Formal Validation of Downtimeless System Evolution in Embedded Automation Controllers

Christoph Suender, Valeriy Vyatkin and Alois Zoitl

***Abstract*—In this paper we report on a new formal approach to validation of on-the-fly modification of control software in automation systems. The concept of downtimeless system evolution (DSE) is introduced. The DSE is essentially based on the use of IEC 61499 system architecture and formal modeling and verification of the hardware and software of an automation device. The validation is performed by means of two complimentary techniques: analytic calculations and formal verification by model checking.**

***Index Terms*—Automation and control systems, Dynamic reconfiguration, Verification and Validation, Manufacturing automation, Model checking**

I. Introduction

Many industrial automation systems require operation without a single stop for weeks or even months. At the same time, it may be necessary to make modifications of the control program in order to achieve higher performance by applying more efficient control, or to add new machines into the manufacturing process, etc. In such cases, update of the control application cannot be done in the usual way by stopping and restarting the controller, instead a more sophisticated 'on the fly' update is required while the plant is running, being driven by the program which undergoes modifications. Some real-life scenarios, for example presented in [1], include (but not limited to) wind turbine control, or rolling mill control in steel making. In general, the ability to do 'on the fly' control reconfiguration can be regarded as one of the major enablers of flexibility and reconfigurability in manufacturing. Needless to say, the modification needs to be done in a way not affecting essential parameters of processes in the plant. For example, in steelmaking such a parameter can be the thickness of the steel sheet.

Some advanced programmable logic controllers (PLC) support such 'on the fly' program modification, but this feature comes with many restrictions. First, as indicated in many research results, the PLCs themselves do not fit to many requirements of flexible reconfigurable manufacturing. Secondly, the PLCs are good for local central control, but not for distributed control with decentralized logic.

For several years the authors have been involved in a research activity towards development of a more general solution for downtimeless change of control logic in automation systems. The core part of the developed εCEDAC solution [2] is the use of new IEC 61499 programming architecture [3], supported by novel software tools and runtime environments. In the course of εCEDAC project the term "Downtimeless System Evolution (DSE)" was coined, whose meaning in the automation control systems context is explained as follows:

- ***Downtimeless:*** Changes have to be applied to a running system with minimal disturbance to the process under control.
- ***System:*** Although software is considered to be the central element that is under change, change in hardware also may be taken into account, so we can talk about evolution of a system as a whole.
- ***Evolution:*** This term reflects the continuous and gradual nature of the changes that are required during a system's life-cycle.

The goal of the DSE validation is to make sure that the system under control works correctly during and after DSE is applied. This paper presents a novel solution for DSE evaluation based on comprehensive formal modeling of the control device with subsequent formal verification.

The paper is structured as follows. Section II presents the broader context of the problem and related works in the areas of software engineering, industrial automation and formal methods. Section III gives a short overview on the prerequisites for this work. The methodology for evaluation of DSE is introduced in section IV, followed by the discussion on appropriate evaluation means in section V. Section VI provides a description of modeling DSE in the formal language of Net Condition/Event Systems (NCES) [19]. A simple example will be used to demonstrate the practical application of the evaluation framework in section VII. The paper will be concluded with a summary of open issues (section VIII).

II. Broader Context and Related Works

In the last 30 years several studies concerning the behavior of software over its life-cycle have resulted in the so-called laws of software evolution, which are discussed with respect to component-based software engineering by Lehmann and Ramil [4]. According to these studies it can be stated that a program has to be continually adapted in order to satisfy the requirements of the user over its lifetime. Although the idea of evolution was initially conceived to support such maintenance steps as an update of a program to a newer version, it is also

Christoph Suender is with Thales Austria, Vienna, Austria, christoph.suender@thalesgroup.com
Valeriy Vyatkin is with the University of Auckland, New Zealand v.vyatkin@auckland.ac.nz
Alois Zoitl is with Technical University of Vienna, Austria, Zoitl@acin.tuwien.ac.at

applicable to software engineering in a broader sense. The need for software systems that run continuously without downtime applies to both business software and control applications used in automation and control systems (ACS). The challenges for software evolution have been summarized in Mens et al. [5] with the main statement that "the only way to overcome or avoid the negative effects of software aging is by placing change and evolution in the center of the software development process". Some challenges, according to [5], are:

- ***Evolution as a language construct:*** "Programming (or modeling) languages should provide more direct and explicit support for software evolution."
- ***Post-deployment runtime environment*****:** "There is an urgent need for proper support of runtime adaptations of systems while they are running, without the need to pause them, or even to shut them down."
- **Formal support for evolution:** "In order to become accepted as practical tools for software developers, formal methods need to embrace change and evolution as an essential fact of life".

