Methods for Reliable Simulation-Based PLC Code Verification

Henrik Carlsson, Bo Svensson, Fredrik Danielsson, and Bengt Lennartson, *Member, IEEE*

***Abstract*—Simulation-based programmable logic controller (PLC) code verification is a part of virtual commissioning, where the control code is verified against a virtual prototype of an application. With today's general OPC interface, it is easy to connect a PLC to a simulation tool for, e.g., verification purposes. However, there are some problems with this approach that can lead to an unreliable verification result. In this paper, four major problems with the OPC interface are described, and two possible solutions to the problems are presented: a general IEC 61131-3-based software solution, and a new OPC standard solution.**

***Index Terms*—Industrial control system, programmable logic controller (PLC), simulation, simulation-based PLC code verification, virtual commissioning.**

I. Introduction

The term "industrial control system" is a broad definition for programmable controllers used in industry to control machines and processes. An example of an industrial control system commonly used in industry today is the Programmable Logic Controller (PLC). An industrial PLC handles not only discrete events and supervisory control; it also handles analogue feedback, motion control, positioning control, and other time-critical functions. Therefore, in this paper, PLC is used as a general name for industrial control systems. The most characteristic feature of PLCs is the reprogrammable control function that is described by the control code. PLCs are usually defined as hard real-time systems, which implies that the PLCs have a guaranteed scan cycle time and that the control code must be executed within this time frame [1].

Traditionally, the development of PLC controlled applications, mechanical design and control code programming have been performed sequentially [2], [3] and partly online, where the control engineer has to wait with the programming, verification and optimization of the control code until the mechanical engineer is done with his or her work.

Manuscript received June 17, 2011; revised September 13, 2011; accepted November 16, 2011. Date of publication January 02, 2012; date of current version April 11, 2012. This work was supported in part by the European Commission Project FLEXA, under Grant 213734. Paper no. TII-11-287.

H. Carlsson, B. Svensson, and F. Danielsson are with the Flexible Industrial Automation Research Group, Department of Engineering Science, University West, SE-46186 Trollhättan, Sweden (e-mail: henrik.carlsson@hv.se; bo.svensson@hv.se; fredrik.danielsson@hv.se).

B. Lennartson is with the Flexible Industrial Automation Research Group, Department of Engineering Science, University West, SE-46186 Trollhättan, Sweden and also with the Department of Signals and Systems, Chalmers University of Technology, SE-41296 Göteborg, Sweden (e-mail: bengt.lennartson@chalmers.se).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TII.2011.2182653

A more attractive way is to do this in a concurrent way and totally offline, where both the mechanical and the control engineers work in parallel [3], [4]. A common name for this offline and concurrent process planning approach is virtual commissioning [3]–[7]. A broad definition of virtual commissioning might include such tasks as design (e.g., fixtures, robot tools and factory layout), programming (e.g., PLC, robots, CNC machines, servo cams), verification and optimization [8], [9]. However, the focus in this paper is on simulation-based PLC code verification where real PLCs are used together with simulation tools. Several examples exist of how industrial control systems and PLCs can be connected to simulation tools; two industrial *de facto* standards for this connection are RRS and OPC. RRS is a standardized interface between a simulation tool and a representation of a robotic control system, while OPC is a more general approach for communication with PLCs and other control equipment. RRS has a synchronization mechanism that makes sure that all robot movements are simulated [10].

The more general approach, OPC, includes a number of specifications that define interfaces between PLCs and regular computer applications. Today, state-of-the-art robot simulation and discrete event system simulation implement an OPC client that allows PLCs to control the simulation model. However, OPC suffers a major drawback; no mechanism exists [10] that guarantees that all computations performed in the PLC are considered in the simulation. This can lead to two types of problem: 1) real-world errors are not discovered in the simulated world and 2) errors are discovered that do not exist in the real world.

In this paper, four major issues that will trigger the above-mentioned OPC problems are identified and discussed. These issues will cause unreliable PLC code verification. To solve these issues two solutions to the problem are suggested: 1) an IEC 61131-3-based solution that could be implemented in a PLC at the design phase of the application and 2) a proposal for a new OPC interface that includes a mechanism to guarantee reliable PLC code verification. These solutions have been verified with a formal model, constructed in NuSMV [11] and proved to work. A case study is also presented to show the hazardous effect of unreliable PLC code verification.

II. Interfaces Between PLCS and Cape Tools

Tools that could be used for simulation-based PLC code verification usually sort under Computer Aided Production Engineering (CAPE) tools. CAPE is a general term for production-related simulation tools. There are two main subtypes of tools [12]–[15]:

1) Discrete event system simulation (production flow simulation), which can be used to analyze performance or product

flow in a cell, factory or enterprise. Months or even years of production can be simulated in a short while.

2) Robot simulation (geometric and kinematic simulation or Computer Aided Robotics), where robots and other moving devices can be simulated and programmed offline.

A common feature of all these CAPE tools is their ability to handle several types of production scenario on different levels, where a variety of robots, machines, manufacturing resources, control logic representations, etc., are integrated in a unified simulation. However, the representation of the control functions in CAPE tools is usually conducted on a general level and described using a simplified model of the main functional behavior of the real PLC [16]. An example of a simplified control function representation in a CAPE tool is a Sequence of Operations list. Consequently, with simplified PLC models, the real control functions, including motion controls, etc., are not executed.

A solution to the CAPE tools drawback of simplified PLC models is to use a hardware-in-the-loop simulation. Hardware-in-the-loop simulation, described in, e.g., [17] and [18], is a real-time simulation method in which real hardware, e.g., a PLC, is embedded in the simulation. Hardware-in-the-loop simulation as a means of testing control systems is not new; the aerospace industry has been using this technique ever since software first became a safety-critical aspect of flight control systems [19].

Freund *et al.* [20] have identified and described the integration problem between a real PLC and the remaining part of the simulation model. The main problems identified include a lack of time synchronization and a real-time data transfer mechanism. The time synchronization problem is due to the fact that the CAPE part of the simulation model runs in another time space (virtual time) compared to the introduced PLC, which runs in real-time. Ma *et al.* [15] identify the problem with real-time dependent control system functions, e.g., timers, when other parts of the simulation run in virtual time.

There are some different methods to include a PLC in a simulation. In [21], the simulation uses interfaces to a fieldbus for communicating with the PLC. A more general approach is to use the OPC interface that is presented in detail in this paper.

A. Realistic Robot Simulation (RRS)

Realistic Robot Simulation (RRS) is a standard interface between robot simulation tools and robot control systems. By using RRS, it is possible to use offline created control programs in the real robot without any correction [22].

The main driving force for the RRS project was the error introduced in simulated robot paths due to a lack of realistic simulation of the controller behavior.

The idea of RRS is to integrate the part of the robot motion control software that is responsible for the motion behavior into the simulation system. Hence, the simulation is controlled by the same motion control strategies as the real robot will be [23]. In 1998, the RRS-2 project was started; the aim was to cover full functionality of robot controllers.

Further, the RRS consortium has presented a solution for simulating the control function of general PLCs [10], [23], [24]. This approach is based on virtual controllers, developed by the PLC manufacturer, connected to the simulation tool. However, to the authors' knowledge there is no product available today that uses this approach.

B. Fieldbus Emulation

Fieldbus emulation is a technique to include the target fieldbus in the simulation. The main purpose of fieldbus emulation is to test the real PLC code with correct addresses and the complexity of the real bus. This will allow verification of hardware configurations and signal allocations. It will also enable the developer to use the real I/O dependent variable names and symbols. A real PLC or emulated one is connected to the virtual fieldbus and the bus nodes are modelled inside the CAPE tool. Fieldbus emulation can be found in, e.g., WinMOD. Most of the existing fieldbus emulators do not address unreliable verification introduced by the simulation. However, SIMBA Pro and SIMIT address this by adding additional hardware. They offer the possibility to implement small models on board dedicated hardware. For larger models they offer an OPC connection to more standard CAPE tools. Further, SIMBA Pro and SIMIT are Siemens S7 specific solutions and not generic.

