Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

General Time Synchronisation Method for PLC Programs

Aiming at virtual verification and development with CAPE tools

Henrik Carlsson*, Fredrik Danielsson*, Bengt Lennartson**
*Department of Engineering Science at University West, SE-46186 Trollhättan, Sweden
(Tel: +46-520-223266; e-mail: Henrik.carlsson@hv.se, Fredrik.danielsson@hv.se)
**Control and Automation Laboratory, Department of Signal and Systems,
Chalmers University of Technology, SE-412 96 Göteborg, Sweden (e-mail: bengt.lennartson@chalmers.se)

Abstract: The latest state-of-the-art Computer Aided Production Engineering (CAPE) simulation technology offers OPC integration for PLC verification. A critical drawback with this technology has been identified and described within this paper. A new time synchronisation method and a simulation architecture are therefore presented and proposed. The time synchronisation method together with the architecture can be used when verifying and developing real-time dependent control logic for industrial control systems, e.g. PLC with CAPE tools. The method described in this paper is general and should work on any PLCs that are compatible with the IEC 61131-3 standard. A test case was also carried out, showing that by disregarding time synchronisation it is impossible to verify real-time dependent PLC functions together with CAPE tools in a reliable way. However, the test case also shows that by applying the proposed time synchronisation method together with the described simulation architecture a successful industrial verification method is achieved.

1. INTRODUCTION

The term industrial control system (ICS) is a broad definition for programmable controllers used in industry to control machines or processes. An example of an ICS commonly used in industry today is the PLC (Programmable Logic Controller). A characteristic feature of ICSs is the reprogrammable control function that is described by the control code. The most common control code development and verification method is to program and verify the control code without any connection to a real process, instead using manual stimuli to simulate the real process. Other existing methods are to connect the real ICS to the real process (Furusawa, 2002) or to a specially built process hardware model (mock-up) (Schludermann et al., 2000). Today, however, it is possible to verify and develop the control code against a virtual prototype using simulation software (Freund et al., 2002). One recent trend in conducting verification by means of a virtual prototype is to connect a real ICS to a CAPE simulation tool. CAPE (Computer Aided Production Engineering) is a classification for production-related simulation tools. The most common CAPE tools used for ICS development and verification are discrete event-based simulation (DES) and Computer Aided Robotics (CAR) (Ng, 2003), (Cho, 2005), (Qingwei Ma, 2001). A common feature of all of these tools is their ability to simulate several types of production scenarios on different levels where a variety of robots, machines, manufacturing resources, control logic representation etc. are integrated into the simulation.

The latest state-of-the-art technology offers OPC integration. OPC is a standard for information exchange between the ICS and its surrounding environment. OPC has and is still developed by the OPC Foundation (OPC Foundation, 2007-04-05). The OPC server, which is vendor specific, is connected to the ICS (Ling et al., 2004). The OPC clients e.g. as implemented in CAPE tools, are general and can connect to any server. This makes it possible to connect to an ICS in a general way, even if the OPC server is vendor specific.

This OPC connection makes the inclusion of a real ICS in a CAPE simulation possible. A simulated machine or process can thereby be controlled in the same way as it would be in reality. Examples of CAPE software with an OPC functionality include; Delmia Automation (Delmia, 2007-04-04), Visual Components (VisualComponents, 2007-04-04) and Process Simulate (UGS, 2007-04-04). This control code verification approach works well if the control function is event-driven. However, its efficacy is limited if the control function is time-dependent. Examples of time-dependent control functions include timers, servo control loops etc. In essence, the problem is that a regular PC and its operating system is not a real-time system (Xiang, 2005), and a non real-time system is not designed to respond to time-dependent signals in a deterministic way. Since most of today's CAPE simulation tools are Windows or Unix-based there is obviously a problem with the verification of time-dependent control functions. Even if it was possible to use CAPE tools in a real-time operating system, there would be time uncertainties due to the microprocessors in regular computers (Proctor and Shackleford, 2001).

