) output derivatives of the FMUs. The variable stepsize calculator returns to the master algorithm the next stepsize.

After a `doStep`, the master algorithm asks the variable stepsize calculator to validate the taken step, i.e. to check whether any constraints have been violated. If that is the case, a warning is issued. If all FMUs support rollback, a rollback is initiated and the master algorithm asks the variable stepsize calculator for a new, reduced stepsize.

The algorithm for derivative estimation, see Section 2.3.2, has been moved from the variable stepsize calculator to the COE. This is done so that the master algorithm may estimate derivatives and supply these to FMUs that have the capability canInterpolateInputs. To be clear, if the FMU that supplies these signals also provides derivatives, these are used, but if that FMU has `maxOutputDerivativeOrder=0` (or `<=1` in the case of second order input derivatives]), the estimated values are used.

2.2 Constraint Types

There are four constraint types:

- Zero Crossing
- Bounded Difference
- Sampling Rate
- FMU Max Step Size

The constraints are defined in the JSON file (see Section 1.6). The fourth constraint, FMU Max Step Size, was enabled by default until COE version 0.2.14. See Section 2.6 for more info on the FMU Max Step Size Constraint.

2.3 Zero Crossing Constraints

A zero crossing constraint is a continuous constraint. A zero crossing occurs at the point where a function changes its sign. In simulation, it can be important to adjust the stepsize such that a zero crossing is hit (more or less) exactly. For instance, a ball should rebound from a wall exactly when the distance between the ball and the wall hits zero and not before or after that. A solver in a tool such as Simulink can adjust the stepsize using iterative approaches, but in a co-simulation a rollback of the participating models' internal states is in general not possible or efficient. Hence, the variable stepsize calculator bases its stepsize adjustments on the *prediction* of a future zero crossing.

2.3.1 Extrapolation

To predict a future zero crossing, the zero crossing function f must be extrapolated.
For first order extrapolation, the following calculation is used:
$f(t+\Delta t) = f(t) + \dot{f}(t)\Delta t$
For second order extrapolation, the following calculation is used:
$f(t+\Delta t) = f(t) + \dot{f}(t)\Delta t + 0.5\ddot{f}(t)\left(\Delta t\right)^2$

2.3.2 Derivative Estimation

The derivatives $\dot{f}(t)$ and $\ddot{f}(t)$ are either provided by the FMUs (if the capability `maxOutputDerivativeOrder` is high enough), or estimated. For first order extrapolation, the last two data points are used to estimate the first derivative. For second order extrapolation, either the last three data points are used to estimate the first and second derivate, or, if the FMU provides the first but not the second derivative, the last two data points and their first derivatives are used to estimate the second derivative.

2.3.3 Extrapolation Error Estimation

Extrapolation will generally incur an extrapolation error; the variable stepsize calculator estimates that error based on past extrapolation errors. After completion of a time step, the variable stepsize calculator compares the actual value x of the zero crossing function f with the value $\hat{x}$ that was predicted one time step earlier. The estimated extrapolation error $\hat{\epsilon}$ follows:

$$\epsilon \leftarrow \left\{ \begin{array}{ll} \alpha\hat{\varepsilon} + (1-\alpha)\,|x-\hat{x}| & \text{if } \hat{\varepsilon} > |x-\hat{x}| \\ |x-\hat{x}| & \text{otherwise} \end{array} \right\} \tag{1}$$

For example, it decreases slowly ($\alpha = 0.7$) with a first order IIR-filter rule when the extrapolation error becomes smaller, and rises abruptly to the actual value when the extrapolation error becomes larger.

2.3.4 Estimation of the number of timesteps to a zero crossing

The variable stepsize calculator (conservatively) estimates the number of timesteps n to hit the predicted zero crossing $f(t_{ZC}) = 0$ at time t_{ZC}, when starting from the current time t (with $t \leq t_{ZC}$) and when keeping the current stepsize Δt constant, to:

$$n = \frac{t_{ZC} - t}{\Delta t} \cdot \frac{1}{1 + \hat{\varepsilon} + \sigma} \tag{2}$$

where $\hat{\varepsilon}$ is the estimated extrapolation error and σ the (additional) level of conservatism optionally specified by the attribute `safety` in the JSON config file.
The rationale of this equation is that the left term predicts the zero crossing exactly when the zero crossing function f is, in the case of first order extrapolation, a straight line, or, in the case of second order extrapolation, a straight line or second order parabola. An extrapolation error generally occurs for all other functions f, with the danger of overestimating n and thus potentially choosing a too large stepsize (that steps over the zero crossing with the consequence that the tolerance of the zero crossing may be violated). Therefore, n is conservatively underestimated. The degree of this conservatism is defined by the second term and depends on both the (time-varying) estimated extrapolation error $\hat{\varepsilon}$ and the (constant) value of the safety attribute σ.