For relatively slow processes and small systems this might be an efficient way. With extensive simulation models real-time indeed becomes an issue. It is possible to combine the proposed synchronization method presented in this article with fieldbus emulation techniques to ensure correct behavior in all circumstances. A fieldbus emulator for Profibus and CAN has been implemented to test this. However, this is not the focus of this paper and therefore only mentioned as this.

C. OPC

OPC was founded by a few companies in the mid 90s in order to ease the exchange of process data [25]. OPC has been, and still is, developed by the OPC Foundation[26]. OPC is server-client based, where the server is vendor-specific and the client general.

OPC consists of several specifications [25]–[27]. These specifications contain information about how the server and the client should exchange data. OPC DA (Data Access) was the first specification [28] that made it possible for any OPC DA client to access data from an OPC DA server that fulfils the same specification. OPC DA is used for moving real-time data from devices to Microsoft Windows applications, where real-time here means current data in the device, not historical. Nothing is said about how the data is transferred from the device to the application, and real-time is not defined either.

OPC DA is based on COM/DCOM, which is a Microsoft technology. This choice leads to platform dependency, and there is, for instance, no standard solution for using OPC on Linux, even though such solutions exist. Another issue with OPC is the problem with firewalls when the OPC client and server are located on different machines. This is due to DCOM, a problem that can, however, be overcome by a tunneller [29].

A first attempt to select another platform than COM/DCOM was presented within the OPC XML-DA specification, where COM/DCOM was replaced by web services. However, the performance was very poor compared to OPC DA [30].To overcome this weakness, a new specification was introduced; this specification is called OPC Unified Architecture (OPC

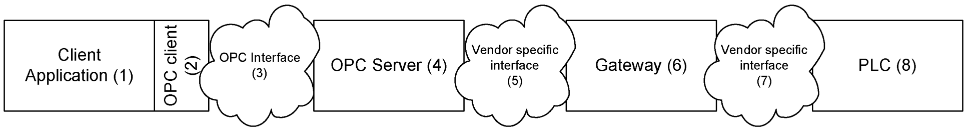

Fig. 1. OPC communication model.

UA) [31]. OPC UA uses two different transport protocols; SOAP over HTTP and TCP [32]. Compared to OPC XML-DA, OPC UA also supports binary encoding of the data instead of the XML encoding that produces a large amount of overhead data. The performance of OPC UA compared to OPC DA is somewhat slower, between 1.1 and 1.6 times when reading values [32].

According to Matrikon [33], the plan is not to replace OPC DA with OPC UA, instead both specifications should coexist and complement each other. OPC DA is still the most common specification, 99% of all OPC products today are implementations based on OPC DA [32]. Since this paper deals with CAPE tools, and to the authors' knowledge there is no commercial CAPE tool available today that supports OPC UA, only OPC DA is considered. OPC UA is, however, very interesting. PLCOpen [34], a worldwide organization that works for resolving topics related to control programming, has chosen OPC UA as its technology for data exchange. An alternative to OPC DA is the OMG [35] specification Data Acquisition from Industrial Systems (DAIS) [36]. DAIS is based on real-time CORBA [37]. However, this specification is not used in the type of application covered in this paper. CAPE tools and PLCs of today generally utilize OPC for intercommunication.

D. OPC Together With CAPE Tools

The possibility of including real PLCs in CAPE tools has been around since the mid 90s, utilizing vendor-specific protocols. Some common purposes are offline programming, verification, and optimization. Since the beginning of the last decade, many CAPE tools have an OPC client implemented, usually an OPC DA client, which makes it possible to connect to any OPC DA server [38]. The I/Os in the PLC control code are made available to the OPC server. CAPE tools usually have some kind of signal representation similar to real machines, e.g., a start signal for a robot. These signals are then mapped to the I/Os from the PLC via OPC. The simulated machine or process can thereby be controlled in the same way as it would be in reality. Examples of CAPE software with OPC functionality include: Delmia Automation, Visual Components, Process Simulate, and Arena.

E. OPC Communication

To ease further discussion of OPC, a model of OPC communication is introduced, see Fig. 1.

Fig. 1 shows a general model for how an application (1), with an integrated OPC client (2), can connect to a PLC (8). There are three different communication parts in this model: the OPC interface (3), i.e., COM/DCOM, the communication interface (5) between the OPC server (4) and the gateway (6), and the gateway interface (7) to the PLC.

There might be other models, where, e.g., the OPC server has an integrated gateway, but the principle is still the same. Another scenario is an integrated OPC server in the PLC, where (6) and (7) might be unnecessary. In general, the model is valid for a standard OPC connection.

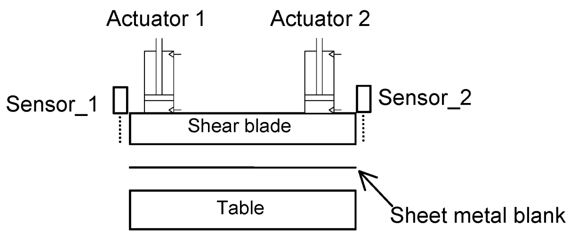

Fig. 2. Explanatory sketch of example application.

III. Identified Problems With Current Interfaces

With today's general OPC interface it is easy to connect a PLC to a simulation tool for, e.g., verification purposes. However, when verifying PLC control code with this method, one might encounter several unexpected problems due to free-wheeling. Free-wheeling is defined in this paper as asynchronous execution of the PLC and simulation tool. These problems will indeed affect the result and lead to unreliable verification. The problems identified in this paper can be divided into four main categories, namely: time delay, jitter, race condition, and slow sampling. These will be described in detail in the following subsections. OPC UA, which was mentioned earlier, does not introduce any mechanism that would solve this problem, so the rest of this paper will only deal with OPC DA, since it is the most commonly used specification today.

A PLC controlled sheet metal shear will be used as an example to explain the different problem categories, see Fig. 2. A typical shearing line is used to produce sheet metal blanks of the desired length. In this example two actuators, 1 and 2, are used to drive the shear blade up and down. Two sensors, Sensor_1 and Sensor_2, are used to level the shear blade at the upper position. Each actuator is equipped with a position feedback sensor to be able to control the lower position limit. A kinematic simulation model of the described process was implemented and then connected to a real PLC via OPC to demonstrate the phenomena. The PLC has a cycle time of TPLC. A simplified PLC code example of the low level control is shown in Fig. 3. It will represent the PLC (8) in the model described in Fig. 1.

To be able to perform an accurate and reliable simulation, all values from the simulation should be considered in the PLC and *vice versa*. The tools (hardware and software) that have been used in this paper are as follows.

- CoDeSys Soft PLC [39].
- BinarBifas 60 PLC [40].
- CoDeSys OPC server, (supports OPC DA 2.0) [39].
- Process Simulate.