Connecting a real-time system to a non real-time system, without any time synchronisation mechanism, can be referred to as 'free-wheeling', since no common time exists. Despite of the fact that the method itself introduces time related problems, free-wheeling appears to be the most commonly used method today when verifying the control code in real ICSs against CAPE simulations, The most common solution found in the literature to the freewheeling problem is time synchronisation methods based on the assumption that the CAPE tool always runs faster than the ICS (Freund et al.,

2002, Schludermann et al., 2000, Zaeh et al., 2005). According to this assumption, it should be possible to halt, when necessary, the faster CAPE tool in order to maintain real-time behaviour. However, these time synchronisation methods do not work for control functions with a time resolution that is close to, or indeed smaller than the CAPE cycle time.

Another problem encountered, when verifying control functions against a complex simulation model, relates to the CAPE simulation performance. When one or more ICSs interact with an extensive plant model, the CAPE execution time can become an obstacle. A solution to this is to distribute the simulation to parallel CPU's or computers (Ledin, 2001). However, in order to be able to properly handle distributed time-dependent simulations, both an architecture and a time synchronisation mechanism are required.

A simulation architecture that includes a time synchronisation mechanism and supports distribution is presented in (Danielsson, 1999). The work reported in this paper uses a simulation-architecture, SDSP (Synchronised Distributed Simulation Protocol), which handles the time synchronisation between an *emulated* ICS and the CAPE tool. The time synchronisation mechanism guarantees real-time behaviours in a deterministic way by introducing virtual real-time. The emulation of ICS is a technique used to obtain an exact representation of the real ICS (LeBaron and Thompson, 1998). The ICS emulator used in (Danielsson 1999) has been extended to include a number of additional SDSP functionalities, thus making it possible to use it in conjunction with the time synchronisation mechanism. To the author's knowledge, no working general method currently exists that address the described problem for a *real* ICS connected to CAPE tools.

The latest state-of-the-art CAPE tools offers OPC integration; however it is not possible to use this concept when verifying time dependent control functions as is shown and discussed in detail in this paper. To tackle this major drawback an extension of the work reported in (Danielsson 1999) is presented, in the form of a new method to handle time synchronisation with a real ICS. This method is based on general IEC 61131-3 languages and can thus be applied to a wide range of commercial ICSs.

2. SDSP SIMULATION ARCHITECTURE

Due to the lack of suitable deterministic methods for the verification of ICSs together with CAPE tools, an architecture for distributed simulation and time synchronisation - Synchronised Distributed Simulation Protocol (SDSP) - was formulated in (Danielsson, 1999). The concept is server-client based, with the server being responsible for managing the simulation. Management tasks include:

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

Whilst the SDSP communication is mainly based on TCP/IP, other methods of communication are also possible, such as, for instance, DDE, OPC and shared memory. One of the most important tasks for the server is to manage each subset of simulations to form a 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 that makes it possible to write a SDSP-client application that can handle the communication between the CAPE tool and the SDSP server. This is a common feature for many CAPE tools e.g. ROSE for Robcad and Tecnomatix .NET SDK for Process Simulate (UGS, 2007-04-04).

In essence, the server contains common data (e.g., I/O values, servo values, and simulation parameters) and common logic. The server also contains a virtual time, $t_{virtual}$ and a simulation state *s, s*={*simulation start, simulation initial-write*, s*imulation pre-write,* s*imulation write,* s*imulation pre-read, simulation read*}. Initially, the server sets up the necessary structure in the database and sets the simulation state to start, *s=simulation start*. As a minimum requirement to form a simulation, the server needs information about the time step t_{step}, the formal start condition, *simulation start condition,* and the number of clients. The start condition tells the server when the entire simulation can commence, and generally it is set to be the total number of clients required to run the simulation, see Figure 1. t_{step} is the smallest time step that the simulation can handle. However, a mechanism allows a client to run with a different time step that is a multiple of t_{step}, than the rest of the simulation without sacrificing time synchronisation. For example, this mechanism can be used in discrete event simulations to delay the synchronisation for a specific client until $t_{next\ update}$. Normally $t_{next\ update}$ is equal to $t_{virtual}$.