This paper addresses these challenges specifically in ACS by proposing rigorous techniques for evaluation of the impact of evolution upon the system during DSE. The following works have essentially influenced our research:

Kramer and Magee [6] used the term *dynamic configuration* for "the ability to modify and extend a system while it is running". Their model for dynamic reconfiguration is based on a configuration manager, which is capable of translating requests for configuration changes expressed in the CONIC configuration language into commands to the operating system.

Walsh et al. [7] investigated a conceptual framework, which systematically and consistently addresses problems and solutions related to *dynamic reconfiguration*. The action of dynamic reconfiguration is categorized into different change types. For each change type the management of integrity is considered. In that work, this is used for the design of a system capable to provide dynamic reconfiguration by building a domain model which also includes fault tolerance modes.

The IEC 61499 standard [3] defines a reference system architecture for the next generation of distributed embedded control systems. This open architecture provides several enabling technologies improving flexibility and reconfigurability of industrial control systems, such as: component-based design using *function blocks* (FBs), configurability, interoperability and portability. With the growing support of commercial tools and platforms (e.g. ISaGRAF [8] and NxtControl [9]), this architecture makes a good progress towards becoming the major enabler of flexible automation solutions. There is great number of related research works. Thus, Brennan et al. [10] describe an enhanced model for IEC 61499 FBs that enables also the modeling of reconfiguration. The general idea is based on two different kinds of control paths within an IEC 61499 application: the execution path which is responsible for operating the normal control flow, and, orthogonally, a configuration control path that can be used for reconfiguration of the control application.

The related research on formal verification in the reconfiguration context is exemplified for instance by Tešanović et al. [11], who present a model-checking algorithm that is capable to verify properties of reconfigurable components. Their approach is based on aspect-oriented software development which modifies given components during the establishment of a system by applying certain aspects.