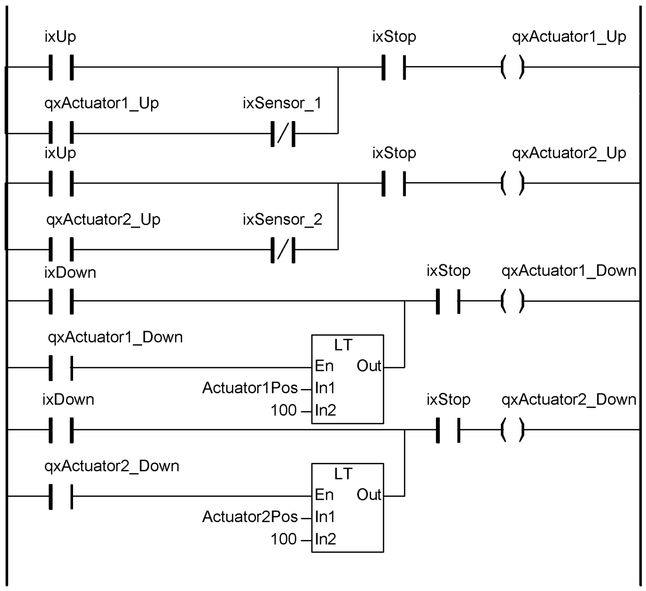

Fig. 3. Part of the PLC program to control the sheet metal shear implemented in IEC 61131-3 Ladder Diagram (LD), the declaration part is not included. ixDown and ixUp are signals to run the shear blade up and down.

- Integration Objects' OPC Data Access client SDK (supports OPC DA 2.05) [41].
- A sheet metal shear model.

A. Free-Wheeling

To run a PLC and simulation asynchronously, i.e., the PLC is running at its own pace with respect to the simulation pace, is defined in this paper as free-wheeling. Free-wheeling appears to be the most commonly used scenario today when connecting real PLCs to CAPE simulations. Free-wheeling will introduce four major issues namely jitter, race condition, slow sampling and time delay. Simulations using the OPC or similar techniques that do not consider these four problems will in this paper be referred to as unreliable simulations.

The classification presented in this paper; time delay, jitter, race condition and slow sampling, are not covered in the literature in relation to OPC communication and CAPE tools. All four issues are usually presented in the literature as one common, unspecified issue or problem. In this paper, it is referred to as free-wheeling.

B. Common Solution to the Free-Wheeling Problems

The most common solution found in the literature to the problems introduced by free-wheeling is synchronization through halt, based on the assumption that the CAPE tool has the possibility to run faster than the PLC [15], [39], [40]. According to this assumption, it should be possible to halt/slow down, when necessary, the faster CAPE tool in order to maintain a synchronized system. However, these time synchronization methods are shown in this paper to not work. To show this effect, such a synchronization method was applied in the sheet metal shear example. A high-speed timer was used in the simulation to achieve high accuracy and to match the PLC cycle time. This timer was used to halt the simulation in order to achieve a synchronized

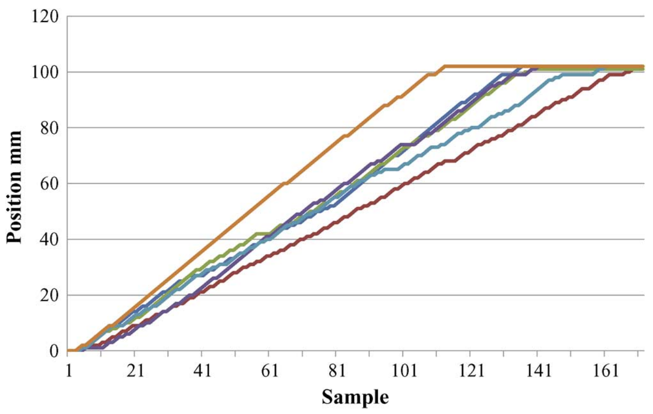

Fig. 4. Measured actuator position during the simulation of the sheet metal shear. Each line represents a run, and each run was carried out at the exact same conditions.

system. The result from the experiment, shown in Fig. 4, reveals the behavior of a nonreliable verification. If the suggested methods in the literature had worked as expected, Fig. 4 should have shown one single straight line.

C. Jitter and Time Delay

The cycle time of the CAPE tool, (1) in Fig. 1, T_{CAPE}, is the time needed for the simulated actuators to execute new values. Theoretically, if T_{CAPE} is not equal to T_{PLC}, nondeterministic behavior might arise. In practice, due to non-real-time behavior of common operating systems, T_{CAPE} is indeed not equal to T_{PLC}. In essence, the problem is that a regular PC, its operating system and the CAPE tools cannot be considered to be a real-time system [42], and a non-real-time system is not designed to respond to time-dependent signals in a deterministic way. The common suggestion, found in the literature, is to speed up the CAPE simulation. However, T_{CAPE} must still be exactly equal to T_{PLC} to guarantee deterministic behavior. No working mechanism exists in the OPC specification that can solve this problem.

Even if it were possible to run CAPE tools at the same pace as the real PLC, there would still be time uncertainties due to the microprocessors in regular computers and the operating system. This phenomenon is referred to as *jitter* [43].

Jitter can be defined as randomly varying time delays [44]. If a constant time offset exists, it can be referred to as *time delay*. Hence, the total time uncertainty is *jitter+time delay*.

The complicated communication paths in the OPC specification, see Fig. 1, will introduce jitter and time delay. This will indeed be intensified if a communication component with low bandwidth is a part of the OPC chain. The jitter and time delay will cause unreliable verification results.

For instance, the OPC interface (3), based on COM and DCOM, can introduce problems because of its lack of real-time support [45]. In [45], a possible approach to providing real-time support for COM is presented, provided that the underlying operating system supports real-time operations. Since OPC uses COM for the communication between the server and the client, it is possible to apply this approach. However, there are to the authors' knowledge no examples of industrial CAPE tools that support real-time operations, and thus would it still be

```
a:=R_TRIG(CLK:=Sensor_1);
b:=R_TRIG(CLK:=Sensor_2);
c:=R_TRIG(a AND b);
```

Fig. 5. Sheet metal shear alignment code example in IEC 61131-3.

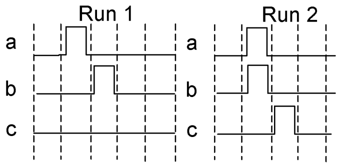

Fig. 6. Two identical simulations of the calibration code in Fig. 5. In Run 1, a race condition phenomenon occurs, and the OK signal c is not triggered as in Run 2.

problem with lack of real-time support in the communication model presented in Fig. 1.

To demonstrate the result of jitter, randomly varying time delays, the sheet metal shear example was setup to run with a PLC scan cycle T_{PLC} of 10 ms and a simulation scan cycle T_{CAPE} of 10 ms. The OPC update rate was set to 1 ms for the OPC client group used, see OPC group object in the OPC DA specification [28] for more details.

The position of Actuator 1 was measured and plotted in Fig. 4 for six different runs. According to the deterministic behavior of both the PLC and the kinematic model, the expected outcome is six identical runs. However, due to the jitter problem introduced by the OPC interface or due to the possible skew PC clock, the outcome is unreliable and stochastic. The test was carried out both on a soft PLC and a real PLC with the same result.

The target equipment, e.g., sensors and actuators, might introduce jitter dependent on mechanical or electrical properties. However, the jitter found in the real equipment does not correspond to the one introduced by the OPC interface. The intention with this work is to eliminate phenomena introduced by the simulation tools such as jitter. If it is important to include the jitter from the real equipment in the simulation it should be modelled in a standard way, e.g., as a normal distribution $N(\mu, \sigma)$.

D. Race Condition

A race condition arises when the result is dependent on the sequence or timing of several events. A race condition might occur when two or more OPC dependent variables change at the same time (i.e. in the same cycle), but the result is transferred to the receiver at different times. The problem can be explained by the following sheet metal shear example, where the code section in Fig. 5 is used for detecting miss alignment of the shear blade.

The result is simulated behavior, see Fig. 6, that differs from the real one. This error will not occur in the real application, but it will cause unnecessary troubleshooting.

There is no mechanism in the OPC specification that can guarantee correct behavior in this situation. This behavior is actually described in the OPC DA specification, and the solution to the problem is additional handshaking and flag passing between the client and server [28]. This behavior can also be referred to as inexact synchronization [46].