A single client is considered as ready to start when it has been connected to the server and has joined the simulation. When *simulation start condition* = AND($client_1$, $client_2$, …, $client_n$), where *n* is the number of clients in the simulation, is fulfilled, the initial write state *s=simulation initial write* is entered. This mechanism provides a deterministic start behaviour for the overall simulation. In the initial write state, each client has reached their *client setup* state where they are supposed to initialise their own data (e.g. all inputs and outputs are set to 0), whilst the simulation time $t_{virtual}$ is set to $-t_{step}$. To enter the following states, the *Synchronised* condition must be fulfilled;
synchronised = AND(*client* $sync_1$, *client* $sync_2$, …, *client* $sync_n$). The *client* $sync_x$ signal indicates when a single simulation client has arrived the point when it is ready and has executed the current time step or that the synchronisation for the client has been delayed until $t_{next\ update\_x}$;
client $sync_x$=*OR(client ready* $signal_x$, $t_{virtual} < t_{next\ update\_x}$*),* see Figure 2.

After the *simulation initial-write* state, the simulation enters the *simulation pre-read* and then the *simulation read* state, see Figure 1. In synchronisation each client has reached the *client read* state, see Figure 2. At this state, each client is supposed to read data from the server. When all of the clients

have executed their client read task and thereby synchronised the simulation, then the simulation enters the pre-write state, *s=simulation pre-write*

The simulation time, $t_{virtual}$, is updated to the next time step at the end of each *simulation read* state, *s=simulation read* → $t_{virtual}=t_{virtual}+t_{step}$. This tight synchronisation procedure is necessary to prevent clients from acting in the wrong time space i.e. one time step before or after the desired time.

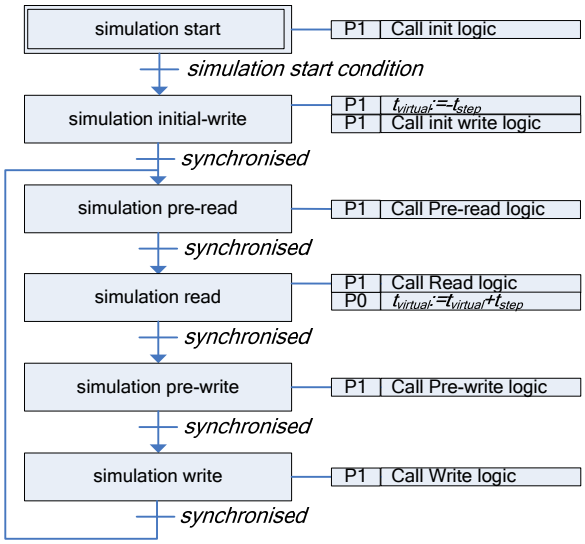

Figure 1 *States and associated actions for the overall simulation as represented in the SDSP server, described in SFC.*

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 an ICS.

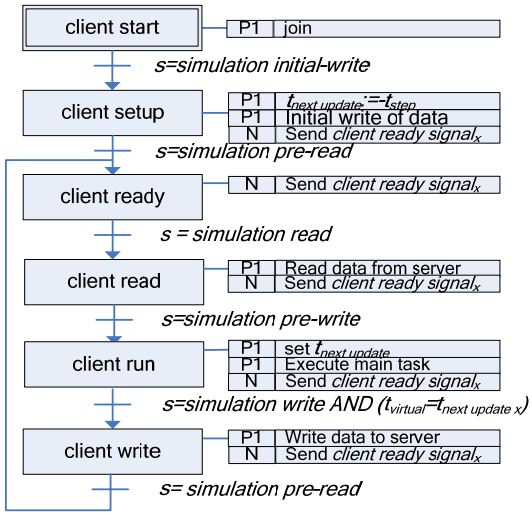

Figure 2 *States and associated actions for each client (sub-simulation) as represented in the SDSP server, described in SFC. Note that in our SFC notation must the P1actions be completed before the N actions start.*