In several works related to reconfigurable manufacturing

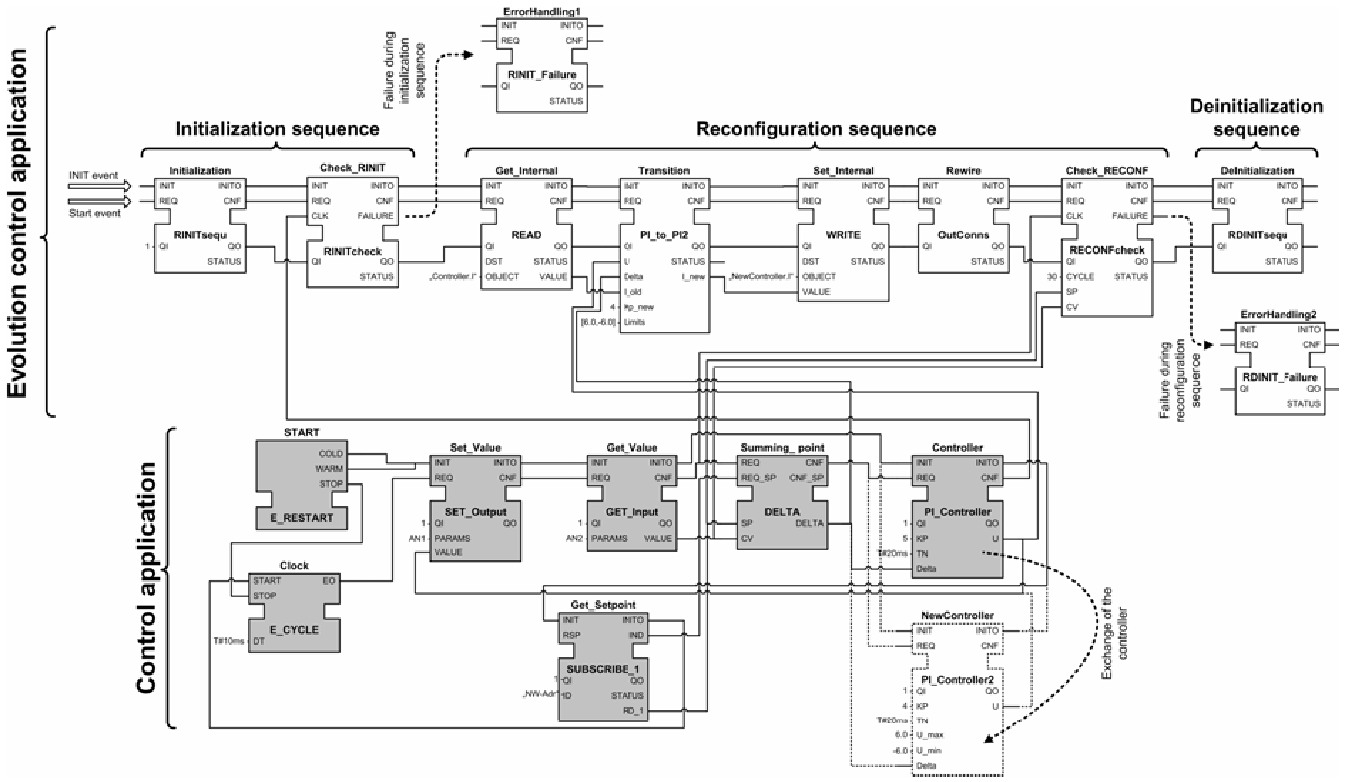

Figure 1. The idea of evolution control application (ECA).

systems, formal models were used in the design process in order to generate the control logic. Kalita and Khargonekar [12] present a methodology that combines both theorem proving and model checking based on timed transition models. Li et al. [13] aim at the design of reconfigurable logic controllers by rewriting Petri net based controllers. A similar approach with Petri net rewriting rules is given in Alcaraz-Mejía and López-Mellado [14]. The dynamic reconfiguration is expressed directly as model's rewriting. Park et al. [15] consider a controller capable to change within three pre-given modes. As a consequence, along with the formal model of different controllers and their control modes, also the mode-switching logic is included into the model of the system. This approach is based on Petri nets and automatic code generation from these models.

Although there are several developments aiming at dynamic reconfiguration, the existing works focus only on certain layers of software. No work is known to us that can model changes to the whole system and rigorously evaluate the evolution process. We are also not aware of any work towards practical application of the dynamic reconfiguration methods in the flexible automation context. The work, reported in this paper, aims at bridging this gap.

III. DSE Framework

The results of this work have been developed in εCEDAC project. Some preliminary ideas of the DSE engineering process and validation have been reported by Rooker et al. in [2]. DSE is based on the services provided by the runtime environment as described by Zoitl in [16].

A. Real-time reconfiguration runtime environment

DSE sets two important requirements to the runtime environments of embedded controllers: support for *dynamic reconfiguration* and *execution with guaranteed real-time properties*. The real-time reconfiguration runtime environment (R^3E) [16] fulfills these requirements. R^3E is a fully functional IEC 61499 – compliant platform. In addition to the event-driven execution of FBs, R^3E supports the execution of FB applications with regard to real-time constraints by providing a real-time scheduling solution for IEC 61499. The chain of FB executions started through an event occurrence at one event source FB and ending in an event sink FB serves as the execution context that is mapped to tasks within the operating system. The real-time constraints can be applied to the event source FBs within the application.

Further, the R^3E provides enhanced capabilities for control logic reconfiguration during its execution. It supports standard management commands defined in IEC 61499, but enhances this set to provide a complete basis for dynamic reconfiguration. The enhancements are related, in particular, to the access to FB internal variables and execution control services for managed FBs. These commands are called basic reconfiguration services and are implemented as a library of FB types. A special kind of FB application—termed as an *evolution control application* (ECA)—is constructed from the instances of those FBs to implement the desired reconfiguration. In Figure 1, a sample ECA is changing one controller function block to another controller function block. The control application is located in the lower part of the Figure with gray shaded shapes of FBs, and the block to be substituted is of white color. The ECA is in the top part of the Figure. The logic of ECA execution is chain-like, upon completion of one step the corresponding FB emits an event, activating the next FB in the chain.

Since R^3E can guarantee real-time properties of executed FB application, it can guarantee the fulfillment of such constraints for the combination of the original control application and the ECA, providing the basis for downtimeless system evolution.

The ECA is custom made for a particular control application being reconfigured, but templates can be developed for some typical cases. Thus Guler et al. [17] provide an idea of such a template for transition management in the case of the substitution of components (e.g., a controller in a closed-loop circuit).

B. Execution of the Evolution Control Application

As shown in Figure 2, evolution starts with the upload of an ECA into the device. If new hardware components are necessary within the system evolution step, they have to be made available already in this step. This is followed by three core sequences for applying changes to the software within the system, which are modeled according to the rules listed below. Evolution is concluded by the deletion of the ECA from the devices and the removal of the hardware that is no longer required.