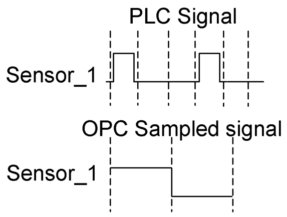

Fig. 7. Example of slow sampling.

E. Slow Sampling

An OPC client specifies the fastest rate at which data changes may be updated from the server, i.e. the sampling rate. However, this sampling rate is not necessarily the same rate at which data will be transferred between the PLC and the OPC server.

Indeed, the sampling rate is crucial for a deterministic and reliable simulation. Changes that are faster than the update rate will not be recognized by the client, see Fig. 7.

In the sheet metal shear example the upper sensor signals Sensor_1 and Sensor_2 are critical input signals from the process. However, with an OPC update rate that does not match the PLC or the simulation, slow sampling problems may easily occur.

F. New Standard

The purpose of OPC has never been to supply CAPE tools with data in real-time; this functionality has been adopted by the CAPE tool vendors. It would, however, be possible to use OPC as a more extensive tool for PLC code verification if the PLC vendors, CAPE vendors and OPC Foundation could agree on an extension to OPC, where some kind of synchronization is built into the specification. The purpose of the synchronization is to achieve reliable simulation results. A proposal for this extension is presented at the end of this paper.

IV. SDSP Simulation Architecture

Due to the lack of suitable, reliable and deterministic methods for the verification of PLCs together with CAPE tools, an architecture for distributed simulation and time synchronization—SDSP—was formulated in [48]. Distributed simulation with SDSP is not discussed further in this paper, for more information and examples see, e.g., [47] and [48].

SDSP is responsible for the following.

- Communication with all simulation clients.
- A common synchronized time.
- Common simulation data, e.g., I/Os.
- Handling of distributed simulation.

SDSP is a server-client-based concept with the server being responsible for managing the simulation. Whilst SDSP is mainly based on TCP/IP, other methods of communication are possible, such as, for instance, DDE, OPC or shared memory. One of the most important tasks for the server is to manage each subset of simulations to form an overall time-uniform simulation. The way in which this can be accomplished is set out below.

In order to use a CAPE tool in this concept, the tool must have an application programming interface (API) that makes it possible to include an SDSP client within the tool. This client can then handle the communication between the CAPE tool and

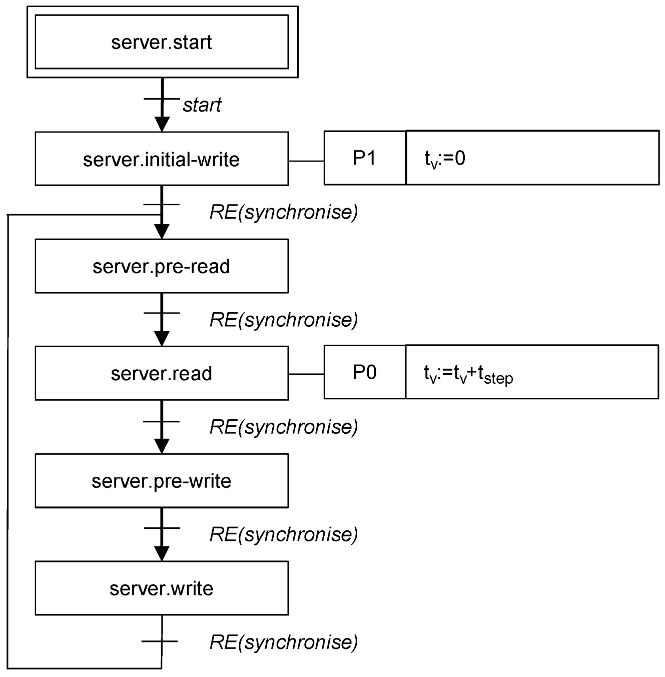

Fig. 8. States and associated actions for the simulation as represented in the SDSP server, described in SFC. The qualifier P1 means that the action is executed once at the entrance of the step, and P0 once at the exit of the step. RE means that the *synchronize* event is triggered on a rising edge.

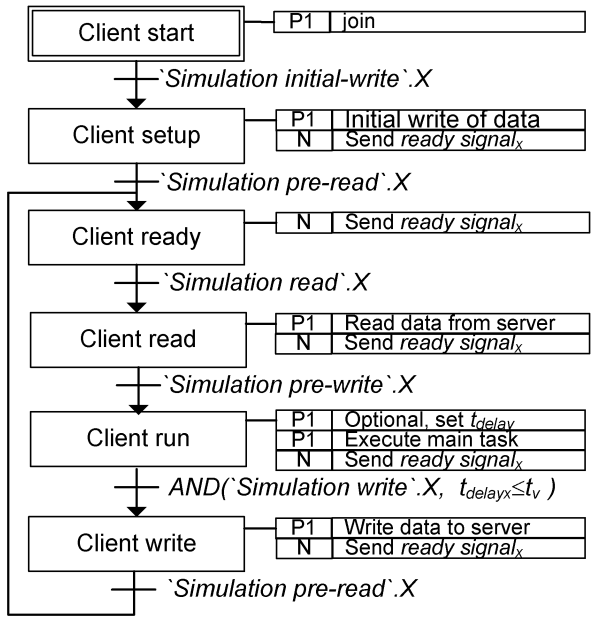

Fig. 9. States and associated actions for each client (subsimulation) as represented in the SDSP server, described in SFC. The qualifier P1 means that the action is executed once at the entrance of the step. N means that the step is executed as long as the step is active, but after P1.

the SDSP server. This is a common feature for many CAPE tools, e.g., ROSE for Robcad and Tecnomatix. NET SDK for Process Simulate.

In essence, the server contains common data (e.g., I/O values, servo values, and simulation parameters) and common logic. The server also contains an overall simulation or virtual time t_v and a state set {*server.start*, *server.initial-write*, *server.pre-write*, *server.write*, *server.pre-read*, *server.read*}, see Fig. 8. Initially, the server sets up the necessary structure in the database and sets the overall simulation state to *server.start*.

As a minimum requirement to form a simulation, the server needs information about the time step t_{step}, a formal start condition *start*, and the number of clients. The start condition tells the server when the entire simulation can commence, and generally the start condition defines when all clients required to run the simulation are ready to start. The step t_{step} is the smallest time step that the overall simulation can handle. However, a *delay* mechanism allows a client to run with a different time step, larger than t_{step}, than in the overall simulation without sacrificing time synchronization. For example, the delayed mechanism can be used in discrete event simulations to delay the synchronization for a specific client x with the time t_{delay_x}, where t_{delay_x} is the time for the next event.

A client is considered to be ready to start when it has been connected to the server and has joined the simulation. When the start condition

$$\mathit{start} = \mathit{AND}(\mathit{start\ logic}, \mathit{client}_1.\mathit{start}, \ldots, \mathit{client}_n.\mathit{start})$$

is fulfilled, the initial write state, *server.initial-write*, is entered. n is the number of clients in the simulation; the *start logic* argument can be used to introduce additional start conditions for the start of the simulation.

This mechanism provides the deterministic start behavior for the overall simulation. In the initial write state, each client has reached their *client.setup* state where they are supposed to initialize their own data (e.g., all inputs and outputs are set to 0), while the overall simulation time, the virtual time t_v, is also set to 0. To enter the following states, the following *synchronize* condition must be fulfilled:

$$\mathit{synchronize} = \mathit{AND}(\mathit{client}_1.\mathit{sync}, \ldots, \mathit{client}_n.\mathit{sync}).$$

The $\mathit{client}_x.\mathit{sync}$ signal indicates when a single simulation client is running (*FALSE*) or the point when it is ready and has executed the current time step (*TRUE*)

$$\mathit{client}_x.\mathit{sync} = \mathit{OR}\left(\mathit{ready\ signal}_x, t_{delay_x} > t_v\right).$$

After the initial write state, the overall simulation enters the pre-read and then the read state, see Fig. 8. Each client reaches its *client.read* state synchronously, see Fig. 9. In this state, each client is supposed to read data from the server. When all of the clients have executed their client read task and thereafter are synchronized by the server, the overall simulation enters the pre-write state, *server.pre-write*.

