D3.5	**IEC 61131-3 code simulation for Software-in-the-loop PLC testing with EQUA tools**
Access[1]:	**PU**
Type[2]:	**Prototype**
Version:	**0.4**
Due Dates[3]:	**M24, M36**

Open Cyber-Physical System Model-Driven Certified Development

Executive summary[4]:

The purpose of this work is to develop a prototype simulation system for efficient simulation of physical systems controlled by discrete-time controllers that are described using the standard PLC programming set of languages: IEC 61131-3. Using the same actual control code for simulation as for physical deployment is expected to reduce the number of errors that are introduced in the final control code with respect to present state-of-the-art, when continuous function blocks are manually transcribed in the PLC programming system.

In this M36 final-edition of the report, the structure of the system is described along with the implemented numerical methods. The report concludes with an evaluation of the prototype system for software in the loop PLC testing with EQUA tools.

[1] Access classification as per definitions in PCA; PU = Public, CO = Confidential. Access classification per deliverable stated in FPP.
[2] Deliverable type according to FPP, note that all non-report deliverables must be accompanied by a deliverable report.
[3] Due month(s) according to FPP.
[4] It is mandatory to provide an executive summary for each deliverable.

Deliverable Contributors:

	Name	Organisation	Primary role in project	Main Author(s)[5]
Deliverable Leader[6]	EQUA	EQUA Simulation AB	T5.4 lead	
Contributing Author(s)[7]	Per Sahlin	EQUA Simulation AB		X
	Patrik Skogqvist	EQUA Simulation AB		
	Markus Högberg	EQUA Simuation AB		
Internal Reviewer(s)[8]	Lena Buffoni	LIU	WP3 lead	

Document History:

Version	Date	Reason for Change	Status[9]
0.1	09/11/2017	First Draft Version	Draft
0.2	28/11/2017	First M24 milestone issue	Released
0.3	23/11/2018	Updates for M36 milestone	In Review
0.4	04/12/2018	Modifications based on review	Released

[5] Indicate Main Author(s) with an "X" in this column.
[6] Deliverable leader according to FPP, role definition in PCA.
[7] Person(s) from contributing partners for the deliverable, expected contributing partners stated in FPP.
[8] Typically person(s) with appropriate expertise to assess deliverable structure and quality.
[9] Status = "Draft", "In Review", "Released".

CONTENTS

ABBREVIATIONS

List of abbreviations/acronyms used in document:

Abbreviation	**Definition**
BPS	Building performance simulation
HIL	Hardware-In-the-Loop
HVAC	Heating ventilation and air conditioning
Hybrid DAE	Differential-Algebraic Equations with discrete events and hysteresis
IDA ICE	IDA Indoor Climate and Energy – an EQUA product
IDA SE	IDA Simulation Environment – an EQUA product
IDE	Integrated (IEC 61131-3) Development Environment
NMF	Neutral Model Format – a simple modelling language for Hybrid DAEs
PLC	Programmable logic controller
SIL	Software-In-the-Loop

1 INTRODUCTION

Many industrial processes and plants are controlled by so called PLC:s, i.e. dedicated processors with the sole purpose of running a control algorithm. PLC:s are programmed using a series of connected languages that have been standardized in IEC 61131-3:

- Ladder diagram (LD), graphical
- Function block diagram (FBD), graphical
- Structured text (ST), textual
- Instruction list (IL), textual
- Sequential function chart (SFC), graphical.

The objective of this work is to develop functionality for efficient simulation of IEC 61131-3 PLC code in conjunction with large physical system models.

Study of complex control strategies plays an increasing role in building and tunnel design. (EQUAs two main areas of application.) Discrepancy between the intentions of the designer, often expressed as non-formalized control laws, and the as-built implementation is a frequent source of malfunction and energy waste.

With a control object as costly, cumbersome, and slow as a building or tunnel, thorough testing of algorithms with respect to a real object is only rarely possible. A single seasonal storage borehole strategy, for example, may take a few years to evaluate in real time. The only option is to rely on a simulation model with all relevant systems, a model close enough to reality to allow off-line testing of all important control modes.