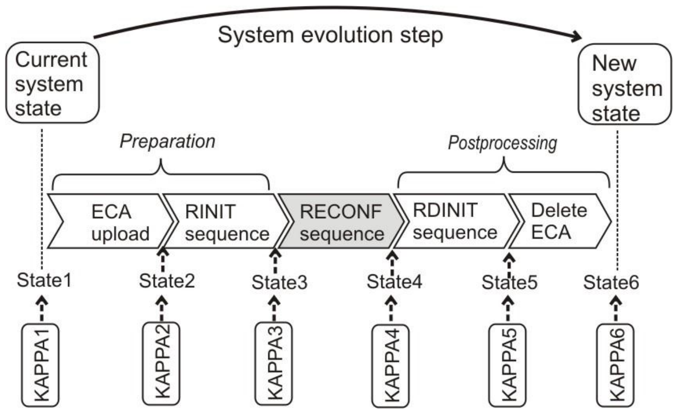

Figure 2. Execution phases of a system evolution step.

The steps in between apply changes to the control application based on the execution of the ECA:

Initialization sequence (RINIT) is the first sequence within the execution of the ECA. Typical actions within this sequence are the creation of new FBs and their input connections. No action within the RINIT sequence will affect the execution and behavior of the current control application. As a consequence these actions are not time critical and may be executed whenever a control device is not busy with executing control application's FBs.

Reconfiguration sequence (RECONF) follows the initialization and is responsible for making behavioral changes to the current control application. Based on the preparations (RINIT sequence) the current application can be modified at this stage to the new application. The actions within the reconfiguration sequence are *time critical*. In the case of FB substitution, the output connections have to be reconnected from old FBs to new FBs, and the internal states have to be properly set in the new FBs.

Deinitialization sequence (RDINIT) is responsible for bringing the system into a “clean” state. As the RECONF sequence is the critical one, no time should be spent at that stage for deletion of old FBs or connections. These elements can be deleted later within the RDINIT sequence. This sequence is not time critical as it does not influence the behavior of the control application.

C. Models for evaluation of ECA

The ECA formal evaluation technique developed in this work is based on two recent developments discussed as follows:

The authors have proposed in [23] a comprehensive classification of control devices that reflects their multilayer architecture and captures various characteristics, from properties of hardware to details of a particular control application. This allows representing each particular configuration as an array of parameters; each of which is associated with a numeric value. This array of parameters is referred to as *KAPPA vector*. Based on the classification, a universal analytic model of a control device was developed in [23] that can provide, for a given configuration and state, numeric estimations for such parameters as response time of the device, or schedulability bounds of real-time constrained function block chains [16]. The KAPPA vector is constant during the normal system operation (as it does not include internal variables of FBs), but may change after any change is applied to the ACS during system evolution, as shown in Figure 1. Stable states before and after DSE are characterized by the vectors KAPPA1 and KAPPA6 respectively. The vectors KAPPA2 to KAPPA5 correspond to the intermediate configurations achieved after the application of different reconfiguration command sequences of the system evolution step.

Another foundation of this work is the recent progress in comprehensive formal modeling and verification of control systems. New modeling languages and tools have enabled composition of formal models from pre-defined modules. Model-generators can create models automatically given source code of the controller. Powerful model-checking software tools can check the validity of complex temporal logic properties against such comprehensive models.

One such modular modeling language is Net Condition/Event Systems (NCES) (Rausch and Hanisch [19]). The dynamic behavior of modules is described as a Petri net extended with condition and event signals. The composition of such modules is achieved by interconnecting their event and condition interfaces. The composition’s result can also have an interface, so it can be used in other composite models. In this way complex models can be structured hierarchically. In order to analyze such a hierarchical model by model checking, it can be transformed into a flat model without modules. The VisualVerification (ViVe) toolset [26] performs the transformation to the flat model and model-checking followed by visual interpretation of its results.

Substantial experience has been obtained in NCES modeling of closed-loop automation systems, e.g. [21] presents details of the modeling of both plant and controller parts. Modeling of IEC 61499 function blocks was addressed in [22] and [25]. The most important features of NCES for DSE evaluation are:

Modularity: The model of a control device is a modular and hierarchical composition of modules modeling details of the plant and the control devices (including operating system, the runtime environment, the control program, and so on). Thus, model-checking can reveal how the changes, applied to the controller, effect the behavior of plant.

Control flow via events: Invocation of a code segment can be modeled by passing an event to the NCES transition, modeling the first command in the segment.

Timing: The model can capture timing properties of the controller commands, which also provides the possibility to correctly model preemption of tasks by the operating system (see [23] for details).

The NCES dialect used in this paper follows [20] and is characterized by such features as timed arcs and arcs with multiple tokens capacity, but no colored tokens. The approach to NCES modeling of ECA and runtime environment is based on the work [23] and [25]. Analytic evaluation and model-checking are complementary techniques, and our approach proposes using both, but for evaluation of different system evolution steps. This will be further discussed in section V.