At the same time all clients enter the *client.run* state and are supposed to execute their main tasks, e.g., a program cycle for a PLC. The virtual time, t_v, is updated to the next time step at the *read* state, *server.read*, meaning that $t_v = t_v + t_{step}$. This tight synchronization procedure is necessary for a reliable simulation, and to prevent clients from acting in the wrong time space, i.e., one time step before or after the desired time.

To verify that this concept works, a formal model of the server-client concept has been formulated in NuSMV [11], see the Appendix. This model allowed it to be verified, among other results, that the clients follow each other synchronously.

V. Time synchronization With PLCS

The SDSP simulation architecture presented above is designed to handle verifications of time-dependent control code in PLCs connected to CAPE tools, i.e., to deal with the four problems, *time delay*, *jitter*, *race condition, and slow sampling*, described previously.

The internal clock in a PLC controls the execution to guarantee the cycle time and to achieve deterministic behavior. Due to this fundamental behavior in the real PLC, it has only been possible to incorporate an emulated or simulated PLC in the SDSP simulation architecture.

Emulation of a PLC is a technique used to obtain an exact representation of a real PLC [49]. In [47], an emulator of a PLC was used and it was extended to include a number of additional SDSP functionalities, thus making it possible to use it in conjunction with the time synchronization mechanism.

Even though an emulator is a good representation of a PLC, the real PLC is nevertheless preferable in many situations. This is because of a lack of emulators, coupled with the fact that creating an emulator can be extremely time-consuming.

However, even if it is possible to connect PLCs to the simulation architecture over OPC, it is not possible to directly time synchronize them with the mechanism within the architecture. To overcome this hurdle, a general method for the time synchronization of an IEC61131-3-based PLC is described in this section.

In order to utilize the new time synchronization method, the following requirements must be fulfilled.

- The PLC must support the IEC 61131-3 programming language standard.
- The PLC must be able to communicate with a regular computer, e.g., by OPC.
- The entire simulation must utilize an architecture that offers a time synchronization mechanism.

IEC 61131 is an international standard for programmable controllers, and is divided into several parts. IEC 61131-3 [50] describes a PLC software structure, languages and program execution [51].

In order to use the method presented in this paper, three modifications to the original control code to be verified need to be made: (1) A scheduler Program Organization Unit (POU), which is used as a complement to the original scheduler, must be added to the control code. (2) All program POUs within the configuration need a specific execution control function. (3) All time dependent function blocks and functions must be converted to deal with virtual time. The sections marked in Fig. 10 represent these modifications and will be described.

A. The POU Scheduler (1)

In order to implement a complementary scheduler, all POUs within the configuration must be set to the same priority, e.g., 1, and organised in the same task. The complementary scheduler POU is set to a higher priority, e.g., 0. This is to guarantee that the scheduler is executed first in every program cycle. The scheduler communicates with the SDSP architecture through two synchronization variables via, e.g., OPC.

When it is time to execute the control code in the PLC, the scheduler receives a signal from the server and the scheduler decides which of the other POUs should be executed by sending specific *execution signals*, see Fig. 10.

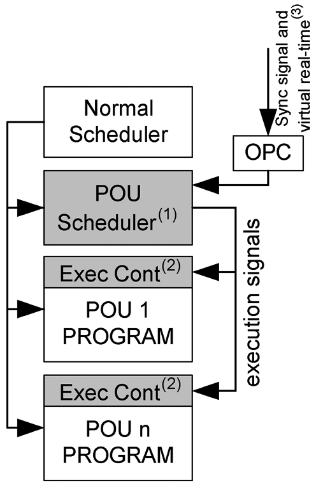

Fig. 10. Description of the supervisor POU and the connection to the normal POUs.

```
    IF NOT Simulated OR execution(task_n)
       (*Regular program begins here*)
        ...
       (*Regular program ends here*)
  END
```

Fig. 11. Pseudocode describing the execution controller.

There is no standard or general way to halt the real-time clock on a real PLC, as demanded by the simulation architecture.

Consequently, the scheduler also handles the virtual time, t_v, by reading the current value from the simulation server. This time is then stored as a global resource. The resource is then used as a replacement for the real-time clock.

B. Execution Controller (2)

To be able to control the execution order and timing of all POUs within a configuration, certain additional functionality must be added to each scheduled POU. This is accomplished by means of a specific "header and footer." This execution control can be implemented for all IEC 61131-3 languages. The execution controller, described in pseudo code in Fig. 11 determines whether or not the desired program should run.

However, the graphical language SFC has a more complex execution order that requires another type of execution controller [52]. An SFC consists of a series of steps and transitions, where each step can be associated with either one or a series of actions [51], the behavior of the action being determined by the action qualifier. For example, the qualifier "N" means that the action will execute while the step is active and the qualifier "L" will execute for a limited time, defined by "T."

According to the IEC 61131-3 standard [50], each action is associated with an instance of an *Action Control function block*. This function block controls the activation and deactivation of the action, depending on the action qualifier used. Due to the fact that some of the action qualifiers are time dependent, the PLC programming environment must offer access to modify the

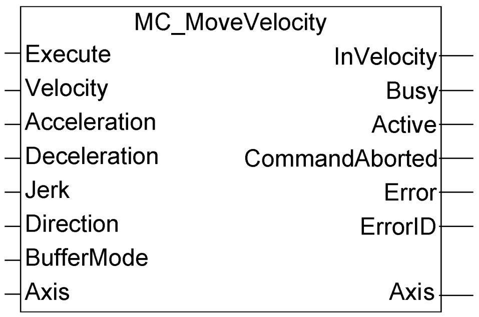

Fig. 12. Example of a PLC Open motion control function block.

action control function block in order to be able to use the proposed method within an SFC POU. The internal time dependent functions used in the function block must be updated so they can handle virtual time and an extra execution signal must be added that enables the SFC POU to execute at correct time.

In SFC, the transitions between each step can also be time dependent, i.e., a particular step can be active for a specific amount of time. This can be solved if it is possible to control these transitions. Should this not be possible, however, the same behavior can be obtained within the actions.

C. Time Dependent Functions (3)

IEC 61131-3 contains four standard timer function blocks, Timer On Delay (TON), Timer Off Delay (TOF), Timer Pulse (TP) and Real-Time Clock (RTC) [50]. However, these function blocks cannot be used directly, since they are usually based on the hardware clock that continues to run even though the clock should be halted according to the server. Therefore, special replacement timers are used. The replacement timers behave in the same manner as the regular ones, the only difference being that they use the virtual time received from the server. The virtual time is updated for all clients in the simulation at the same time.

There may also be other (not IEC 61131-3) vendor-specific functions, such as motion control blocks, that depend on the hardware clock. Thus, in order to be able to use the proposed time synchronization method these functions must also be upgraded to handle the virtual clock.

For motion control [53], [54], PLCOpen [34] offers a motion control library specification based on IEC 61131-3. The PLCOpen Technical Committee 2—Task force motion control—has defined a suite of specifications that defines function blocks for motion control: see Fig. 12 for an example of a function block that can be used for controlling an axis with a fixed velocity. Later specifications also include coordinated multi axes motion in 3D space. The first basic specification, which was released in 2005, has been implemented in over 30 products [34].

If the application that is to be programmed offline and simulated uses the PLC Open motion control function, it is possible to use a modified function block that uses the virtual clock.