Logic can be assigned to each state in the simulation, e.g., a logic function can be called when the time is updated and another function can be called when the simulation is commenced, see logic action in Figure 1. These user-defined logics can be described in IEC 61131-3, C/C++ or SBASIC, which is an internal programming language in the SDSP server. User-defined logics can also be assigned to data variables, e.g., an I/O signal. Each time an assigned data variable is accessed, the desired user logic is called. With user-defined logics at the server level, it is easy to create an overall system logic that will operate at a specific time interval, record data at each time step, and act whenever a variable changes etc.

3. ICS TIME-SYNCHRONISATION

The SDSP simulation architecture presented above is designed to handle verifications of time-dependent control functions in ICSs connected to CAPE tools, i.e. to deal with the free-wheeling problem described previously in the introduction.

The internal clock in an ICS controls the execution to guarantee the cycle time and to achieve deterministic behaviour. Due to this fundamental behaviour in the real ICS it has only been possible to incorporate emulated or simulated ICS in the SDSP simulation architecture. Even though an emulator is a good representation of the real ICS, a real ICS is nevertheless preferable in many situations. This is because of the lack of emulators coupled with the fact that creating an emulator can be extremely time-consuming,

Even if it is possible to connect ICSs to the simulation architecture over OPC, it is generally not possible to directly time synchronise them with the mechanism within the architecture. The reason is that most commercially available ICSs are vendor specific and not open for new internal functions – such as SDSP – in the scheduler. Thus, there is a problem in adopting this synchronisation method to existing ICSs. To overcome this hurdle, a general method for time synchronisation of IEC61131-3 based ICSs is described in this section.

In order to utilise the new time synchronisation method the following requirements must be fulfilled:

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

IEC 61131 is an international standard for programmable controllers, and is divided into several parts. IEC 61131-3 describes an ICS software structure, languages and program execution (Lewis, 1998). The standard supports five different languages, two that are text-based, the structured text (ST) and instruction list (IL) and three that are graphical, namely, the ladder diagram (LD), function block diagram (FBD) and sequential function chart (SFC). IEC 61131-3 covers all languages at syntax level but lacks a detailed definition on the execution level (Hellgren et al., 2005). An IEC 61131-3 project consists of a number of Program Organisation Units (POU). A POU can be a program, a function block, or a function.

In order to use the method presented in this paper, three modifications of the original control code to be verified need to be made. (1) A scheduler 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 the virtual time. The marked sections in Figure 3 represent these modifications and will be described below.

3.1 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 shall guarantee that the scheduler is executed first every program cycle. The scheduler communicates with the SDSP architecture through two synchronisation variables over e.g. OPC.

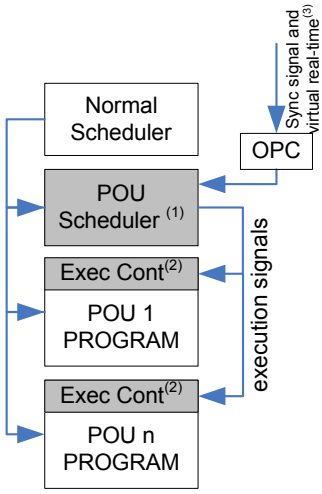

Figure 3 *Description of the supervisor POU and the connection to the normal POU's.*

When it is time to execute the control code in the ICS, 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 Figure 3. There is no standard or general way to halt the real-time clock on a real ICS, as demanded by the simulation architecture. Consequently, the scheduler also handles the virtual time, t_o, by reading the actual 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.

3.2 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 below, determines whether or not the desired program should run.

```
IF NOT Simulated OR execution(task_i)
    (*Regular program begins here*)
    …
    (*Regular program ends here*)
END
```

However, the graphical language SFC has a more complex execution order that requires another type of execution controller (Hellgren et al., 2005). An SFC consists of a series of steps and transitions, where each step can be associated with either one or a series of actions (Lewis, 1998), the behaviour 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 61131-3 standard (IEC 61131-3, 2003) 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 that is used. Although it is not a requirement, according to the 61131-3 standard that this function block actually exists (IEC 61131-3, 2003), (Lewis, 1998), the activation and deactivation of the actions should nevertheless have the same behaviour that is specified within the action control function block. Due to the fact that some of the action qualifiers are time dependent, the ICS programming environment must offer access to modify the action control function block in order to be able to use the proposed method within a SFC POU. An example of an execution control function added to an action control function block is shown in Figure 4. The execution control function block is implemented in CoDeSys (CoDeSys, 2007-05-02).

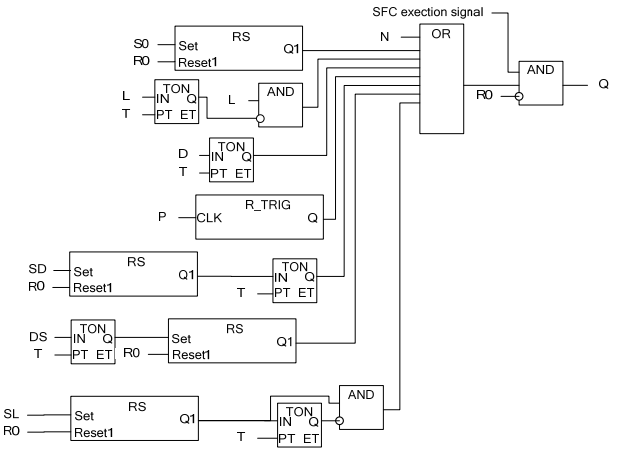

Figure 4 *Example of a function block implementation of an Action Control block.*

In Figure 4, *'SFC execution signal'* is the execution signal that is sent from the scheduler when it is time to execute the SFC POU.

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. However should this not be possible, the same behaviour can be obtained within the actions.

3.3 Time dependent functions (3)

IEC 61131-3 contains 4 standard timer function blocks, Timer On Delay (TON), Timer Off Delay (TOF), Timer Pulse (TP) and Real-Time Clock (RTC) (IEC 61131-3, 2003). However these function blocks can not be used directly, since they are usually based on the hardware clock. Therefore, special replacement timers are used. The replacement timers behave in the same manner as the regular ones, the only difference being that they are based on the virtual time received from the server.

There may also be other (no IEC 61131-3) vendor-specific functions that are hardware clock dependent, e.g., motion control blocks. Thus, in order to be able to use the proposed time synchronisation method these functions must also be upgraded to handle the virtual clock.

For motion control, PLCOpen (PLCOpen, 2007-09-18) offers a motion control library specification based on IEC 61131-3. It is thus possible to implement this specification together with the proposed time synchronisation method and virtual time.

4. APPLICATION TEST CASE

To demonstrate the proposed time synchronisation method, a test case was set up. A real ICS from Binar (Binar AB, 2007-09-03) was programmed to control a two dimensional servo controlled robot, see Figure 5. The control function is real-time dependent due to servo control loops in the robot. A simulation model was created in a CAPE tool, RobCad (UGS, 2007-04-04). The model represents the 3D geometry and kinematics of the robot. Two scenarios where carried out:
In Scenario 1, the CAPE tool was connected to the ICS in an unsynchronised way (free-wheeling) currently provided by CAPE tools. Further, in scenario 2, the model was connected according to the synchronisation method proposed in this paper.

To be able to utilise the new time synchronisation method, scenario 2, the following preparations were carried out in the ICS control logic:

- All tasks in the ICS 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 synchronisation.

The robot positions were measured in the CAPE tool and plotted, (1) scenario 1 with no time synchronisation , Figure 6, and (2) scenario 2 with the proposed time synchronisation method, in Figure 7.

5. TEST CASE RESULT

Figure 6 shows the result from the run when no time synchronisation is used, scenario 1. The enlarged part showing the non-deterministic behaviour when no time synchronisation is employed (compared to the pre-programmed desired behaviour in Figure 5). This non-deterministic behaviour is due to the free-wheeling problem described earlier.

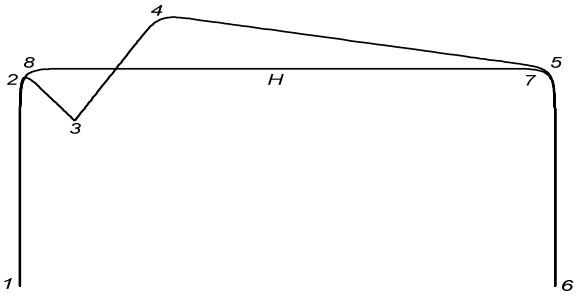

Figure 5 *The pre-programmed Robot path used in scenario 1 and 2. The figures represent the run order of the path and represent a 2D location (x, z).*

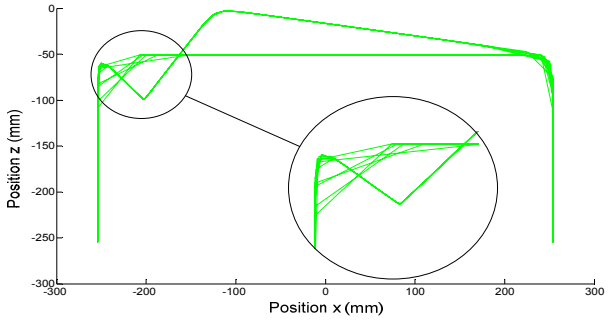

Figure 6 *Measured robot path with standard free-wheeling method used in CAPE tools.*

Figure 7 shows the result when the proposed time synchronisation method is used. The plot shows clearly that the non-deterministic behaviour from the free-wheeling method (Figure 6) is solved. The results show that this type of exact time synchronisation is necessary when verifying real-time dependent control functions with CAPE tools.

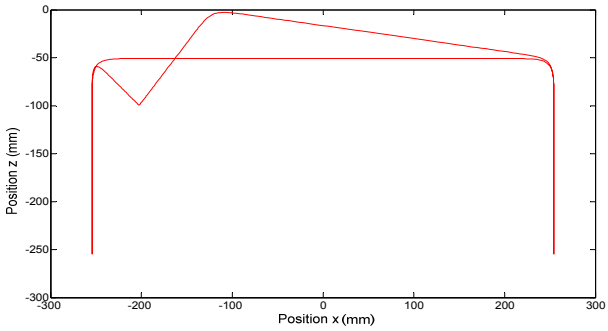

Figure 7 *Measured robot path with the new proposed time synchronisation method.*

6. CONCLUSION

In this paper a new time synchronisation method together with a simulation architecture has been presented. Such a method and architecture must be used when verifying and developing real-time dependent control logics for ICS with CAPE tools. It has been shown that without this synchronisation method the result will be a non deterministic verification. This non deterministic behaviour will indeed introduce virtual verification problems such as; false collision detection, wrong sensor signals etc. The proposed

synchronisation method together with the architecture has been demonstrated to work on ICSs that are compatible with the IEC 61131-3 standard. To implement this time synchronisation method in an ICS a new scheduler, execution controllers and replacement timers are necessary. For industrial usefulness, these method-specific requirements can be automatically generated for general IEC 61131-3 languages before the downloading of the control logic to the ICS. Notwithstanding the problem that there might be few ICS vendors that are 100% compatible with this standard, it is nevertheless possible to use this method with certain modifications.

One drawback with this method is that it produces some overhead code for the time synchronisation that is not necessary in the real application. However, if desirable, this overhead can be removed before commissioning the real application.

The test case scenario presented identifies problems that might arise when working with a non-time-synchronised verification method. From an industrial point of view, a time synchronisation method is necessary when verifying complex real-time dependent control functions. However, industrial state of the art verification methods based on CAPE lack this type of feature. Industries working with motion control related problems lack the possibility to verify ICS for motion control. It should be possible to extend this work and apply it on PLCOpen general motion control blocks.

ACKNOWLEDGEMENT

This work is being funded by the Swedish Foundation for Knowledge and Competence Development. The authors gratefully thank Tetra Pak in Lund, Sweden, and Binar AB in Trollhättan, Sweden, for all contributions and support.

REFERENCES

BINAR AB (2007-09-03) http://www.binar.se.

CHO, S. (2005) A distributed time-driven simulation method for enabling real-time manufacturing shop floor control. *Computers and Industrial Engineering,* 49**,** 572-590.

DANIELSSON, F. (1999) A distributed system architecture for optimizing control logic in complex manufacturing systems. IN YANG, Q. (Ed.) *ISCA 12th international conference.* Atlanta, USA.

DELMIA (2007-04-04) http://www.delmia.com.

FREUND, E., HYPKI, A., BAUER, R. & PENSKY, D. H. (2002) Real-time Coupling of the 3D Workcell Simulation System COSIMIR [registered trademark]. Bathurst, Australia, Charles Sturt University, Albury, NSW 2640, Australia.

FURUSAWA, K., YOSHIKAWA, T (2002) Synchronization mechanism in integrated simulation for manufacturing systems. *the 10th Mediterranean Conference on Control and Automation - MED2002.* Lisbon, Portugal.

HELLGREN, A., FABIAN, M. & LENNARTSON, B. (2005) On the execution of sequential function charts. *Control Engineering Practice,* 13**,** 1283-1293.

IEC 61131-3 (2003) *Programmable controller - Part 3: Programming Languages, second edition*.

LEBARON, T. & THOMPSON, K. (1998) Emulation of a material delivery system. *Proceedings of the 1998 Winter Simulation Conference, WSC. Part 2 (of 2), Dec 13-16 1998.* Washington, DC, USA, IEEE, Piscataway, NJ, USA.

LEDIN, J. (2001) *Simulation Engineering, Build better embedded systems faster,* Lawrence, USA, CMP Books.

LEWIS, R. W. (1998) *Programming industrial control systems using IEC 1131-3, Revised edittion,* London, Institution of Electrical Engineers.

LING, Z., CHEN, W. & YU, J. (2004) Research and implementation of OPC server based on data access specification. *WCICA 2004 - Fifth World Congress on Intelligent Control and Automation, Conference Proceedings, Jun 15-19 2004.* Hangzhou, China, Institute of Electrical and Electronics Engineers Inc., Piscataway, United States.

NG, H. C. (2003) An integrated design, simulation and programming environment for modular manufacturing machine systems. *Mechatronics Research Group Faculty of Computing Sciences and Engineering De Montfort University, United Kingdom (Sponsored by the University of Skövde, Sweden).* De Montfort University.

OPC FOUNDATION (2007-04-05) http://www.opcfoundation.org.

PLCOPEN (2007-09-18) http://www.plcopen.org.

PROCTOR, F. M. & SHACKLEFORD, W. P. (2001) Real-time operating system timing jitter and its impact on motor control. Boston, MA, United States, The International Society for Optical Engineering.

QINGWEI MA, R. P. J., ROBERT LIPSET (2001) Distributed Manufacturing Simulation Environment. *2001 Summer Computer Simulation Conference.*

SCHLUDERMANN, H., KIRCHMAIR, T. & VORDERWINKLER, M. (2000) Soft-commissioning: hardware-in-the-loop-based verification of controller software. *Proceedings of WSC 2000, Winter Simulation Conference, 10-13 Dec. 2000.* Orlando, FL, USA, IEEE.

UGS (2007-04-04) http://www.ugs.com.

VISUALCOMPONENTS (2007-04-04) http://www.visualcomponents.com.

XIANG, F. (2005) Towards real-time enabled Microsoft Windows. *Proceedings of the 5th ACM international conference on Embedded software.* Jersey City, NJ, USA, ACM Press.

ZAEH, M. F., POERNBACHER, C. & MILBERG, J. (2005) A model-based method to develop PLC software for machine tools. *CIRP Annals - Manufacturing Technology,* 54**,** 371-374.