Within an equation-based simulation framework, the behavior of both the physical object and the control system can be modeled. Continuous-time models of all standard functional blocks allow construction of controllers with the same topology and parameters as those employed in the physical controller. This allows realistic development of any control strategy that can be formulated by basic control blocks.

In addition to the obvious advantage of having the same control description for both simulation and hardware deployment, there are two fundamental reasons for simulating complete discrete-time control programs in the context of a simulated building or tunnel:

- Some complex control schemes cannot conveniently be expressed exclusively by “extended” equations. Truly algorithmic descriptions are sometimes needed. An example is model predictive control, where a simulation model of the control object itself is exercised by the control algorithm in order to find an optimal control scheme.

- Quality assurance. To find programming errors, a testing environment as close as possible to the as-built situation is desirable. In addition, effects that stem from the limited sampling rate of the real implementation can only be investigated in the correct time scale.

One way of studying the behavior of as-built control programs with respect to a simulated building or tunnel is hardware-in-the-loop simulations, where a physical controller is interacting with a simulated control object. Although often valuable as a last step before physical commissioning, this approach is cumbersome since experiments normally must be carried out in close to real time.

A more attractive option is to execute the actual control code much faster than real time while retaining the realistic algorithmic and sampling behavior of the physical controller. In its most straightforward implementation, this implies simulation of both controller and control object at the sampling rate of the controller, i.e. a global simulation time step is taken for each controller sample. For a building, this normally yields excessive simulation times, since a typical controller operates with an unsuitably short step (possibly less than a second). Below, in Section 2, an IDA Simulation Environment (IDA SE) implementation of discrete-time controllers is presented that enables multi-rate simulation, i.e. a short fixed time step is used for the simulated controller, while a physically motivated much longer (and variable) step is used for the building

The first requirement for multi-rate simulation of discrete-time controllers is obviously to have a numerical method that enables this. A major part of the development of this method was carried out during the OPENPROD ITEA2 project. In OPENCPS, focus is instead on the automatic generation of C code that can be linked to the simulator.

The original plan for OPENCPS T3.3 was to develop a C code generator for IEC 61131-3 Structured text and Function block diagrams. However, during the initial stages of the work it was discovered that an open source implementation of a Structured text to C translator already exist: the MATIEC compiler. Associated with this is also an open source, complete IEC 61131-3 programming system: Beremiz. Focus was then shifted into evaluation of the usefulness of these tools and an analysis of how the generated code could be adapted to simulation.

In the next section, the numerical method of simulation for combined discrete and continuous systems is described.

2 DISCRETE-TIME ALGORITHMIC CONTROLLERS IN IDA SIMULATION ENVIRONMENT

Often, when discrete-time controllers are simulated within a basically continuous system simulator, the simulation is restarted for each sample of the controller, i.e. the (fixed) time step of the controller is used for all submodels. If the dynamics of the physical system are such that this time step is fairly well suited to resolve relevant transients, this will yield acceptable simulation performance. However, this is rarely the situation for buildings. On average, a suitable time step for a whole-year, whole-building building simulation will be at least a few minutes. In a variable time step environment, steps of several hours may be taken during the night, when little is happening in the system. Meanwhile, a typical PLC will repeatedly execute its program at full speed rendering a sampling interval on the order of seconds. Buildings and similar objects that are slow with respect to typical sampling times will clearly require special methods. Such a method has been developed for IDA SE and it will be presented in the next few sections.

2.1 Multi-rate simulation method

The method described has been implemented in IDA SE and can presently be applied either by direct interactive construction of controllers using discrete time block libraries (not yet described by IEC 61131-3) or by incorporating a controller that is described in, for example, C.