In the proposed third edition of the IEC 61131-3 standard, there are some new features proposed that might ease the implementation of these synchronization functionalities. One proposed feature is that it will be possible to call a POU program from another program.

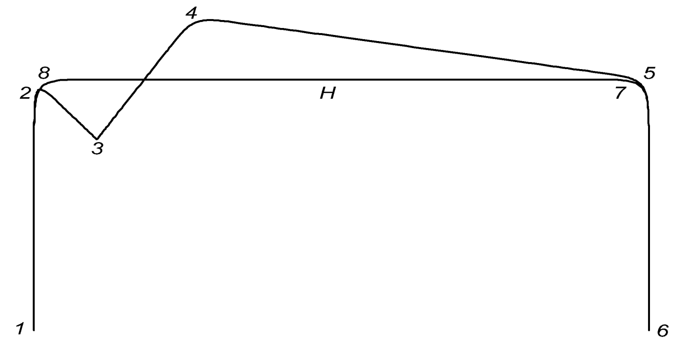

Fig. 13. The pre-programmed robot path used in Scenarios 1 and 2. The numbers represent the run order of the path and represent a 2D location (x, z).

VI. Case Study

A. Case Study Setup

To demonstrate the proposed time synchronization method and the disadvantages of unreliable simulation, a case study was set up. A real PLC from Binar [40] was programmed to control a two-dimensional servo controlled robot. The robot was programmed, in IEC 61131-3, to follow the path described in Fig. 13.

A simulation model was created in a CAPE tool, Process Simulate. The model represents the 3D geometry and kinematics of the robot. Two scenarios were carried out:

In Scenario 1, the CAPE tool was connected to the PLC in an unsynchronized way (free-wheeling) provided by CAPE tools today. In Scenario 2, the model was connected according to the synchronization method proposed in this paper.

To be able to use Process Simulate with the proposed method an adaption of the software was made. This application reads and writes values to the SDSP server and overrides the internal simulation engine in Process Simulate in order to use the values from the SDSP server. To be able to utilize the new time synchronization method, Scenario 2, the following preparations were carried out in the PLCcode:

- All tasks in the PLC configuration were set to the same priority and the same cycle time.
- A scheduler was implemented and added to the configuration.
- All timers in the configuration were replaced with modified timers including virtual time.
- An execution controller was added to each program.
- The servo control function blocks were replaced with new ones implying time synchronization.

B. Results

Two test scenarios were setup: 1) with no time synchronization and 2) with the time synchronization method proposed here. Several cycles were executed in both scenarios. Fig. 14 shows the results from the runs when no time synchronization was used, (1). The enlarged part shows the nondeterministic behavior when no time synchronization was employed (compared to the preprogrammed desired behavior in Fig. 13).

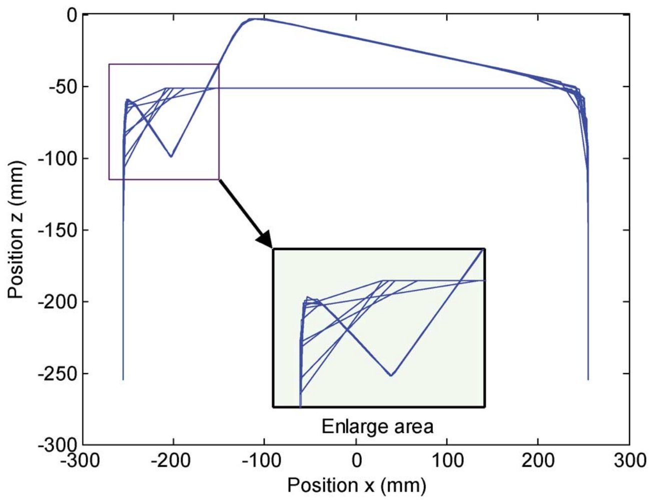

Fig. 14. Measured robot path without synchronization.

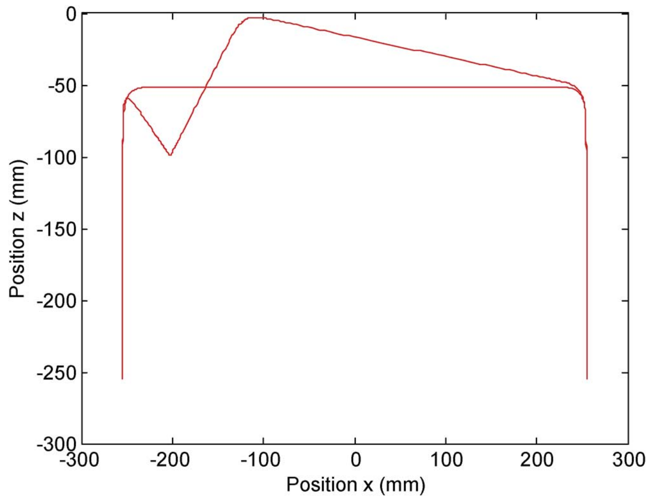

Fig. 15. Measured robot path with the new proposed time synchronization method.

This nondeterministic unreliable behavior is due to the four problems described earlier.

Fig. 15 shows the results when the proposed time synchronization method was used (2). The plot clearly shows that the nondeterministic unreliable behavior in Fig. 14 has disappeared. The results show that the suggested reliable time synchronization is necessary when verifying control code with CAPE tools. The case study was not intentionally designed to show unreliable behavior, it is an existing industrial application. All four issues discussed in the previous sections were found in this case study. Thus, these issues should not be neglected or considered to be marginal.

VII. Proposal for an Extension of OPC

As shown in this paper, the proposed synchronization mechanism based on IEC 61131-3 will fulfil the requirements to perform reliable PLC code verification. However, a more industrially attractive solution would be to embed a synchronization mechanism into the PLC and simulation tool. To achieve industrial acceptance, a solution based on an already accepted standard, such as OPC, is preferable.

The OPC Foundation, PLC vendors and CAPE tool vendors could together agree on an extension to the OPC standard and a simulation mode in the PLCs that would make it possible to fulfill the requirements for reliable simulation. The authors have identified the following requirements for a new standard to be of importance.

- Compatibility with existing simulation tools and PLC solutions.
- A synchronization mechanism to guarantee a reliable simulation and deterministic results as described in Sections IV and V.
- A synchronized common start, stop and reset functionality for simulation and PLC.
- Vendor independency.

VIII. Conclusion

OPC is today established as a *de facto* standard for connecting PLCs to CAPE tools for verification purposes, but in this paper four major issues of concern regarding its usage are presented and described in detail namely; jitter, time delay, race condition, and slow sampling. Each of these four issues will indeed result in unreliable results and hazardous effects as shown in the case study, e.g., false collision detection, wrong sensor signals, etc. To overcome these issues, two different approaches have been presented in this paper.

1) A new time synchronization method based on IEC 61131-3 together with simulation architecture. Such a method and architecture can be used for reliable verification and the development of PLC code with CAPE tools. The proposed synchronization method, together with the architecture, has been demonstrated to work on PLCs that are compatible with the IEC 61131-3 standard. For industrial usefulness, the synchronization part can be automatically generated and attached for general IEC 61131-3 languages before the downloading of the control logic to the PLC.
2) An idea to an extension of the already existing OPC standard is also formulated.

Both approaches have been tested on real PLCs as well as soft PLCs with successful results. The concept has also been verified by a formal model, implemented in NuSMV [11]. The four issues mentioned with free-wheeling were solved with both these synchronization methods. In a long-term perspective a standardized OPC-based solution (2) is the most attractive one. A further advantage of the presented approach is that it does not conflict with real-time capabilities of the target application. Thus, it is possible to combine the proposed time synchronization method with existing PLC scan cycle time watchdogs (a check if specified real-time limits are exceeded).