IV. Methodology of DSE Evaluation

The goal of this section is to extract properties that will be checked at each step of DSE in order to evaluate its correctness. For this purpose, the basic reconfiguration services of the runtime environment R^{3}E will be evaluated in terms of the reference architecture for dynamic reconfiguration [7] by Walsh et al., where a general model of changes to a component-organized software system was proposed, including system integrity characteristics.

We will apply the ideas of [7] in the context of IEC 61499, regarding function blocks as software components, although a more specific treatment may be required for Service Interface Function Blocks (SIFBs) and composite FBs (for more detailed analysis the interested reader is referred to [18]).

A. Changes of FB applications supported by R^{3}E

The following general types of changes from [7] are supported by the R^{3}E run-time environment and taken into account in our study:

- ***Protocol change*** refers to communication protocol used to interact between parts of a distributed FB application. This type of change can be achieved by re-directing connections between existing FBs (to different communication FBs (CFB)) or by assigning new parameters to some CFBs. The software components (FB instances) themselves may not need to be changed.
- ***Application topology change*** is achieved by modifications to the software components, i.e. the FB instances. Two kinds of changes can be applied: (1) the substitution of an

FB type without changing its interface, and (2)the relocation of application parts from one device to another.

- ***Architectural change*** can be defined as the combination of the protocol and topology changes, covering most of changes to an application within the ACS.
- ***Internal change:*** R^{3}E supports access to internal variables of an FB. This way internal change of a component can be achieved, that can result in changing behavior of the FB instance. The scope of supported internal changes is limited to the values of the FB state vector. No basic reconfiguration services exist in order to change the type of an FB (for that a new FB type needs to be created). Although FB type substitution is not possible by using the basic reconfiguration services as a single command, it can be implemented within an ECA by a combination of topology, protocol and internal changes.

B. System integrity characteristics for DSE

During the evolution of a system, system integrity characteristics must be preserved. These characteristics can be derived taking into account the type of change and properties of the particular application, including dynamic properties of the controlled process and characteristics of the control device. The following list describes the different system integrity characteristics, which had to be enhanced in comparison to the reference architecture [7]. The enhancements, in particular, deal with composite FBs and Service Interface FBs.

Global and local consistency: In terms of DSE, global consistency preserves the specifications of the control application and the process under control. These specifications are split up into plant, process, and product specifications. Each of these categories may have top-level global integrity characteristics that may be split up into local aspects that are mentioned within the local consistency characteristic.

Presence of ***active references*** between components, especially SIFBs, may impact on system integrity. A SIFB can encapsulate any kind of service which can include also dependencies on other SIFBs (this is typically the case in communication SIFBs). Changes applied to such an active reference may produce failures in the behavior of the control application. On the one hand such a dependency may be violated in the new system state and therefore has to be detected during the evaluation of the new application. On the other hand the dependency may be violated during the system evolution step temporarily (e.g., due to a disorder of basic reconfiguration services) which has to be proved by the evaluation of DSE.

State management: Although no basic reconfiguration service exists in order to exchange an FB instance it is possible to implement such an exchange by a sequence of commands. The state management requirement can be represented as a property within the evolution specification and proved with respect to both FB exchange and substitution of the entire FB network.

Dependent DSE operations: The DSE actions need to be applied in a proper order, for that the execution order needs to be established. The order can be influenced by both event and data flow interrelations between the function blocks being the subject of the evolution. In contrast to the “active reference” property, which aims at internal dependencies within SIFBs, the dependent DSE operations refers to the dependencies based on the execution of basic reconfiguration services.

Real-time constrained operation is a very important requirement also within DSE. Usually it originates in the need to achieve certain quality of control characteristics. As the ECA reconfigures the control application, the changes to the control application may be subject to certain real-time constraints. This occurs during the RECONF sequence when preserving the global and local consistency. Real-time constrained operation is a part of these integrity characteristics because any execution phase of a system evolution step can influence the execution of the control application due to the sharing of computational resources.

Requirements of resources: The most important resources for DSE are:

- memory required to store the ECA;
- memory for intermediate results of the control application.
- computational performance of the control device
- availability of required FB types in the device library.

Requirements of resources are associated with architectural change and global and local consistency through the specifications of global and local properties.

V. DSE Evaluation

The evaluation of DSE has to prove that the DSE complies with the evolution specification by not violating any properties of the plant, process, or product specification (global and local consistency), and the other system integrity characteristics. Verification by model checking is applied only for the reconfiguration sequence. All other sequences can be evaluated by using appropriate analytic methods determining the quantitative effect of the ECA on the control application. The reconfiguration sequence represents the most important phase during the execution of a system evolution step. In this phase, the control application undergoes modifications while running, which implies the time critical execution of the basic reconfiguration services and calculations included therein. This situation requires investigation on the effects between the control application and the ECA, which will be performed by applying the model checking technique. The obvious difficulty is that the model needs to be changed during the model checking process. To the best of the authors' knowledge, no model-checking tool can do this directly, so the developed solution is to include elements of self-modification in the basic models of the FB language constructs. The used models will be described in detail in section VI.