2.1.1 Solver characteristics

The numeric solver used in IDA SE is a variable step and variable order prediction-corrector solver based on the MOLCOL methods (Dahlquist, 1983), a generalization of the implicit BDF methods. In each normal time step, the future development of all continuous variables is predicted using a polynomial extrapolation. A solution of the system of equations is then calculated by a modified Newton-Raphson method, outlined next.

The global system Jacobian is assembled and factorized. The predicted values are inserted in the equations and residuals are calculated and used as right-hand side when solving the linear system of equations to get a correction vector. The scheme is iterated until residuals and corrections become sufficiently small. Jacobians are not necessarily computed in each time step, just when convergence is poor.

The predictor-corrector step may fail, either due to lack of convergence, or, because the local truncation error of the difference approximation is deemed too large. This error depends on time step size and on the order used, and can be expressed in terms of the difference between the predicted and calculated solutions. In either case, the time step is reduced and a new prediction is calculated. After a successful step, the local truncation error is used to control the order and the tentative length of the next time step.

When the integration is started, an initial value calculation is made to find a start solution to the equations. An initial value calculation will also be made when a discontinuity is met in driving data or an abrupt event is signaled by a component model.

2.1.2 Interface to programmable sampling components

IDA SE handles component models written in either the Neutral Model Format (NMF) or IDA Modelica. The component models are automatically translated to Fortran or C and linked into Windows dynamically linked libraries (DLLs), i.e. individual component models are normally pre-compiled.

To incorporate an arbitrary discrete-time programmable controller in an IDA simulation, it has to meet some basic requirements:

- It must be written in a language that can be translated to a Windows DLL.
- It may have internal memory and internal states that should be preserved between activations. However, since the solver will need to rerun sequences of sampling steps, the internal states must be stored in an array that is accessible from the solver.

The interface between solver and sampling component has been implemented as a shell model. The shell delivers inputs to, and fetches output from the discrete component. It also provides memory space for the internal states of the sampling component.

2.1.3 Connecting continuous and sampling pre-compiled components

The solver normally divides the simulated system into three distinct sections: The central aggregate of continuous components (equation-based), an algorithmic section preprocessing input data and providing input to the continuous section, and an algorithmic section performing post-processing of simulation results. In each time step, the sections are processed in order pre, central, post.

The components making up the system are linked together by links (signal aggregates) that can be directed (causal) or undirected (acausal). Within the central section, acausal links are permitted, while in pre and post sections all links must be causal (inputs of one component connected with outputs of another).

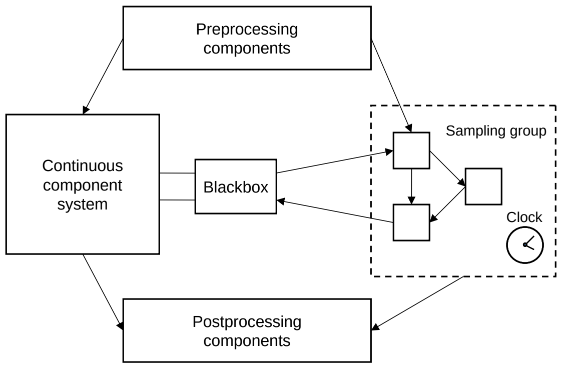

Figure 1. Component categories.

When sampling components are added to this setup, they change the pattern. The sampling components will only be connected by causal links. They are allowed to take input from pre and central sections and to deliver output to central and post sections. The pre and post sections will retain their positions first and last in the processing chain, but the central section will be interacting intimately with the sampling components. In this interaction, the sampling components will fetch input from the central section and deliver output back to the central section. See Figure 1.

2.1.4 Groups of sampling components

In the current implementation, each sampling component is connected to a clock, defining a constant sampling rate. Several components may be combined into a sampling group, provided that they use the same clock, and that they are connected by causal links into a directed network with a defined execution order. The solver will always activate the group as a whole.

2.1.5 Typical integration algorithm

The rationale behind the implementation is the possibility to use multi-rate integration, with the central system taking steps much longer than the sampling steps. This is possible if the sampling outputs during longer periods appear as continuous, differentiable signals.

A typical global integration step will progress as follows (see Figure 2). A prediction is calculated for the continuous system, including the variables that are connected to sampling components. In a simple case with a single controller, these variables could be e.g. a temperature sent to the sampling component and a control signal coming back. A more complex case could include several sampling groups, with possibly different sampling rates, and each having multiple inputs and outputs.

Integration time is advanced through the global step and the sampling groups are executed at their respective intervals. For each such sampling execution, input signals are interpolated in the prediction. The outputs from the sampling groups are compared with the interpolated predictions for the receiving continuous variables. The sample stepping continues to the end of the global step, unless a too large discrepancy develops underway. In the latter case, the latest sampling execution is cancelled, and the global step is truncated prior to the divergence.

Next, the global system is solved with Newton iteration, and the accuracy is checked. This scheme is based on the assumption that, as long as the prediction is good enough to provide acceptable truncation error in the continuous equations, the sampling steps, run against the same prediction, can also be accepted as they are, without update to match the corrected continuous solution. Some problems related to this assumption appear, and the remedies taken are discussed below.

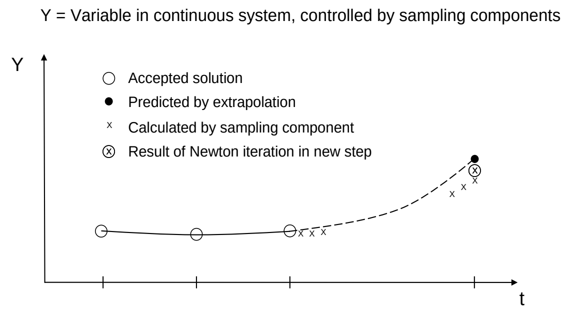
Y = Variable in continuous system, controlled by sampling components
Y
○ Accepted solution
● Predicted by extrapolation
× Calculated by sampling component
⊗ Result of Newton iteration in new step
t

Figure 2. Solution sequence.

2.1.6 IDA Implementation

The chosen solution introduces an extra continuous component for each sampling group. This ‘black box’ component is described as an NMF component and emulates the sampling group, seen from the continuous system. It partakes in all activities pertinent for continuous components; it delivers residuals and Jacobians, the latter obtained by numeric differentiation. This makes a reliable and effective Newton iteration possible.

The component defines one equation for each output from the sampling group, equating this output to the corresponding controlled variable in the continuous system.

All sampling outputs from the group, required for calculation of residuals in these equations, are obtained concurrently by a call of a solver routine. The id of the sampling group is provided as a parameter together with the inputs to the sampling group. The solver uses these data to execute the group, rerunning the latest sampling step with sampling state memory fetched from backup.

The black box component is only active in the solving of the continuous system. The activation of the group when stepping through the global step is done by the solver without reference to the black box.

2.1.7 Short step regime

The discussion so far focuses on the continuous behavior of the sampling systems. An entirely different scenario appears, if the global time step happens to be shorter than the sampling steps. This may very well happen, when an abrupt change in the central system or in its inputs triggers fast transients.

Now, the activity of a sampling group can no longer primarily be regarded as smoothly incorporated in a continuous long-term progression. Rather, its discrete nature comes to the fore. When some short global steps have been taken, and the time to activate a sampling group is reached, whatever output it produces will have to be accepted. If they represent a discontinuous change, the solver will make an initial value calculation and then resume global integration. If they appear small, the global integration will have opportunity to increase step length and return to the normal long step regime.

2.2 Application examples

2.2.1 Basic performance experiments

The performance of the modified implementation is illustrated by some test results presented below. The simulated system is a single office zone with local heating and cooling. Tests were run for a three month summer period with observed climate data.
The cooling room unit was controlled by a PI-controller, implemented both as a continuous model and as a sampling discrete-time algorithm with a rate of 1000 activations per hour. The NMF equations for the continuous version were:

```
E := IF Mode < 0.5 THEN
       (SetPoint - Measure)
     ELSE
       (Measure - SetPoint)
     END_IF;

OutSignalTemp := k * (E + Integ);

OutSignal = IF OutSignalTemp > hilimit THEN
              hilimit
            ELSE_IF OutSignalTemp < lolimit THEN
              lolimit
            ELSE
              OutSignalTemp
            END_IF;

Integ' = E/ti + (OutSignal - OutSignalTemp)/tt;
```

where

SetPoint	Reference signal
Measure	Input signal
E	Control error
Integ	Integrator term
OutSignal	Control signal
OutSignalTemp	Control signal (temp)
k	Gain parameter
ti	Integration time in seconds
tt	Tracking time in seconds
mode	Control mode: 0= heating type control, 1= cooling type control
hilimit	High limit for OutSignal
lolimit	Low limit for OutSignal

In the discrete version, the differential equation for the Integ term was instead solved locally:

```
IntegPrim := E/ti + (OutSignal - OutSignalTemp)/tt ;
Integ := Integ + h*IntegPrim;
```

where

IntegPrim	Integrator derivative
h	Sampling interval

In the discrete version, the Integ variable is declared as an NMF assigned state, which means that it is memorized between evaluations.

The cases with the sampling controllers were run once with the global time step equal to that of the controller, and once with multi-rate integration. The outcome is summarized in Table 1 for the three cases:

A. Continuous controller (function block)
B. Sampling controller with multi-rate integration (new method)
C. Ditto, solve global system each sampling step (conventional discrete method)

Table 1. Performance tests, PI-controller

	A	B	C
Number of variables	2 187	2 193	2 193
Number of steps			
global successful	6 946	10 540	2 209 400
global total tried	28 035	42 546	2 216 402
sampling successful	0	2 208 000	2 208 000
sampling total tried	0	2 688 241	2 209 543
Integration time [s]	11	21	1 704

The results show that the sampled controller implementation is a factor of two less efficient than a continuous ditto, but the time reduction of the new method with respect to Case C still exceeds 98%.

Figures 3 and 4 show the zone air temperatures for Cases A and B during the last week of simulation. The cooling setpoint, 25C°, is not always met due to limited cooling power. A slight ripple, reflecting the selected tolerance, can be seen in the discrete case. Results for Case C are identical to Case A and are not shown.

Similar tests have also been done with on-off controllers, to investigate the performance for discrete state control signals. These results are equally satisfactory, actually showing even smaller penalties for the discrete-time implementation (case B vs. A).

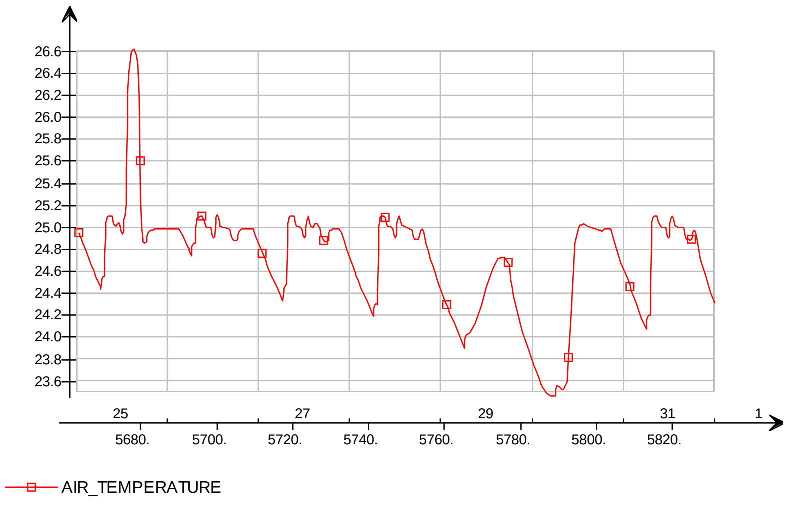

Figure 3. Air temperature for Case A

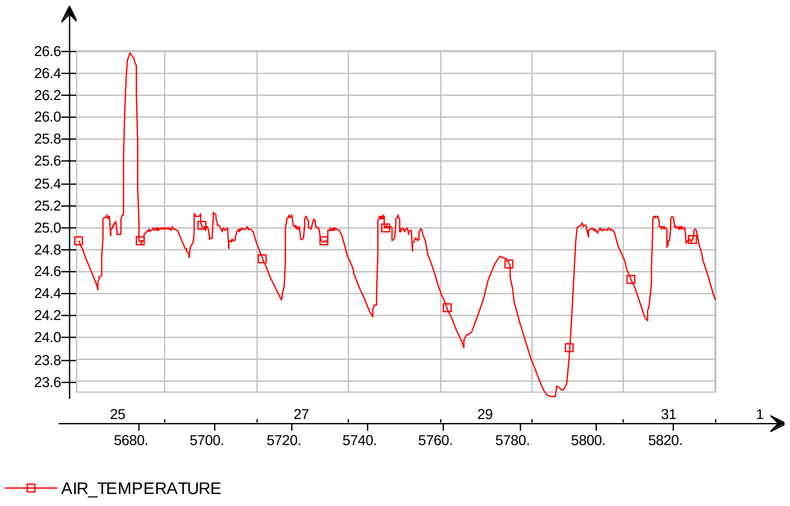

Figure 4. Air temperature for Case B

3 GENERATION OF CONTROLLER C CODE BY THE MATIEC COMPILER

3.1 Usage scenario

Two basic usage modes are envisioned for the combined Beremiz-IDA Simulation Environment system:

- Simulation (SIL) mode
- HIL mode

Figure 5 shows the intended usage scenario in Simulation mode. A control expert uses the Beremiz system to define basic function blocks, such as PID-controllers, filters etc. The pre-compiled function blocks are presented to the modelling engineer (the normal application end-user) on the modelling palette. A new model category of discrete-time components has been defined. The user may construct discrete-time subsystems (mostly controllers) by associating components with a clock. Discrete-time components are automatically arranged in “firing order” according to signal causality when IDA Solver is invoked. Simulation is carried out using the numerical method described in Section 2.

An advanced user may of course use the Beremiz system directly to build more complex specialized controllers in a similar manner and import these directly to the model schema.

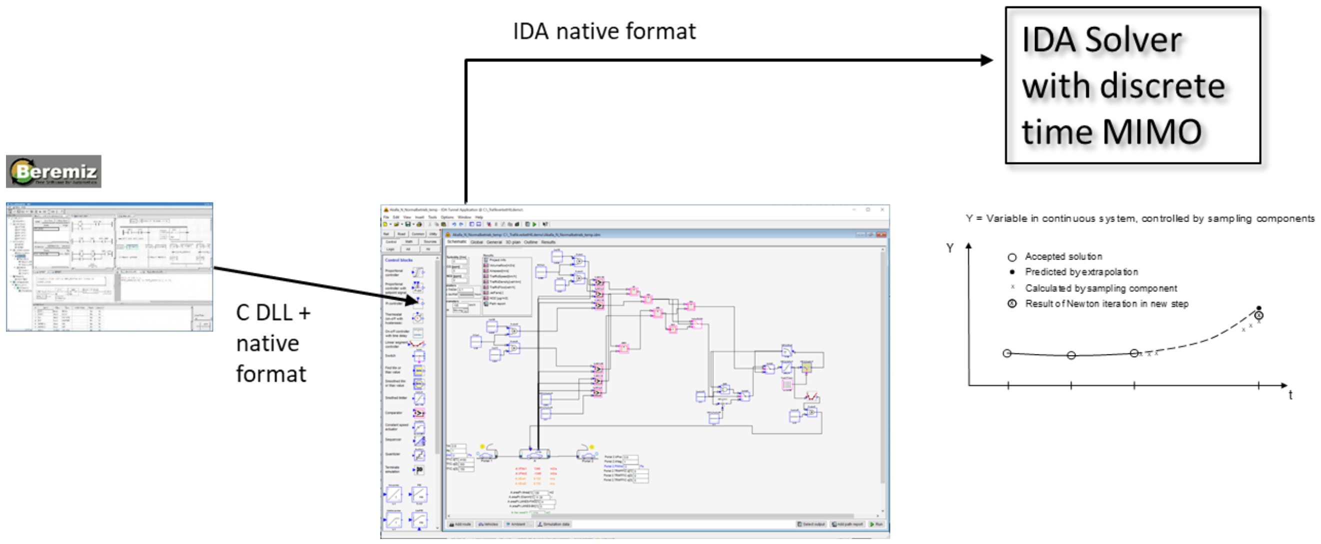
IDA native format
IDA Solver with discrete time MIMO
Beremiz
C DLL + native format
Y = Variable in continuous system, controlled by sampling components
Y
Accepted solution
Predicted by extrapolation
Calculated by sampling component
Result of Newton iteration in new step
t

Figure 5. Simulation mode.

In HIL mode, depicted in Figure 6, the discrete-time (IEC 61131-3 components) are automatically collected in the simulator and a PLCopen XML description of the resulting controllers are generated and opened in Beremiz (or other IEC 61131-3 Integrated Development Environment, IDE).

In the IDE, a control engineer completes the final controllers. Often many functions that are irrelevant for the simulation model will have to be added, e.g., alarms, safety systems, back-up control scenarios, manual intervention scenarios etc. Resulting code is loaded to a physical PLC.

In the simulator, in HIL mode, the discrete-time components have been replaced by import and export objects. Communication with the physical PLC is established based on signal identities from the simulation model (that have also been reflected in the PLCOpen XML file).

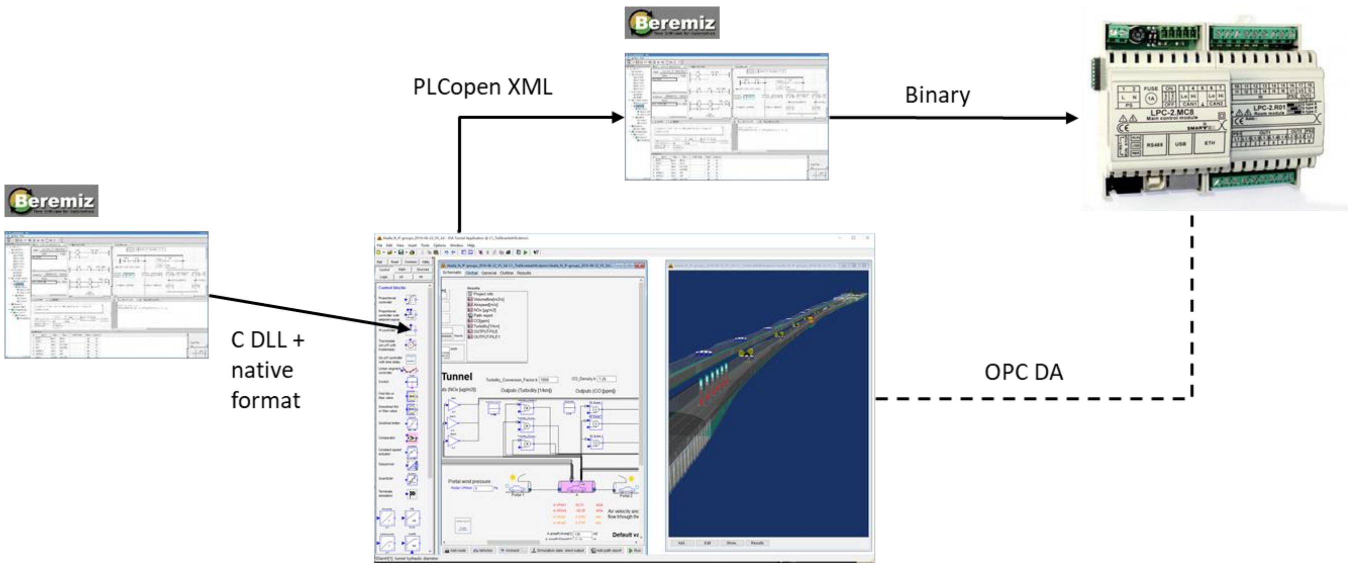

Figure 6. HIL mode.

3.2 Prototype implementation

The C code that is generated by the open source MATIEC compiler has been equipped with functions for saving and restoring state. This is not a standard operation for PLC-systems, but fortunately the structure of MATIEC enabled this change which is crucial for the numerical method as described in section 2.1.2. A wrapper function for using generated code from the MATIEC compiler with the IDA solver has been developed and tested on complex examples in the SIL mode. The wrapper is configurable and can be reused for different projects by adjusting the configuration to match the set of states of the controller.

The same generated code has also been compiled and transferred to a physical PLC which was then connected with the IDA simulation environment in HIL mode utilizing the OPC UA communication capabilities developed for this purpose.

The MATIEC compiler has also been used to generate code for a library of basic Function Blocks from the IEC 61131-3 standard. In the IDA simulation environment, corresponding GUI representations of the individual Function Blocks have been implemented. This collection of new discrete control components allows users of the IDA simulation environment to design and simulate control systems using standard Function Blocks. It is then possible to make a direct translation of the Function Block composition in the IDA simulation environment back to the IEC 61131-3 languages via a corresponding PLCOpen XML description that can be imported into the Beremiz programming system (or into any other IDE).

4 EVALUATION OF THE PROTOTYPE SYSTEM

The prototype system has been used to demonstrate the feasibility and illustrate the benefits of SIL testing for tunnel as well as building control systems. Testing the actual control code in a simulated environment allows thorough validation of the performance in extreme situations like when there are accidents and fires in a tunnel, and the prototype has allowed exploration of this scenario both in HIL and SIL mode.

Control system performance can be evaluated for purposes of optimization of energy consumption or what if analysis for different scenarios, and SIL testing allows the control designer to deliver solutions with efficient normal operation as well as safer systems in extreme situations thanks to the ability to evaluate many scenarios in short time. A valuable lesson learnt when using the prototype is that apart from models of the physical system and the control system, there is also a need for simulation models capable of replacing the human in the loop to fully exploit the potential of the SIL mode in some scenarios where the impact of non-deterministic human behavior is significant.

In addition, some potential improvements of the prototype system have been identified as a part of the evaluation. Implementation of automatic generation of wrapping functions for IDA Solver memory management of controllers' states and automatic generation of PLCOpen XML descriptions of assembled discrete function blocks could make the process more efficient and less error prone. Functionality similar to what is described in D2.6 could have been useful in this demonstrator. The setup of the HIL PLC connection could then have been simplified by automatic configuration of the connection to the OPC UA server and ultimately result in functionality for straightforward switching between SIL and HIL mode from within the IDA simulation environment.

5 CONCLUSIONS

A prototype simulation system for efficient simulation of physical systems controlled by discrete-time controllers that are described using the standard PLC programming set of languages: IEC 61131-3 has been developed. The prototype system makes it easy to use the same actual control code for simulation as for physical deployment enabling efficient optimization, analysis and validation. Realization of the prototype required a combination of open source development, specialized numerical methods and new functionality in the IDA simulation environment. The prototype system comprises the key components of a commercial solution a provides solid foundation for further development and commercialization of the technology, which is expected to simplify the design of efficient and well tested control systems for tunnels as well as buildings with EQUA tools.

REFERENCES

Dahlquist, G. 1983. On One-leg multistep methods, TRITA-NA-8301, Royal Institute of Technology, Stockholm, Sweden

Sahlin, Per, Pavel Grozman, and Equa Simulation AB. "IDA Simulation Environment-a tool for Modelica based end-user application deployment." Proceedings of the Third International Modelica Conference. 2003.