From an industrial point-of-view, a time synchronization method is necessary when verifying PLC control code. However, industrial state-of-the-art verification methods based on CAPE lack this type of feature. To be able to take the next step in the field of PLC code programming, verification and optimization with the aid of simulation tools, a general vendor independent standard that deals with the issues identified in this paper is needed.

APPENDIX

The following section shows the NuSMV [11] model of the server-client concept described in Section IV. The formal verification based on this model shows, among other things, that the clients follow each other synchronously when the time delays are equal for each client, indicating that the concept works.

```
MODULE client (serverstate, previousserverstate, tv) VAR
        state          : {start, setup, ready, read, run, write};
        previous_state : {start, setup, ready, read, run, write};
        readysignal    : {false, true}; tdelay : 0..5;
ASSIGN
        init (state)         := start;
        init (readysignal)   := false;
        init (tdelay)        := 0;
next (state) := case
        serverstate = initial_write & tdelay=0  : setup;
        serverstate = pre_read & tdelay=0       : ready;
        serverstate = read & tdelay=0           : read;
        serverstate = pre_write & tdelay=0      : run;
        serverstate = write & tdelay=0          : write;
        serverstate = pre_read & tdelay=0       : ready;
        TRUE                                    : state;
esac;
next (previous_state) :=state;
next (readysignal) := case
        state!=previous_state              : false;
        (readysignal=false & tdelay=0) : {false, true};
        TRUE                               : readysignal;
esac;
next (tdelay) := case
        tdelay=0                     : {0..5};
        (previousserverstate=pre_read &
        serverstate=read)            : tdelay - 1;
        TRUE                         : tdelay;
esac;
MODULE server (sync)
VAR
        state : {start, initial_write, pre_read, read, pre_write,
                write};
        previous_state : {start, initial_write, pre_read, read,
                pre_write, write};
        tv: 0..1000;
ASSIGN
        init (state) := start;
        init (tv)    := 0;
next (state) := case
        (state = start)                  : initial_write;
        (sync & state = initial_write) : pre_read;
        (sync & state = pre_read)       : read;
        (sync & state = read)            : pre_write;
        (sync & state = pre_write)      : write;
        (sync & state = write)           : pre_read;
        TRUE                             : state;
esac;
next (previous_state):=state;
next (tv):= case
        tv=1000                                   : 0;
        (previous_state=pre_read & state=read) : tv+1;
        TRUE                                      : tv;
esac;
MODULE main
VAR
        simulationserver: server((client1.readysignal=true |
        client1.tdelay!=0) & (client2.readysignal=true |
        client2.tdelay!=0));
        client1: client (simulationserver.state,
        simulationserver.previous_state, simulationserver.tv);
        client2: client (simulationserver.state,
        simulationserver.previous_state, simulationserver.tv);
– Client 1 follows client 2, valid when the time
– delays for the different clients are equal.
SPEC
        AG (client1.state=client2.state)
– Liveness test (no deadlock)
SPEC
        AG EF (simulationserver.state=write)
– Specific order
SPEC
        AG (simulationserver.state=read → E
        [simulationserver.state=read U
        simulationserver.state=pre_write])
```

REFERENCES

[1] Y. Itoh, M. Fukagawa, T. Nagao, T. Mizuya, I. Miyazawa, and T. Sekiguchi, “Evaluation of execution time in programmable controller,” in *Proc. IEEE Symp. Emerging Technol. Factory Autom., ETFA*, 1999, pp. 1373–1379.

[2] J. Bathelt and J. Meile, “Computer aided methods supporting concurrent engineering when designing mechatronic systems controlled by a PLC,” in *Proc. ICMA’07*, Singapore, 2007.

[3] M. Pellicciari, A. Andrisano, F. Leali, and A. Vergnano, “Engineering method for adaptive manufacturing systems design,” *Int. J. Interactive Design Manuf.*, vol. 3, pp. 81–91, 2009.

[4] K. Thramboulidis, "Model-integrated mechatronics—Toward a new paradigm in the development of manufacturing systems," *IEEE Trans. Ind. Informat.*, vol. 1, no. 1, pp. 54–61, Feb. 2005.
[5] P. Hoffman, T. M. A. Maksoud, R. Schuman, and G. C. Premier, "Virtual commissioning of manufacturing systems a review and new approaches for simplification," in *Pro. 24th Eur. Conf. Modeling and Simulation*, 2010.
[6] R. Drath, P. Weber, and N. Mauser, "An evolutionary approach for the industrial introduction of virtual commissioning," in *Proc. IEEE Int. Conf. Emerging Technol. Factory Autom.*, 2008, pp. 5–8.
[7] D. Thapa, P. Chang Mok, S. Dangol, and W. Gi-Nam, "III-phase verification and validation of IEC standard programmable logic controller," in *Comput. Intell. Modeling, Control Autom., Int. Conf. Intelligent Agents, Web Technol. Internet Commerce*, 2006, pp. 111–111.
[8] G. Reinhart and G. Wünsch, "Economic application of virtual commissioning to mechatronic production systems," *Prod. Eng.*, vol. 1, pp. 371–379, 2007.
[9] M. F. Zaeh, C. Poernbacher, and J. Milberg, "A model-based method to develop PLC software for machine tools," *CIRP Ann.—Manuf. Technol.*, vol. 54, pp. 371–374, 2005.
[10] R. Bernhardt, A. Sabov, and C. Willnow, "Virtual automation system standards," in *Proc. IFAC Cost Oriented Autom.*, Gatineau/Ottawa, Canada, 2004, pp. 43–48.
[11] NuSMV, Jun. 10, 2011. [Online]. Available: http://nusmv.fbk.eu
[12] P. Klingstam and P. Gullander, "Overview of simulation tools for computer-aided production engineering," *Comput. Ind.*, vol. 38, pp. 173–186, 1999.
[13] H. C. Ng, "An integrated design, simulation and programming environment for modular manufacturing machine systems," in *Mechatronics Research Group Faculty of Computing Sciences and Engineering De Montfort University*, United Kingdom, 2003.
[14] S. Cho, "A distributed time-driven simulation method for enabling real-time manufacturing shop floor control," *Comput. Ind. Eng.*, vol. 49, pp. 572–590, 2005.
[15] R. P. J. Q. Ma and R. Lipset, "Distributed manufacturing simulation environment," in *Proc. Summer Computer Simulation Conf.*, 2001.
[16] S. C. Park, C. M. Park, G. N. Wang, J. Kwak, and S. Yeo, "PLCStudio: Simulation based PLC code verification," in *Proc. Winter Simulation Conf.*, 2008, pp. 222–228.
[17] J. Ledin*, Simulation Engineering, Build Better Embedded Systems Faster*. Lawrence, KS: CMP Books, 2001, .
[18] S. Kain, F. Schiller, and S. Dominka, "Reuse of models in the lifecycle of production plants using HiL simulation models for diagnosis," in *Proc. IEEE Int. Symp. Ind. Electron.*, 2008, pp. 1802–1807.
[19] D. Maclay, "Simulation gets into the loop," *IEE Review*, vol. 43, pp. 109–112, 1997.
[20] E. Freund, A. Hypki, R. Bauer, and D. H. Pensky, "Real-time coupling of the 3D workcell simulation system COSIMIR [registered trademark]". Bathurst, Australia, pp. 645–650, 2002.
[21] H. Schludermann, T. Kirchmair, and M. Vorderwinkler, "Soft-commissioning: Hardware-in-the-loop-based verification of controller software," in *Proc. Winter Simulation Conf.*, Orlando, FL, 2000, vol. 1, pp. 893–899.
[22] R. Bernhardt, G. Schreck, and C. Willnow, "Realistic robot simulation," *Comput. Control Eng. J.*, vol. 6, pp. 174–176, 1995.
[23] R. Bernhardt, G. Schreck, and C. Willnow, "The virtual robot controller interface," in *ISATA Autom. Transp. Technol. Simulation and Virtual Reality*, Dublin, Ireland, 2000.
[24] R. Bernhardt, G. Schreck, and C. Willnow, "Development of virtual robot controllers and future trends," in *Proc. 6th IFAC Symp. Cost Oriented Autom.*, Berlin, Germany, 2001, pp. 209–214.
[25] M. H. Schwarz and J. Boercsoek, "A survey on OLE for process control (OPC)," in *Proc. 7th Conf. Int. Conf. Appl. Comput. Sci.*, Venice, Italy, 2007, vol. 7, pp. 192–196.
[26] OPC Foundation [Online]. Available: http://www.opcfoundation.org, 2010-10-01
[27] X. Hong and W. Jianhua, "Using standard components in automation industry: A study on OPC specification," *Comput. Standards Interfaces*, vol. 28, pp. 386–395, 2006.
[28] OPC Foundation, "Data Access Custom Interface Standard," ver. 3.00, 2003.
[29] MatrikonOPC, Oct. 1, 2010. [Online]. Available: www.matrikonopc.com
[30] F. Iwanitz, "XML-DA opens windows beyond the firewall," in *Online Industrial Ethernet Book*. Titchfield, Hampshire, U.K.: GGH Marketing Commununications, 2004.
[31] S. Cavalieri and G. Cutuli, "Performance evaluation of OPC UA," in *Proc. IEEE Conf. Emerging Technol. Factory Autom.*, 2010, pp. 1–8.
[32] W. Mahnke, S.-H. Leitner, and M. Damm*, OPC Unified Architecture*. Berlin, Germany: Springer, 2009.
[33] R. Kondor, "OPC, XML,. NET and real-time application," Matrikon Inc., 2007.
[34] PLCOpen, Oct. 1, 2010. [Online]. Available: http://www.plcopen.org
[35] OMG, Oct. 1, 2010. [Online]. Available: http://www.omg.org
[36] Object Management Group (OMG), "Data acquisition from industrial systems specification," ver. 1.1, 2005.
[37] V. F. Wolfe, L. C. DiPippo, R. Ginis, M. Squadrito, S. Wohlevera, I. Zykh, and R. Johnston, "Real-time CORBA," *Proc. Real-Time Technol. Appl.*, pp. 148–157, 1997.
[38] I. McGregor, "The relationship between simulation and emulation," in *Proc. Winter Simulation Conf.*, San Diego, CA, 2002, pp. 1683–1688.
[39] CoDeSys, Oct. 1, 2010. [Online]. Available: http://www.3s-software.com
[40] Binar AB, Oct. 1, 2010. [Online]. Available: http://www.binar.se
[41] Integration Objects, Jul. 4, 2010. [Online]. Available: http://www.integ-objects.com
[42] F. Xiang, "Towards real-time enabled microsoft windows," in *Proc. 5th ACM Int. Conf. Embedded Softw.*, 2005, pp. 142–146.
[43] F. M. Proctor and W. P. Shackleford, "Real-time operating system timing jitter and its impact on motor control," in *Proc. SPIE-Int. Soc. Opt. Eng.*, 2001, vol. 4563, pp. 10–16.
[44] B. Lincoln and A. Cervin, "Jitterbug: A tool for analysis of real-time control performance," in *Proc. IEEE Conf. Decision Control*, 2002, pp. 1319–1324.
[45] D. Chen, A. Mok, and M. Nixon, "Real-time support in COM," in *Proc. 32nd Annu. Hawaii Int. Conf. Syst. Sci.*, 1999, p. 87.
[46] M. Fabian and A. Hellgren, "PLC-based implementation of supervisory control for discrete event systems," in *Proc. IEEE Conf. Decision Control*, 1998, vol. 3, pp. 3305–3310.
[47] F. Danielsson, "A distributed system architecture for optimizing control logic in complex manufacturing systems," in *Proc. ISCA 12th Int. Conf.*, Atlanta, GA, 1999, pp. 163–167.
[48] B. Svensson, D. Danielsson, and B. Lennartson, "A virtual real-time model for control software development—Applied on a sheet-metal press line," in *Proc. 3rd Int. Ind. Simulation Conf.*, Berlin, Germany, 2005, pp. 119–123.
[49] T. LeBaron and K. Thompson, "Emulation of a material delivery system," in *Proc. Winter Simulation Conf., Part 2 (of 2)*, Washington, DC, 1998, pp. 1055–1060.
[50] *Programmable Controller—Part 3: Programming Languages*, IEC 61131-3, 2003, 2nd ed..
[51] R. W. Lewis, Programming Industrial Control Systems Using IEC 1131-3 Revised edition. London, Institution of Electrical Engineers, 1998.
[52] A. Hellgren, M. Fabian, and B. Lennartson, "On the execution of sequential function charts," *Control Engineering Practice*, vol. 13, pp. 1283–1293, 2005.
[53] J. Proenza and S. Vitturi, "Guest editorial special section on industrial communication systems," *IEEE Trans. Ind. Informat.*, vol. 6, no. 3, pp. 365–368, Aug. 2010.
[54] F. Benzi, G. S. Buja, and M. Felser, "Communication architectures for electrical drives," *IEEE Trans. Ind. Informat.*, vol. 1, no. 1, pp. 47–53, Feb. 2005.

Henrik Carlsson was born in Falkenberg, Sweden, in 1979. He received the M.S. degree in robotics from University West, Trollhättan, Sweden, in 2005. Currently, he is working towards the Ph.D. degree at University West.

He is working as a Simulation Expert at Volvo Cars Corporation, Gothenburg, Sweden. His main areas of interest include virtual commissioning, robot simulation, and PLM systems.

Bo Svensson was born in Mariestad, Sweden, in 1959. He received the M.S. degree in electrical engineering from Chalmers University of Technology, Gothenburg, Sweden, in 1984. Currently, he is working towards the Ph.D. degree in automation at Chalmers University of Technology, and received the Lic.Eng. degree in 2010.

He was a Design Engineer with SAAB Space AB from 1984 to 1987. From 1987 to 1994, he has been with SAAB Automobile AB as a System Engineer. Since 1994, he has been with the Department of Engineering Science, University West, Trollhättan, Sweden, as a Researcher. His main research interests include simulation-based optimization and virtual commissioning of complex manufacturing applications.

Fredrik Danielsson was born in Orust, Sweden, in 1972 . He received the Ph.D. degree in mechatronics from De Montfort University, Leicester, U.K., in 2002.

Since 2003, he has been the Head of the Robot Education at advanced level. Since 2004, he has been Head of the Automation Research Group at the Department of Engineering Science, University West. His main research interests include flexible automation, virtual commissioning, and robot systems.

Bengt Lennartson (M'10) was born in Gnosjö, Sweden, in 1956. He received the Ph.D. degree in automatic control from Chalmers University of Technology, Gothenburg, Sweden, in 1986.

Since 1999, he has been a Professor of the Chair of Automation, Department of Signals and Systems. He was Dean of Education at Chalmers University of Technology from 2004 to 2007, and since 2005, he is a Guest Professor at University West, Trollhättan. He is (co)author of two books and $\sim$180 peer reviewed international papers with $>$2200 citations (GS). His main areas of interest include discrete event and hybrid systems, especially for manufacturing applications, as well as robust feedback control.

Prof. Lennartson was the Chairman of the Ninth International Workshop on Discrete Event Systems, WODES'08, Associate Editor for *Automatica*, and currently he is a member of the Advisory Board for the IEEE TRANSACTION ON AUTOMATION SCIENCE AND ENGINEERING.