The characteristics discussed in section IV translate to the following properties that need to be checked for the RECONF sequence:

- ***Global and local consistency check*** ensures that the properties of the plant, process and the product specifications hold during the reconfiguration sequence.
- ***Active references*** consider the interrelation of different control application parts due to underlying services

encapsulated in SIFBs. Specifically for RECONF the temporal interruption of references needs to be considered and avoided. In case of composite FBs the included component FBs have to be considered, which may again be SIFBs that encapsulate underlying services. The proof of this property within the evolution specification requires a detailed formal modeling of the underlying services.

- ***State management*** is added as a special property of the evolution specification and is also checked by model checking, because the target of DSE is minimization of such disturbances. The behavior of the control application (global and local consistency) is directly influenced by the transition management method that is used. Depending on the context, if the transition management fails, the disturbances to the process may be tolerated by the plant, process, and product specifications.
- ***Real-time constrained operation*** means checking compliance of the timing requirements with respect to the control application during this evolution phase. This is as important as ensuring the functional properties. To achieve this, the corresponding model needs to include information about duration of actions in the control application and in the ECA.

VI. NCES MODELING OF DYNAMIC RECONFIGURATION

The changes applied to the control application in DSE are restricted to the creation and removal of connections between FBs and setting values of parameters such as, for example, internal variables. This allows building the formal NCES model of DSE as a composition of models of a limited number of basic reconfiguration services.

The prerequisite to DSE evaluation by formal verification is availability of comprehensive formal models of control devices. We follow here the modeling approach of [23], where a range of NCES models for all elements of IEC 61499 compliant devices was developed: from the details of hardware to the models of function blocks. In addition, a formal model of a control device must cover the support of basic reconfiguration services. Based on a given specification and a given model the state space of the system will be explored by model checking. Within this process, the model does not need to be changed.

In the following subsections, we will exemplify the ideas of NCES modeling of four different classes of change to the system model. These classes cover the required set of basic reconfiguration services within RECONF. The implemented changes are basic in the sense that they do not include structural changes to the system model.

A. Manipulation of connections

Event and data connections between FBs need to be treated differently. According to the IEC 61499 standard, an event connection is used to trigger the execution of FBs. Based on the execution semantics of R^3E the issuing of an output event means that the input events at the arc's destination will be put into the queue by the event dispatcher. The corresponding NCES module 'ManagedEventConnection' is depicted in Figure 3.

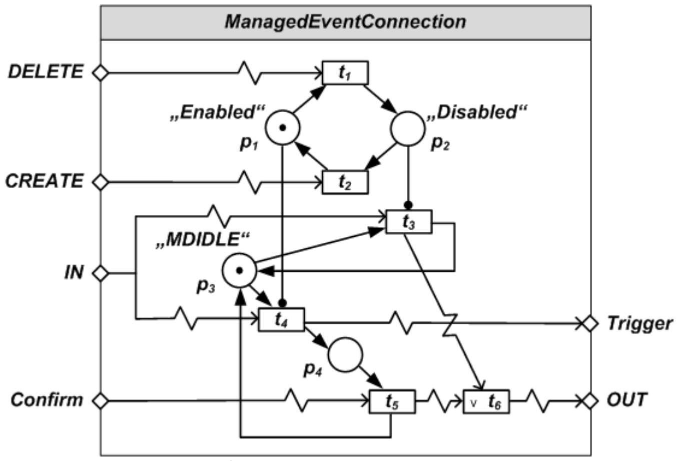

Figure 3. NCES model of a managed event connection.

The model incorporates the creation and deletion of the event connection. The input event 'IN' receives an event if the connection source FB emits the corresponding output event. Based on the internal state of the event connection (represented by places 'p_1' and 'p_2'), two different paths are available. If the event connection is enabled (i.e., it has been created), the output event 'Trigger' will be issued, which is used to put the corresponding input event into the event dispatcher. After a confirmation via the event input 'Confirm' the output event 'OUT' is triggered and the execution flow within the connection source FB will be continued. But if the event connection is disabled (i.e., it has been deleted), nothing else will happen except the output event 'OUT' is triggered. In terms of IEC 61499 this means that the event connection does not exist, because no corresponding entry exists within the event dispatcher.

The creation and deletion of the event connection is triggered by the input events 'CREATE' and 'DELETE', which are issued by the basic reconfiguration services within the ECA. The model in Figure 3 shows a model of an event connection that is initially created (the model of an initially disconnected event connection will be achieved by changing the initial marking from 'p_1' to 'p_2').

The behavior of the data connection model is quite different. In the R^3E implementation a data connection includes a storage element. As soon as an output event occurs which is associated with the data output via the WITH construct, the storage element of the data connection model is assigned to the data output value of the FB. If several data connections exist with the same source (several connections from the same data output to several data inputs) only one storage element will be used for all these data connections. Despite the mentioned differences, the NCES model of a data connection is quite similar to the event connections model, so it is omitted for the sake of brevity.

B. Execution control of FB instances

A simplified model of a managed FB instance, shown in Figure 4, reacts on the START and STOP management commands that can change state of the FB instance from 'IDLE' to 'RUNNING' and 'STOPPED' respectively. An input event may trigger the operation of the FB, upon which this model will receive ‘IN’ event.

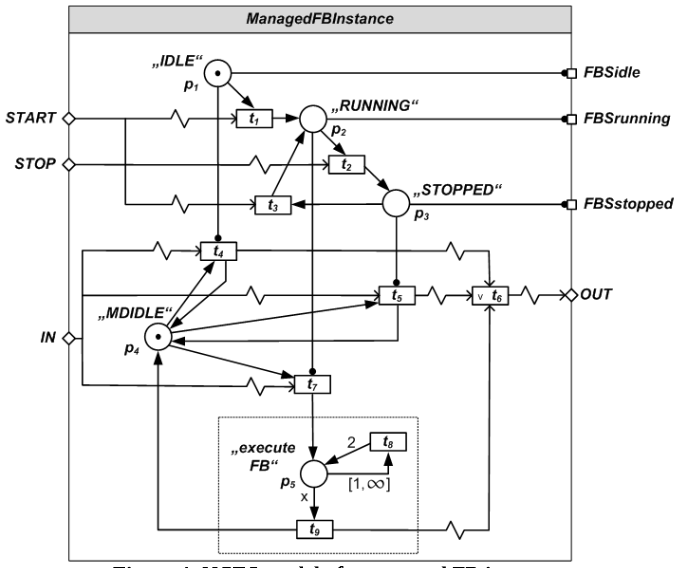

Figure 4: NCES model of a managed FB instance.

If the FB is in the ’IDLE‘ or ’STOPPED‘ state, the invocation will be ignored, resulting in immediate issuance of the ‘OUT’ event. This will be passed to the scheduler in order to let another FB to run. Otherwise, if the FB is in the ’RUNNING‘ state, the state ’execute FB‘ will last for a number of time units proportional to the duration of the FB’s internal logic execution. For a concrete FB instance, the NCES model of this FB has to be used instead of the simple time delay used in Figure 5. The current state of the FB instance is available outside of the module via the condition outputs ‘FBSidle’, ‘FBSrunning’, and ‘FBSstopped’.

Such management commands as READ and WRITE can be modeled without any additional effort in the NCES models. Any variable is represented by a set of places within the formal model. The model of the corresponding basic reconfiguration service within the ECA has to be connected to these places via condition arcs. As soon as the basic reconfiguration service is executed it will gather the current value of the variable.

C. Evolution control application

An ECA is modeled using the same concepts as used for modeling of control applications. The only difference is the use of special FB types implementing the basic reconfiguration services. The formal model of a basic reconfiguration service is similar to the model of any FB. The only difference is that a special interface is added, as specified by the IEC 61499 management commands that is incorporated in the basic reconfiguration service. The interfaces for the relevant management commands within the ECA are based on the formal description of the effects of the basic reconfiguration service described above. For a detailed

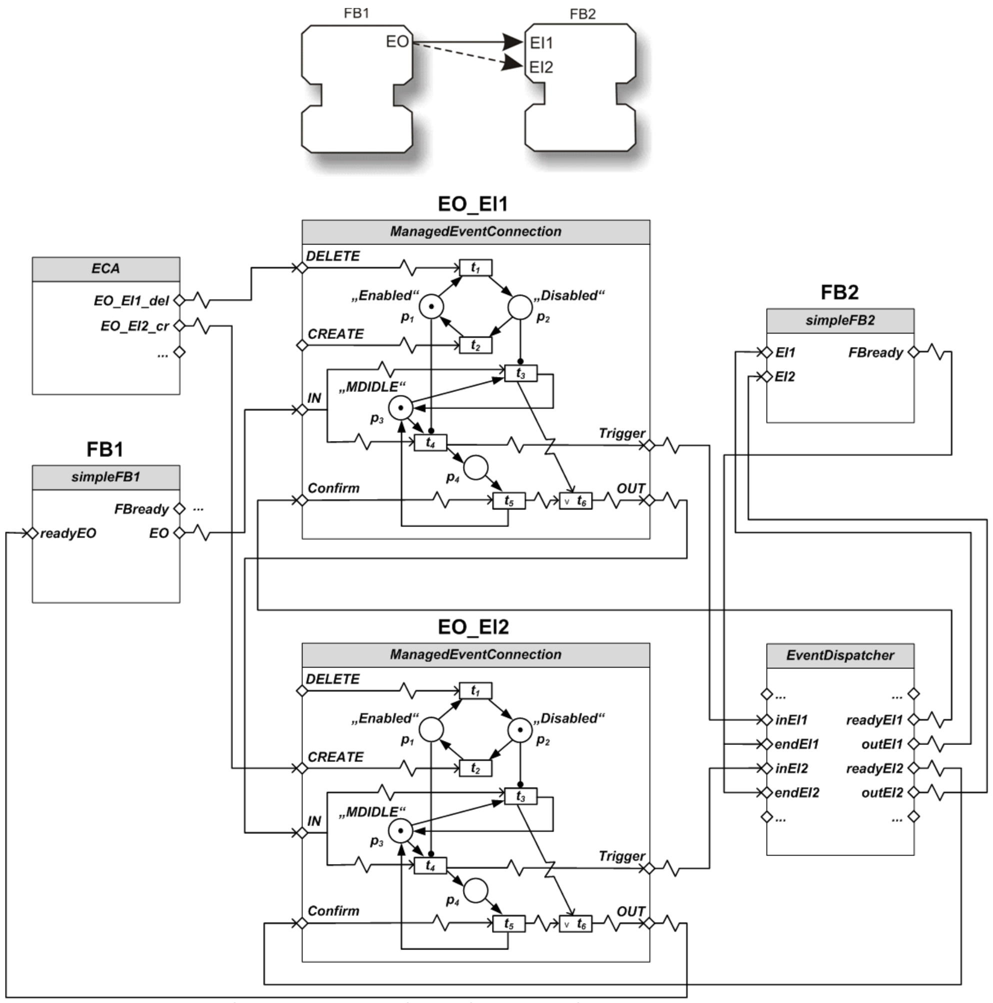

Figure 5. Simplified NCES model of reconfiguration of event connections between two FBs.

description the interested reading is referred to [18].

D. Example of use

The use of the models introduced above will be illustrated on a simple example of modeling of an event connection substitution between two function blocks in Figure . Each of the function blocks ‘FB1’ and ‘FB2’ is represented in the NCES model by an instance of the model types ‘simpleFB1’ and ‘simpleFB2’ (termed FB1 and FB2 respectively). Interfaces of these models are intentionally simplified to include only the elements necessary for this illustration. Both event connections are modeled as instances of NCES model type ’ManagedEventConnection‘ (termed ‘EO_EI1’ and ‘EO_EI2’ respectively). In addition, models of ECA and of the event dispatcher are also shown in the Figure.

The sending of an output event works in the manner already described in section V.A. by inserting triggers into the event dispatcher. But the ‘ManagedEventConnection’ model sends a request to the event dispatcher only if it is enabled. In the initial configuration for this simple example ‘EO_EI1’ is enabled and ‘EO_EI2’ is disabled. Therefore ‘FB2’ will be triggered by the input event ‘EI1’. The ECA controls the reconfiguration of the application and is able to switch ‘EO_EI1’ to “disabled” and ‘EO_EI2’ to “enabled” during the verification process done by model checking. After the emission of the two corresponding events, the ‘ECA’ has switched the event connection from ‘EI1’ to ‘EI2’.

One should note that this model is simplified in many aspects. Nevertheless, it gives impression of the modeling method. It is envisaged that such models can be automatically generated for a given FB application and known details of the control device (hardware, OS, run-time environment).

VII. Illustrative Example of DSE Evaluation

We will demonstrate DSE and its evaluation by exchanging function blocks during the execution of a simple FB application. For illustrative purposes it is simpler than the one in Figure 1, but similar verification experiments have been conducted with that and other control applications.

The application in Figure 6 triggers cyclically the ‘E_CTU’ FB, whose output value is added to an internal variable in ‘ADD_INT_TO_INTERNAL’. The application stops after this variable exceeds a given limit, detected by ‘CHECK_INT_GREATER’.

The DSE substitutes the addition FB for the subtraction FB ‘SUB_INT_FROM_INTERNAL’. After this change, the application needs to stop if the variable goes below the given limit (rather than above as in the original). Therefore also the ