Counterexample-Guided Simulation Framework for Formal Verification of Flexible Automation Systems

Sandeep Patil
Luleå University of Technology,
Luleå, Sweden
sandeep.patil@ltu.se

Valeriy Vyatkin
Luleå University of Technology,
Luleå, Sweden
Aalto University, Helsinki, Finland
vyatkin@ieee.org

Cheng Pang
Aalto University, Helsinki, Finland
cheng.pang.phd@ieee.org

***Abstract* —This paper proposes a framework for formal verification of industrial automation software in an intuitive way. The IEC 61499 function block architecture is assumed to be the input language, and the Intelligent Mechatronic Components (IMC) architecture is assumed as an underlying design pattern for the applications, which implies autonomous control logic in each IMC and their compositions to systems in a plug-and-play way. Then the system is automatically verified using model checking and the counter examples for the failed model checking properties are played back step-by-step and state-by-state in the simulation model that most industrial automation control systems would have built as the basis for initial testing. Net Condition Event Systems formalism (a modular extension of Petri net) is used to model the decentralized control logic and discrete-state dynamics of the plant. The model is then subjected to model checking using the ViVe/SESA tool chain. The method's application is illustrated using a simple pick and place manipulator. A closed loop model of Plant and Controller is used. Controller is extensively verified for safety, liveliness and functional properties of the robot. We then show how a counter example for deadlock detected by the model checker is played back in the simulation model for visualizing how exactly the system deadlocked.**

Keywords — NCES, ViVe, SESA, Formal Verification, Closed-Loop Modeling.

I. INTRODUCTION

Industrial automation is facing challenges related to a manufacturing change from mass production to mass customization. As a result, the focus of automation has been shifting to flexibility, re-configurability, and safety assurance. With this shift, the existing software verification and validation (V&V) techniques, such as testing and simulation, become inadequate. Furthermore, the development of simulation models is time consuming while does not guarantee 100% validation of the automation control software. To address this problem, formal verification [1] has been considered as a proper complementary V&V technique. Discrete state model checking [2] is one of such approaches, which is the process of automatically verifying whether a set of desired formal specifications is satisfied over the target system (model). While model checking is computationally resource hungry, it has been successfully used in other areas of computer system engineering, such as hardware design, proving its ability to handle problems of reasonably large complexity [3, 4]. This suggests that model checking can be applied in the industrial automation domain. There has been an impressive number of research works towards this goal.

Despite these (moderate) successes and promises the reality is that formal verification techniques are rarely used in the development practice by industrial automation engineers. It seems that the existing tools and methods do not fit to the processes of automation system engineering.

In this paper we propose a closed-loop verification framework which we believe will be more efficient and feasible in model checking of control systems in industrial automation domain. The framework consists of 4 main steps:

1) Developing a closed-loop function block application of the control system and a simulation model with initial testing of the control logic by simulation;
2) Automatic generation (and composition) of formal models as a closed-loop model;
3) Model checking the resulting formal model and verifying for a comprehensive set of requirements, which generates counter examples for any failed properties; and
4) Play back of the counter example from Step 3 in the same simulation model generated in Step 1.

The rest of the paper is organized as follows; Section II presents basics about IEC 61499. Section III related work and in section IV we explain each of the 4 steps of the proposed framework. Section V presents the verification results and how a counter example can be played in the simulation model. The paper will end with conclusions and future work discussion in section VI.

II. DESIGN MODELS FOR FLEXIBLE AUTOMATION

The International Electrotechnical Commission (IEC) has come with the IEC 61499 [5] standard for design of distributed automation systems [6] as an extension of the popular IEC 61131-3 standard. The standard aims at enhancing flexibility, interoperability[7] extensibility and re-configurability of distributed systems by better reusability components. This model has demonstrated its benefits for modular mechatronic automation systems [8-11].

The standard provides several design artefacts, such as basic and composite function blocks. The *basic function block* is used for encapsulating the developer's code, similar to the object-oriented design concept where the function block is a class defining the behaviour of multiple instances. The function blocks execution is event-driven which models the message passing communication in distributed systems. Interface of a function block consists of input and output events along with traditional data inputs and outputs as shown in Fig. 1(b).

978-1-4799-6648-6/15/$31.00 ©2015 IEEE

In order to relate event handling and program execution, each basic function block comprises of Execution Control Chart (ECC) and set of algorithms as shown in Fig. 1(d) and Fig. 1(e). The algorithms can be written in PLC standard programming languages (IEC 61131-3) or other languages, e.g. C, C#, or Java. The ECC is a state machine containing EC states, actions and transitions as shown in Fig. 1(d). The transitions are triggered by input events along with some Boolean guard conditions. The states have associated actions, in which algorithms are invoked. Once the algorithms are executed, output events are emitted. For example, the interface of the pick and place robot shown in Fig 3(a) can be specified using an IEC 61499 function block and its ECC would define its control logic [12].

There are several benefits of the new standard for software development, such as model-based design (Fig. 1(b)), component-encapsulation and ease of distribution, along with improved portability, interoperability and re-configurability of software. These benefits should result in design of systems with higher flexibility, fault tolerance and scalability than of current PLCs. Some of the available tools for IEC 61499 automation are NxtStudio [13], ISaGRAF [14] and Function Block Development Kit (FBDK) [15]. All these software tools have inbuilt support for visual simulation in order to support testing and validation. The availability of these new powerful tools has stipulated the development of automation systems with higher degree of logic decentralisation. Quite predictably

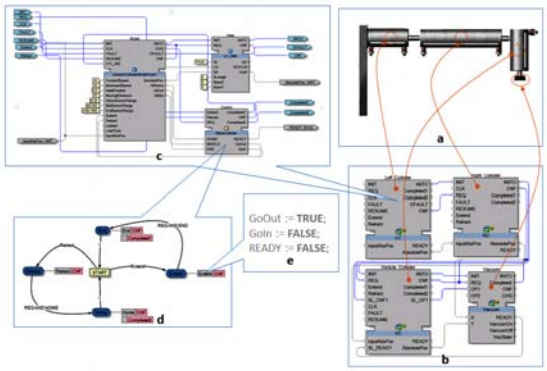

Fig. 1: (a) A pick and place robot, (b) Modular and distributed application, (c) Modularity inside a resource, (d) Basic function block ECC, (e) Algorithm inside basic function block.

that revealed the fundamental challenges of distributed systems verification and validation [16].

III. State-Of-The-Art in Verification and Validation

A. Model checking and different formalisms

Compared to some formal verification methods, such as theorem proving and equivalence checking, model checking has three notable advantages:

1) It enables the unsupervised automatic verification process of a system;

2) It identifies system failure via counter examples; and

3) It makes use of temporal logic for specifications so that model checker can automatically check for different properties (including safety properties).

Fig. 2 shows the typical framework of model checking exemplified on a simplistic single-cylinder automation system. First step involves formal modelling of the target system in a modelling language such as Net Condition/Event Systems (NCES)[17]. The second step is to feed the formal model and the properties specified using a language such as Computation Tree Logic (CTL) to be verified by model checker. In the third step the model checker tool such as ViVe/SESA [18] does the verification and analyses the state space for the given CTL properties and then outputs the counter examples for the failed specifications if any.

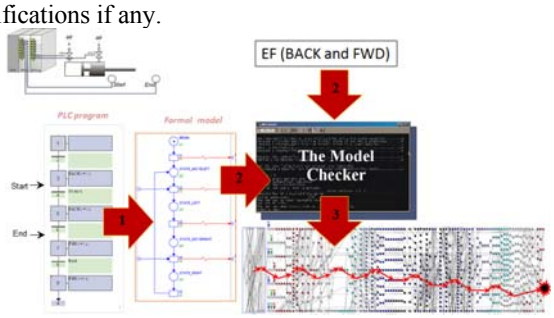

Fig. 2: Traditional model-checking based verification of automation software.

Finite state machines are widely used for the modelling of control flow and so is the formalism of NCES. There are mainly two types of tools that are being researched by academia and industries for model checking. Finite state machine tools such as UPPAAL [19] and SMV [20] that compute sets of reachable states exactly and effectively. There are second set of tools based on Petri-net formalism related tools like NCES and its flavours such as $_D$TNCES[21] and ViVe/SESA[18], that approximate sets of reachable states.

B. Related work in model checking of Automation and Control Systems

There has been quite a bit of interested in this area and lot of research works exists. In this paper we will limited the related work to model checking research to IEC 61499 function block systems.

One of the first works in the verification of IEC 61499 function block system is [22]. There have been many research works since then addressing various aspects of 61499 systems and its semantics[23]. [24] presents a very good overview of IEC 61499 formal modelling and verification. This paper is motivated and builds on top of the approach presented in [25]. There have been other works that stem from the same motivation. [26] presents a complete closed-loop implementation and verification framework based on the $_D$TNCES formalism. [27] also presents another framework with focus on hardware in loop (HiL) and software in loop (SiL) verification.

C. Summary from the litrature

In summary, the following gaps have been identified from the literature analysis and addressed in this paper.

- State space explosion is an obstacle for applying model-checking, this paper presents an approach to control the state space explosion by applying controller non-determinism [28].

- There is no formal verification tool-chain that integrates seamlessly with the IEC61499 development tools. This paper presents integration with FBDK (and in general IEC 61499 tools because the player presented in section IV.E is a function block implementation).
- The main output of the model checker in case of a failed specification is a counter example. Analysis of counter examples has not been fully addressed in the past research works, this paper presents visualisation of counter example in the actual simulation models as a compliment to the Gantt chart approach in [21, 25, 27] to help on-site engineers who have no knowledge about the formal model.
- There is no proper link between the simulation (visualisation) tools and the model checking tools (and the counter examples it generates).

This paper attempts to bridge these gaps by proposing an integrated framework for formal verification, the central part of which is the simulation-based investigation of counterexamples, resulted from model checking.

IV. The Proposed Framework

Fig. 3 below shows the proposed framework. As mentioned in the introduction it is a 4 step process. The first is the actual plant-controller model along with simulation. Second step is generation of the formal model. Third step is model checking and generation of state space and counter example (if any) and finally fourth step is to playback the counter example in the same simulation model developed in first step by implanting a player module in the function block module. The variety of simulation environments, e.g. as surveyed in [26, 29] can be used for the initial simulation model development.

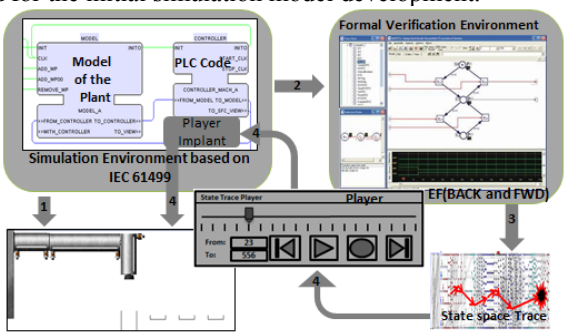

Fig. 3: The proposed framework

A. Case Study Example

To illustrate our approach, we will use a pick and place object shown in Fig. 4.

The system is composed of several mechatronic units as follows:

- There are two horizontal cylinders and a vertical cylinder that extract and retract. The left horizontal cylinder is half the size of the right cylinder. There is also a suction unit. The vertical cylinder picks up the work pieces using the suction unit attached to its end.
- Both horizontal cylinders have two control signals (CGO: Cylinder Go Out: extending, CGI: Cylinder Go In: Retracting). The vertical cylinder has only one control signal (VCGD: Vertical Cylinder Goes Done). When this signal is not active, the cylinder moves up (pulled by the internal spring).

Each of the cylinders have their own sensors that indicate the cylinder's home and end positions. There are also sensors in each of the three input trays (pp1, pp2 and pp3) and one in the slider (pp0) to indicate the presence of a work piece. The suction unit has a built-in sensor, vacuum indicating that a work piece is sucked.

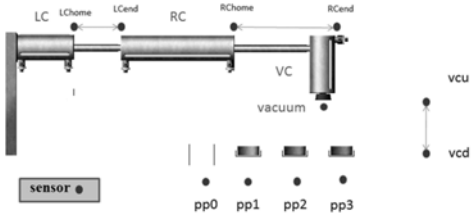

Fig. 4: Reference object: pick and place robot.

Fig. 5 shows the desired behaviour, it specifies the state of each cylinder and the vacuum unit when picking and dropping each of the work pieces.

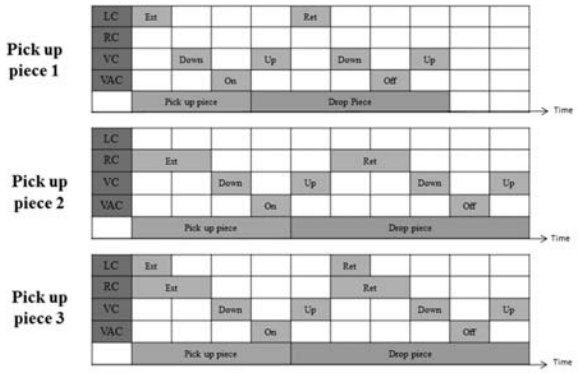

Fig. 5: Activity diagram of the pick and place robot.

B. Step 1: The Function Block Model

The function block model is built as a plug-and-play application using the IMC's described in the previous section. The methodology used by authors in [21, 30] is applied in order to create the function block application of the desired system. Fig. 6 below shows the controller for the case study example described in the previous section.

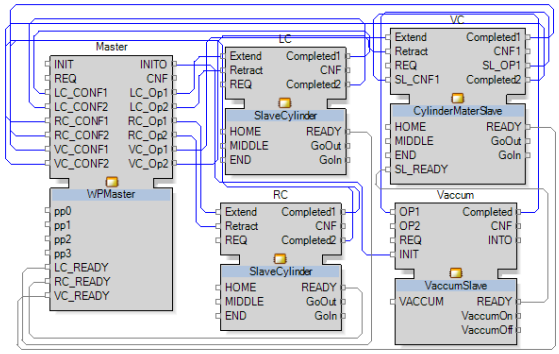

Fig. 6: Distributed controller of the robot with Master –Slave architecture implemented in IEC 61499 (only the important data and event connections are shown).

C. Step 2: Generation of the Formal Model in NCES

Similar to composition of function block application, the NCES model is also built using the plug and play approach making use of the pre-existing library models. In fact, in our tool-chain the controller's formal model of the system is automatically composed as presented in [31].

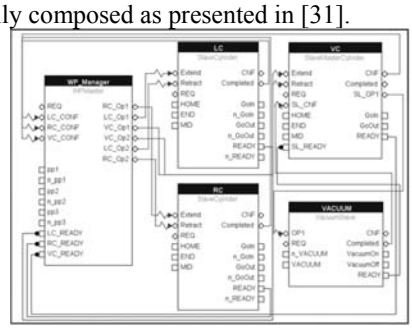

Fig. 7: NCES model of distributed controller of the robot implemented with Master –Slave architecture.

Fig. 7 shows the equivalent NCES model for the function block system shown in Fig. 6.

D. Step3: Model- Checking and Generation of Counter Example

The formal model is verified using ViVe/SESA [18] model checking tools. The properties to be verified are formulated using Computation Tree Logic (CTL) properties. Section V provides the complete details of model checking results. In case of a specification (CTL property in our case) is not true, the model-checking tool generates a counter example, which is a path in the resulting state space. There could be more than one path (counter example) for a single failed property (ViVe/SESA model checkers display a list of states where a property does not hold with possibly multiple paths to the same state). The model checker allows us to specify what variables need to be stored (for each state) in the resulting counter example trace. The counter example generated is stored as a text file consisting of multiple rows and columns. Each row refers to one state in the resulting path and each column in the row refers to the value of system variable in that state. Optionally the trace can also store timestamp of when particular events occur, which can be used for continuous playback in step 4. By time stamping we do not mean actual time when it occurred, but we mean discretized timestamp (discrete model time) [32], which we will use for continues playback.

E. Step4: Play back of the counter example from step 3 in the same simulation model generated in step 1

The player module is implemented as an IEC 61499 function block that reads the text file generated in step 3 and sets the plant model in a predefined state (force set the values of model variables in the plant model as given in the counter example trace's first state) in order to see controller behaviour for that particular value(s) combination. The user can select any state from the state space of the counter example trace and view the variable values in that state (in the simulation model). For example, the simulation model can be set to a start state where cylinder 1 is in extended state and work piece 1 and 3 are present. In the next state (or in any the preceding state), the vertical cylinder also will be extended (to pick work piece 1).

The player also supports playing the whole counter example trace as one continues simulation run making use of the timestamps in the trace text file. For example, in the formal model, the cylinder position is discretized [32] as 20%, 40%, 50%, 60%, 80%, 100% extending and similarly retracting. In NCES modelling each of the above values constitutes to 1 place and transition happens between place to place every 'tick' (fired when no other spontaneous transition are available). So in different states of the trace file, the values for position of cylinder (0, 20, 40, 50, 60, 80 100) and 'tick' value is stored. For simulation, we map 1 tick value to 1 time cycle (using standard library function block E_CYCLE) of the simulation model (cycle value in millisecond is configured by the simulation model using E_CYCLE). Hence instead of cylinder jumping from start state to end state, it gradually jumps according to discretized values. More discretization, the better continuous motion in the playback. Increased granularity must be used with caution as this might increase the state space and probably unreliable results.

V. Model Checking with Vive/SESA

The three cylinders system was checked using the ViVe and SESA tool chain. CTL was used to represent safety, liveliness and other functional requirements. The advantage of these tools is the ease with which properties can be mentioned. The properties are presented in terms of the places in the NCES models.

The ViVe tool flattens the whole model consisting of different sub modules into one (possibly huge) NCES model, the ViVe tools tree view of the flat model is shown in Fig. 8. The flattened model can be exposed then to SESA model checker.

For example, to check for the property that says "Opposite actuator signals (extend and retract) to the cylinders (C1, C2 in case of 3 cylinder model) should never be emitted at the same time", we write the property as "AG (NOT (p136ANDp137))", where places p136 and p137 correspond to global place numbers in the flattened NCES model variables. The global place numbers used by flattening the model aids in better analysis of the counter examples. Because we use a closed-loop model, the property can also be expressed in terms of the plant variables instead of the controller.

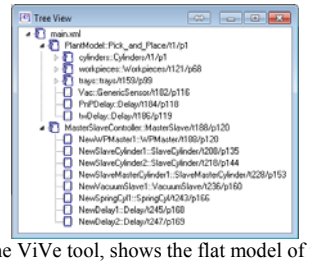

Fig. 8: Tree View of the ViVe tool, shows the flat model of the pick and place robot.

In NCES terminology, the tool checks if at all there is a possibility that a token can be present in both these places at any given time. The base NCES model of the plant (without our controlled non-determinism) was timed and deterministic.

This model was checked against a set of CTL specifications of safe and correct behaviour. Table 1 summarizes all the properties that were checked for our use case.

TABLE 1: LIST OF SAFETY, LIVENESS AND FUNCTIONAL PROPERTIES FOR THE PICK AND PLACE ROBOT.

	Specifications
Safety	Opposite actuator signals to the horizontal cylinders should never be emitted at the same time.
Safety	If the signal to descend the vertical cylinder is emitted, the horizontal cylinder should stand still.
Safety	If there is an emission of a control command corresponding to movements of the horizontal cylinders then the sensor "vcu" of all the vertical cylinders must be true.
Safety	The horizontal cylinders can move only if the value of sensor "vcu" of all vertical cylinders is true.
Liveness	Absence of deadlocks in the (decentralized) control logic.
Functional	If a part is detected by pp1, pp2 or pp3, then in future one of the horizontal cylinders will be extended.
Functional	If a part is detected by pp1, pp2 or pp3, then in the future, the part will be removed from the tray.
Functional	When the vertical cylinder goes down, both horizontal cylinders are (and remain) in their end positions (home or end).

The liveliness property of the cylinder, can be expressed as: all the places in the cylinder model become false (have no token) in future once they were true (had token). The format of the CTL property is:

AG (pXX -> EF (NOT (pXX))),

where "XX" corresponds to every place of the flattened controller model.

Let us consider a property defined as, "if only work piece 2 is present, only the right cylinder should extend and the work piece should be picked up". Fig. 9(a) shows a valid behaviour of the above property, left cylinder is still in retracted position and right cylinder is extended.

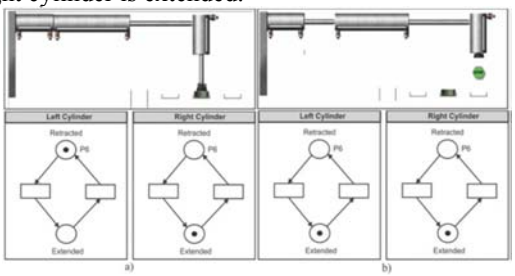

Fig. 9: a) Valid behaviour in a deterministic model; b) Invalid behaviour in a non-deterministic model.

Then the model was checked after non-determinism was introduced to model one abnormal behaviour of the plant (work piece disappears) due to unpredictable external influences (Ex: someone handpicked the work piece). The same property like in the above was checked i.e. "if only work piece 2 is present, only the right cylinder should extend and the work piece should be picked up". But due to non-deterministic model, where work piece disappearance was modelled, the property check failed. The failure scenario is illustrated in Fig. 9(b), which shows both left and right cylinders being extended and system in a deadlocked state. The ViVe/SESA tool also gives us a counter example trace showing how it failed. The sequence was:

1) All the three work pieces arrived.
2) Cylinder 1 starts to extend to pick up work piece 1.
3) Then work piece 1 and 3 disappear.
4) Cylinder 2 also starts extending and system enters a deadlocked state.

TABLE 2: NUMBER OF STATES NEEDED TO DETECT A FAILURE AND TIME TAKEN FOR A NON-DETERMINISTIC NCES MODEL.

No of places where non determinism exists	No of States generated before an error was detected	Time taken to generate the reachability graph
0	532	4 seconds
1	3552	60 seconds
2	5268	90 seconds

In our case both the deterministic and non-deterministic (controlled) NCES models on a system with 1) Intel dual core CPUD525@1.8 GHz, with 2GB of RAM, 2) Windows 7 operating system, 3) ViVe 0.37b version – for NCES model, the state space of the deterministic NCES model was less than 600 states and all the CTL properties were verified to pass. The verification in ViVe/SESA [31, 32] took less than 4 seconds. As seen from Table 2 even with non-determinism the time taken to verify the model with NCES is much lesser. Surprisingly, we also managed to spot the deadlock behaviour of the plant, hence concluding that the controller was still not yet up to the mark. In our approach we assemble the model from modules provided by IMC's without the need to take care of synchronization. Our tool chain helped to "plant" non-determinism modelling failures in particular mechatronic parts of the plant without causing state explosion. Such ability greatly enhances the performance of the developer, making formal verification a practical tool for everyday work. In our experiment we planted non-determinism in 1 or 2 places only in order to show the working of our framework.

VI. CONCLUSION

One of the main issues with formal verification is design and development of formal models and analysis of the counter examples, which are usually in a format that a control engineer would not understand. It is also time consuming, needs some level of understanding and experience in formal modelling. This paper has given a brief overview of previous research works to address this issue and presented a 4 step framework for formal verification of IEC 61499 function block systems with focus on counter example guided verification to compliment and add on top of the other research works cited to address the issue even better. The paper shows how we can make use of simulation (visualization) model to benefit and ease the process of model checking. We also used automated techniques for formal model generation, but added controlled non-determinism to better model the physical behaviour of the plant and exemplified how a counter example can be played back in the simulation to better understand the failures during model checking. Possible future work directions will concern, for example:

- Developing a more realistic timed plant modelling pattern allowing for variable actions duration within a certain interval.

- Integrating this approach with various simulation environment vendors such as CIROS by Festo and with IEC 61499 tool vendors such as 4DIAC and NxtStudio. Apply to real applications such as load balancing in smart grids [33].
- Further investigating the computational impact of the "planted" and incremental non-determinism methodology over range of other embedded control systems.
- Time-synchronisation proposed in [34, 35] can be applied for alignment of simulation models interpretation with player. However, formal model of the corresponding function block semantics is yet to be developed.

VII. ACKNOWLEDGEMENTS

This work was supported, in part, by the grant 381940 of Luleå University of Technology, and SAUNA project of the The Finnish Research Programme on Nuclear Power Plant Safety 2015 – 2018 (SAFIR2018) program.

REFERENCES

[1] R. Drechsler, *Advanced Formal Verification*. Norwell, MA, USA: Kluwer Academic Publishers, 2004.
[2] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge: The MIT Press, 1999.
[3] L. Fix, "Fifteen Years of Formal Property Verification in Intel," in *25 Years of Model Checking*, G. Orna and V. Helmut, Eds., ed: Springer-Verlag, 2008, pp. 139-144.
[4] C. Kern and M. R. Greenstreet, "Formal verification in hardware design: a survey," *ACM Trans. Des. Autom. Electron. Syst.,* vol. 4, pp. 123-193, 1999.
[5] "Programmable Logic Controllers — Part 3: Programming Languages, IEC Standard 61131-3," Third ed, 2013.
[6] V. Vyatkin, "IEC 61499 as Enabler of Distributed and Intelligent Automation: State-of-the-Art Review," *IEEE Transactions on Industrial Informatics,* vol. 7, pp. 768-781, 2011.
[7] S. Patil, J. Yan, V. Vyatkin, and C. Pang, "On composition of mechatronic components enabled by interoperability and portability provisions of IEC 61499: A case study," in *Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on*, 2013, pp. 1-4.
[8] V. Vyatkin, S. Karras, and T. Pfeiffer, "Architecture for automation system development based on IEC 61499 standard," in *Industrial Informatics, 2005. INDIN '05. 2005 3rd IEEE International Conference on*, 2005, pp. 13-18.
[9] C. Maffezzoni, L. L. Ferrarini, and E. Carpanzano, "Object-oriented models for advanced automation engineering - modular modeling in an object oriented database," *Control Engineering Practice,* vol. 7, pp. 957-968, 1999.
[10] C. Pang and V. Vyatkin, "Systematic Closed-Loop Modelling in IEC 61499 Function Blocks: A Case Study," in *13th IFAC Symposium on Information Control Problems in Manufacturing (INCOM 2009)*, Moscow, Russia, 2009, pp. 199-204.
[11] M. Sorouri, S. Patil, V. Vyatkin, and Z. Salcic, "Software Composition and Distributed Operation Scheduling in Modular Automated Machines," *Industrial Informatics, IEEE Transactions on,* vol. PP, pp. 1-1, 2015.
[12] C. Pang and V. Vyatkin, "IEC 61499 Function Block Implementation of Intelligent Mechatronic Component," in *8th IEEE Conference on Industrial Informatics (INDIN 2010)*, Osaka, Japan, 2010, pp. 1124-1129.
[13] nxtControl. (2012). *nxtSTUDIO*. Available: www.nxtcontrol.com
[14] *ICS Triplex ISaGRAF. ISaGRAF Workbench*. Available: http://www.isagraf.com/
[15] *FBDK – Function Block Development Kit*. Available: www.holobloc.com
[16] K. H. Hall, R. J. Staron, and A. Zoitl, "Challenges to Industry Adoption of the IEC 61499 Standard on Event-based Function Blocks," presented at the 5th IEEE International Conference on Industrial Informatics, 2007.
[17] M. Rausch and H. M. Hanisch, "Net condition/event systems with multiple condition outputs," in *Emerging Technologies and Factory Automation, 1995. ETFA '95, Proceedings., 1995 INRIA/IEEE Symposium on*, 1995, pp. 592-600 vol.1.
[18] V. Vyatkin, P. Starke, and H.-M. Hanisch. (1999-2002). *ViVe and SESA Model Checkers*. Available: http://www.ece.auckland.ac.nz/~vyatkin/tools/modelchekers.html
[19] T. Amnell, G. Behrmann, J. Bengtsson, P. R. D'Argenio, A. David, A. Fehnker, *et al.*, "UPPAAL - Now, Next, and Future," presented at the Proceedings of the 4th Summer School on Modeling and Verification of Parallel Processes, 2001.
[20] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, *et al.*, "NuSMV 2: An OpenSource Tool for Symbolic Model Checking," in *Computer Aided Verification*. vol. 2404, E. Brinksma and K. Larsen, Eds., ed: Springer Berlin / Heidelberg, 2002, pp. 241-268.
[21] C. Gerber, *Implementation and Verification of Distributed Control Systems* vol. 7: Logos Verlag Berlin GmbH, 2011.
[22] V. Vyatkin and H. M. Hanisch, "A modeling approach for verification of IEC1499 function blocks using net condition/event systems," in *Emerging Technologies and Factory Automation, 1999. Proceedings. ETFA '99. 1999 7th IEEE International Conference on*, 1999, pp. 261-270 vol.1.
[23] S. Patil, V. Dubinin, C. Pang, and V. Vyatkin, "Neutralizing Semantic Ambiguities of Function Block Architecture by Modeling with ASM," in *Perspectives of System Informatics*. vol. 8974, A. Voronkov and I. Virbitskaite, Eds., ed: Springer Berlin Heidelberg, 2015, pp. 76-91.
[24] H.-M. Hanisch, M. Hirsch, D. Missal, S. Preuße, and C. Gerber, "One Decade of IEC 61499 Modeling and Verification-Results and Open Issues," in *13th IFAC Symposium on Information Control Problems in Manufacturing*, V.A. Trapeznikov Institute of Control Sciences, Russia, 2009.
[25] V. Vyatkin and H. M. Hanisch, "Formal modeling and verification in the software engineering framework of IEC 61499: a way to self-verifying systems," in *Emerging Technologies and Factory Automation, 2001. Proceedings. 2001 8th IEEE International Conference on*, 2001, pp. 113-118 vol.2.
[26] C. Gerber, I. Ivanova-Vasileva, and H.-M. Hanisch, "A Data processing Model of IEC 61499 Function Blocks with Integer-Valued Data Types," in *Workshop on Intelligent Manufacturing Systems (IMS)*, 2008, pp. 239-244.
[27] S. Preusse, *Technologies for Engineering Manufacturing Systems Control in Closed Loop* vol. 10: Logos Verlag Berlin GmbH, 2013.
[28] S. Patil, S. Bhadra, and V. Vyatkin, "Closed-loop formal verification framework with non-determinism, configurable by meta-modelling," in *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*, 2011, pp. 3770-3775.
[29] C. Yang, V. Vyatkin, and C. Pang, "Model-Driven Development of Control Software for Distributed Automation: A Survey and an Approach," *IEEE Transactions on Systems, Man, and Cybernetics: Systems,* vol. 44, pp. 292-305, 2014.
[30] M. Sorouri, S. Patil, and V. Vyatkin, "Plug-and-Play IEC 61499 function blocks for distributed control design of Intelligent Mechatronic Systems," The University of Auckland, Auckland, Submitted for INDIN Conference 2012.
[31] S. Patil, V. Vyatkin, and M. Sorouri, "Formal verification of Intelligent Mechatronic Systems with decentralized control logic," in *Emerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conference on*, 2012, pp. 1-7.
[32] V. Vyatkin, H.-M. Hanisch, C. Pang, and C.-H. Yang, "Closed-loop modeling in future automation system engineering and validation," *IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews,* vol. 39, pp. 17-28, 2009.
[33] S. Patil, V. Vyatkin, and B. McMillin, "Implementation of FREEDM Smart Grid distributed load balancing using IEC 61499 function blocks," in *Industrial Electronics Society, IECON 2013 - 39th Annual Conference of the IEEE*, 2013, pp. 8154-8159.
[34] C. Pang, J. Yan, V. Vyatkin, and S. Jennings, "Distributed IEC 61499 material handling control based on time synchronization with IEEE 1588," in *Precision Clock Synchronization for Measurement Control and Communication (ISPCS), 2011 International IEEE Symposium on*, 2011, pp. 126-131.
[35] C. Pang, J. Yan, and V. Vyatkin, "Time-Complemented Event-Driven Architecture for Distributed Cyber-Physical Systems," *IEEE Transactions on Systems, Man and Cybernetics: Systems,* 2014.