2.3.5 Detection of unstable oscillations

Unstable oscillations around the zero crossing are detected by monitoring the last three data points and checking whether these lie on alternating sides of the zero crossing and increase in absolute value.

2.3.6 Stepsize adjustment strategy

The chosen stepsize Δt is in most cases determined by a factor ρ that is multiplied with the previous stepsize Δt_{prev} (and saturated to lie within the specified stepsize interval). The stepsize is said to be *adjusted to hit* the zero crossing when $\rho = n$ (for $n \leq 1$). The stepsize is said to be *tightened* when $\rho = TIGHTENING\_FACTOR$. The stepsize is *held constant*, when $\rho = 1$. The stepsize is said to be *relaxed* when $\rho = RELAXATION\_FACTOR$. The stepsize is said to be *strongly relaxed* when $\rho = STRONG\_RELAXATION\_FACTOR$. The default values for these factors are listed in Table 1.
By inspecting the last two data points, the direction of the simulated trajectory with respect to the zero crossing can be either:

Table 1: Default values for the stepsize adjustment factors.

$TIGHTENING\_FACTOR$	0.5
$RELAXATION\_FACTOR$	1.2
$STRONG\_RELAXATION\_FACTOR$	3.0

- *distancing zero crossing*,
- *approaching zero crossing* or
- *crossed zero.*

When *distancing a zero crossing*, the stepsize is *strongly relaxed.*
When *approaching a zero crossing*, the current value of the zero crossing function, $f(t)$, is compared to the value of the absolute tolerance, $abstol$.
If:

$$|f(t)| \leq abstol \cdot TOLERANCE\_SAFETY\_FACTOR \tag{3}$$

where $TOLERANCE\_SAFETY\_FACTOR \leq 1.0$ and a default value of 0.5, then $f(t)$ is said to be *well within tolerance*, and the stepsize is *relaxed* (the zero crossing has not yet occurred but is already precisely resolved).
If

$$|f(t)| \leq abstol \tag{4}$$

then $f(t)$ is said to be *within tolerance*, and the stepsize is *held constant* (the zero crossing has not yet occurred but is already resolved).
If

$$|f(t)| > abstol \tag{5}$$

then $f(t)$ is said to be *outside tolerance*, and the (conservatively) estimated value for the number of timesteps to hit the predicted zero crossing, n, is considered.

- If $n \leq 1$, then the stepsize is *adjusted to hit* the zero crossing.
- If $1 < n \leq \delta_{tighten}$, then the stepsize is *tightened.*
- If $\delta_{tighten} < n \leq \delta_{relax}$, then the stepsize is *held constant.*
- If $\delta_{relax} < n \leq \delta_{stronglyrelax}$, then the stepsize is *relaxed.*
- If $\delta_{stronglyrelax} < n$, then the stepsize is *strongly relaxed.*

The default values of the parameters δ_i are listed in Table 2.

When the simulated trajectory *crossed zero* in the previous time step, it is checked whether or not unstable oscillations around the zero crossing are building up.

- If unstable oscillations occur, and $f(t)$ is *well within tolerance*, then the stepsize is *held constant.*
- If unstable oscillations occur, and $f(t)$ is *within tolerance*, then the stepsize is *tightened.*

Table 2: Default values of the distance bin separators for the number of timesteps to hit a predicted zero crossing.

$\delta_{tighten}$	1.8 ($= 1.5 \cdot RELAXATION\_FACTOR$)
δ_{relax}	3.0 ($= STRONG\_RELAXATION\_FACTOR$)
$\delta_{stronglyrelax}$	30.0 ($= 10.0 \cdot \delta_{relax}$)

- If unstable oscillations occur, and $f(t)$ is *outside tolerance*, then the stepsize is set to its minimal value.
- If unstable oscillations do not occur, and $f(t)$ is *well within tolerance*, then the stepsize is *relaxed.*
- If unstable oscillations do not occur, and $f(t)$ is *within tolerance*, then the stepsize is *held constant.*
- If unstable oscillations do not occur, and $f(t)$ is *outside tolerance*, then the stepsize is *tightened.*

The final three reactions (when unstable oscillations do not occur) are somewhat conservative, with the intention of discouraging possible oscillations around the zero crossing from developing. Therefore, the stepsize immediately after the zero crossing is kept small. Altogether, these are the 14 possible reactions of the variable step size calculator to exhaustively handle a zero crossing constraint.

2.4 Bounded Difference Constraints

A bounded difference constraint is a continuous constraint. A bounded difference ensures that the minimal and maximal value of a set of values do not differ by more than a specified amount (the underlying assumption is that this difference becomes smaller when the stepsize is reduced). For the definition of a bounded difference constraint in the JSON file, see Section 1.6.2.
The capability to impose a bounded difference can be useful in co-simulation, for instance, in the calculation of the heat exchange between model A of temperature T_A and model B of temperature T_B. Here, at least one of the models must calculate the heat flow, which is a function of both T_A and T_B. The model that calculates the heat flow, say model A, knows its own temperature T_A but only has a view, $T_{B,view}$, on model B's true temperature T_B. To bound the error of the calculated heat flow, a bounded difference between T_B and $T_{B,view}$ is imposed.
The bounded difference problem is distinct from the zero crossing problem in that there is not a specific time *instant* (the zero crossing) to hit, but rather a specific time *difference* (the stepsize that keeps the difference bounded).
To choose the next stepsize, the current absolute and relative differences between the minimal and maximal values, δ_A and δ_R, are calculated and compared to the absolute and relative tolerances, ε_A and ε_R, respectively. Based on this comparison, the absolute and relative differences are each assigned to one of five distance bins. The bins are determined with the safety factor:

$$\sigma = \frac{1}{1 + safety}, \tag{6}$$

where $i = A, R$, and with the default values of the parameters

$$\alpha_{SAFE} \leq \alpha_{TARGET} \leq \alpha_{RISKY} \leq 1 \tag{7}$$

listed in Table 3.

Table 3: Default values of the parameters used in the distance bin assignment of the bounded difference algorithm.

α_{RISKY}	0.6
α_{TARGET}	0.4
α_{SAFE}	0.2

If:

- $\delta_i > \varepsilon_i$, then the difference i is assigned to the VIOLATION bin.
- $\varepsilon_i \geq \delta_i > \varepsilon_i \sigma \alpha_{RISKY}$, then the difference i is assigned to the RISKY bin.
- $\varepsilon_i \sigma \alpha_{RISKY} \geq \delta_i > \varepsilon_i \sigma \alpha_{TARGET}$, then the difference i is assigned to the TARGET bin.
- $\varepsilon_i \sigma \alpha_{TARGET} \geq \delta_i > \varepsilon_i \sigma \alpha_{SAFE}$, then the difference i is assigned to the SAFE bin.
- $\varepsilon_i \sigma \alpha_{SAFE} \geq \delta_i$, then the difference i is assigned to the SAFEST bin.

Of the two assigned distance bins, the less safe one (the one ranking higher in the bullet list above) is chosen. If this distance bin is the

- VIOLATION bin, then the stepsize is *strongly tightened.*
- RISKY bin, then the stepsize is *tightened.*
- TARGET bin, then the stepsize is *held constant.*
- SAFE bin, then the stepsize is *relaxed.*
- SAFEST bin, then the stepsize is *strongly relaxed.*

A *strongly tightened* stepsize means that $\delta = STRONG\_TIGHTENING\_FACTOR$ with default value 0.01 is multiplied with the previous stepsize $(\Delta t)_{prev}$ to obtain the next stepsize Δt. The meaning of the other stepsize adjustments is analogous to the implementation of the zero crossing algorithm (see Section 2.3.6). The chosen stepsize is saturated to the stepsize interval.
This algorithm for the bounded difference handler tries to adjust the stepsize such that it is kept within the TARGET bin throughout the simulation. Because a variable stepsize calculator in a co-simulation cannot (efficiently) obtain the stepsize through an iterative approach, it needs to make fairly sure that the stepsize it selects does not lead to a tolerance violation. The stepsize calculation must therefore be somewhat conservative, which is essentially manifested in the RISKY bin as a buffer between the TARGET and VIOLATION bins.
On the safe side of the TARGET bin, two bins must exist. The SAFE bin has an associated relaxation factor that is small enough so that a stepsize relaxation should not lead to an overshoot of the bound difference beyond the TARGET bin in the next time step. The SAFEST bin has an associated strong relaxation factor that is equal to the strong relaxation factor used by all other continuous constraints to prevent interference between continuous constraints (see Section 2.7.1).
Note that the above described algorithm of the Bound Difference handler is extended below to prevent interference by discrete events (see Section 2.7.3).

2.5 Sampling Rate Constraints

A sampling rate constraint is a discrete constraint. It constrains the stepsize such that repetitive, predefined time instants are exactly hit. This can be useful in co-simulation, for instance, when a modeled control unit reads a sensor value every x milliseconds. For the definition of a sampling rate constraint in the JSON file, see Section 1.6.3.
The chosen stepsize is either the time difference between the current time and the time instant of the next sampling, or the maximal stepsize, whichever is smaller. Note that the minimal stepsize may be violated to hit a sampling event.

2.6 FMU Max Step Size Constraints

The FMU Max Step Size constraint limits the step size to the value returned from an FMU if the function is supported by the FMU. A proposal is underway to extend the FMI standard with the procedure:

```
fmi2Status fmi2GetMaxStepSize(fmi2Component c, fmi2Real *maxStepSize);
```

This means that an FMU can report in advance the maximal stepsize that it will accept in the next time step. The variable stepsize calculator queries all FMUs for these stepsizes and uses the minimum of the reported values as upper bound for the next stepsize. The implementation in the COE is based on the principle presented in [1, 2] for *Master-Step With Predictable Step Sizes*. To the authors knowledge, this feature is implemented in FMUs exported from the tools: 20-sim, OpenModelica and Overture.

2.7 Interference between constraint handlers

When multiple constraints are present, their handlers may interfere with each other in the sense that one constraint may become active only because another one has been active in the previous step. Measures are taken to counter such interference.

2.7.1 Interference between continuous constraints handlers

Interference between continuous constraint handlers occurs when:

1. In one time step, Constraint A is active (i.e. constrains the stepsize);
2. In the next time step, the handler for Constraint A relaxes the stepsize by a factor $\rho_A > 1$, and
3. Constraint B becomes active – not because its handler protects against a potential violation, but only because it cannot relax the stepsize by more than a factor $\rho_B < \rho_A$.

To prevent such interference, all continous contraints must have the same value for their respective maximal relaxation factors. Therefore, in the implementation of the variable stepsize calculator, $STRONG\_RELAXATION\_FACTOR$ is the maximal relaxation factor for both Zero Crossing and Bounded Difference constraints and defined in the scope of the whole calculator – not in the scope of individual constraints (as other factors are). When constraints *relax strongly*, $STRONG\_RELAXATION\_FACTOR$ is used[1].

[1]Strictly speaking, when all continuous constraints *relax strongly* with the same relaxation factor, they all become active. The important point is that none of them slows down the relaxation process unnecessarily by relaxing less than the others.

2.7.2 Interference between discrete constraints handlers

Discrete constraints handlers base their stepsize requirements on independent time instants and therefore do not interfere with each other.

2.7.3 Interference between discrete and continuous constraint handlers

When a discrete constraint handler has limited the stepsize in the previous step, the question arises how a continuous constraint handlers shall proceed with its calculation of the next stepsize. The situation that shall be avoided is this: all continuous constraint handlers would allow a large stepsize, but a discrete constraint handler enforces a sudden, strong reduction of the stepsize. In the steps that follow, there are no discrete events, but the continuous constraint handlers require potentially many steps to repeatedly *strongly relax* the stepsize until it becomes large again.
The solution to this problem is different for Zero Crossing and Bounded Difference constraint handlers.

Extension of the Zero Crossing handler To prevent the above described undesired situation, Zero Crossing handlers calculate the next stepsize based on the last stepsize *that was not limited by a discrete constraint.*
To be precise, a Zero Crossing handler uses the previous data points irrespective of the previously active constraints to calculate the extrapolation. But when it calculates the next stepsize, it discards all previous stepsizes that were limited by a discrete constraint and chooses the last stepsize that was limited by a continuous constraint. With the thus chosen previous stepsize (and the result of the extrapolation), the handler calculates the factor ρ that is multiplied to the chosen previous stepsize in order to obtain the next stepsize. With this approach, introduced discrete events do not markedly affect the tightening and relaxation of the stepsize selected by a Zero Crossing handler.
This approach is safe, in the sense that a zero crossing should not be crossed prematurely, for two reasons. First, introduced discrete events always shorten the stepsize when approaching the zero crossing, which is conservative. Second, the assumed previous stepsize may be larger than the true previous stepsize (that was limited by a discrete constraint handler), but this does no harm: The calculation of the next stepsize is based on the number of timesteps to the predicted zero crossing, n, with the assumption that the (assumed) previous stepsize is held constant. When the previous stepsize is larger, n becomes smaller, favoring a stronger tightening of the next stepsize in particular close to the zero crossing, where the stepsize is *adjusted to hit.*
Essentially, the Zero Crossing handler can safely ignore previous stepsizes that were limited by discrete constraints because it needs to hit a time *instant* (i.e. the zero crossing) and that time instant does not depend on the previous stepsizes (time *differences*). The situation is different for the Bounded Difference handler.

Extension of the Bounded Difference handler Whereas the Zero Crossing handler needs to hit a time *instant* (i.e. the zero crossing) that does not depend on the previous stepsizes (time *differences*), the Bounded Difference handler needs to limit a value difference that does depend on the stepsize. When the Bounded Difference handler notices that the previous stepsize was limited by a discrete constraint, it may proceed in either of two ways.
First, the Bounded Difference handler could simply go forward as usual (i.e. it calculates the next stepsize by scaling the previous stepsize by the factor that is associated with the determined diffe-rence bin). Because the previous stepsize was limited by a discrete event and was therefore

shorter than the stepsize that the Bounded Difference handler would have chosen, this strategy will frequently lead to the stepsize being *relaxed* or *strongly relaxed*.
Second, the Bounded Difference handler could take the last stepsize that was limited by a continuous constraint and repeat the decision it made then *on that stepsize*. To prevent that a repeated decision overly relaxes the stepsize, the repeated decision will *hold* the stepsize *constant* whenever the past decision was to *relax* or *strongly relax* it. To prevent that a repeated decision overly tightens the step-size, the chosen next stepsize may never be smaller than the one obtained with the above (usual) strategy.
By default, the second strategy is enabled. However, in rare cases that strategy may lead to a tolerance violation (a chain of discrete events could carry a past decision to *hold* the stepsize *constant* through time; when the chain of discrete events stops, the stepsize will be *held constant* in the next step but it might have needed to be *tightened* instead). Therefore, it is possible to disable the second strategy by setting the optional attribute `"skipDiscrete"` to `false` in the definition of the Bounded Difference constraint in the JSON configuration file (see Section 1.6.2). When the second strategy is disabled, an active discrete constraint will likely reduce the next stepsize(s) proposed by the Bounded Difference constraint handler, potentially reducing efficiency.

2.8 Logging

The variable stepsize calculator writes to the same log as the COE.
When a step is taken with maximal stepsize, the variable stepsize calculator produces no log output.
When a step is taken with a less than maximal stepsize, the variable stepsize calculator logs the identifiers of the active constraints and the action of their handlers. For instance, a log entry would read

```
Time 0.9499999999999998, stepsize 0.09, limited by constraint
"bd" with decision to hold the stepsize constant
(absolute difference within target range)
```

When all continuous constraints relax strongly, the log entry does not list all constraints but is shortened to:

```
Time 5.000458745644138, stepsize 9.536808544011453E-4,
all continuous constraint handlers allow strong relaxation}
```

When a Zero Crossing constraint handler detects a zero crossing, it produces a log entry which would read:

```
A zerocrossing of constraint "zc" occurred in the time interval
[ 14.999971188014648 ; 15.000117672389647 ] and was hit
with a distance of 0.18103104302103257
```

When the variable stepsize calculator detects that a constraint has been violated in the previous step, it logs a warning. For instance, such a warning would read:

```
Absolute tolerance violated!
        | A zerocrossing of constraint "zc"
        | occurred in the time interval [ 4.998123597131701 ; 5.008123597131701 ]
        | and could only be resolved with a distance of 11.789784201164633
        | which is greather than the absolute tolerance of 1.0
        | The stepsize equals the minimal stepsize of 0.01 !
        | Decrease the minimal stepsize
        or increase this constraint's   tolerance}
```

A Variable stepsize

A.1 Variable step size configuration example

The variable stepsize calculator is selected and configured in a JSON config file. This file is posted to the COE with the initialize command (see Section 1.6). An example JSON config file follows.

```
{
    "fmus":[
        "{MassAFMU}":"src/test/resources/varsteptest/
            VarStepSolverTest_FMUs_MassA",
        "{MassBFMU}":"src/test/resources/varsteptest/
            VarStepSolverTest_FMUs_MassB",
        "{SineFMU}":"src/test/resources/varsteptest/VarStepSolverTest_FMUs_Sine
            ",
        "{SinkFMU}":"src/test/resources/varsteptest/VarStepSolverTest_FMUs_Sink
            "
    ],
    "connections":{
        "{MassAFMU}.massA.QA":[
            "{MassBFMU}.massB.QA",
            "{SinkFMU}.sink.QA"
        ],
        "{SineFMU}.sine.EXTSIG":[
            "{MassBFMU}.massB.EXTSIG",
            "{SinkFMU}.sink.EXTSIG"
        ],
        "{MassBFMU}.massB.TB":[
            "{MassAFMU}.massA.TB",
            "{SinkFMU}.sink.TB"
        ],
        "{MassAFMU}.massA.TB_used":[
            "{SinkFMU}.sink.TB_used"
        ]
    },
    "parameters":{
        "{MassBFMU}.massB.B.C":15000.0,
        "{MassAFMU}.massA.A.C":120000.0
    },
    "algorithm":{
        "type":"var-step",
        "size":[
            1E-6,
            1.0
        ],
        "initsize":1E-4,
        "constraints":{
            "zc":{
                "type":"zerocrossing",
                "ports":[
                    "{SineFMU}.sine.EXTSIG"
                ],
                "order":2,
                "abstol":1.0,
                "safety":0.0,
            },
            "bd":{
                "type":"boundeddifference",
                "ports":[
                        "{MassBFMU}.massB.TB",
                        "{MassAFMU}.massA.TB_used"
                    ],
                    "abstol":1.0,
                    "reltol":1.0,
                    "safety":0.0,
                },
                "sr":{
                    "type":"samplingrate",
                    "base":-1,
                    "rate":10,
                    "startTime":15
                }
            }
        }
}
```

B Program properties

The COE program flow can be changed using the following Java properties which can be set on program launch using `-D` followed by the property and a value can be specified after an equal sign:

`simulation.program.delay.enable` If this property is set to `true` then the COE will interpret the time step size in seconds and make sure that the steps are at least separated by a program delay of that time step size. It only introduces a delay if the execution of `doStep` on all FMU instances are faster than the requested time step size.

`fmi.instantiate.with.empty.authority` This inserts an empty authority into the URI sent to the FMU in `instantiate`. E.g. `file:/` will become `file:///`. This can be necessary for faulty FMU implementations.

`coe.fmu.custom.factory` This must be given a fully qualified name to a class which implements `org.intocps.orchestration.coe.IFmuFactory` if set this class will be instantiated and ask to handle FMU creation before the internal factory. This can thus be used to override the default behaviour.

`coe.livestream.filter` Limits the time resolution on the values send using live logging. The values will only be send at the given interval or greater.

`simulation.parallelise.resolveinputs` Run all per instance input actions in parallel.

`simulation.parallelise.setinputs` Run all per instance set inputs in parallel.

`simulation.parallelise.dostep` Run all per instance `doSte` calls in parallel.

`simulation.parallelise.obtainstate` Run all per instance calls to obtain a global state in parallel.

`simulation.profile.executiontime` Log execution time for the main calls involved in performing a global step.

Note that enabling parallel execution may not give faster simulations it is highly dependent on the number of instances, the amount of inputs/outputs and the actual execution to of the `doStep` call per instance.

References

[1] D. Broman, C. Brooks, L. Greenberg, E.A. Lee, M. Masin, S. Tripakis, and M. Wetter. Determinate composition of FMUs for co-simulation. In *Embedded Software (EMSOFT), 2013 Proceedings of the International Conference on*, pages 1–12, 2013.

[2] Fabio Cremona, Marten Lohstroh, David Broman, Marco Di Natale, Edward A. Lee, and Stavros Tripakis. Step revision in hybrid co-simulation with FMI. In *MEMOCODE*, pages 173–183. IEEE, 2016.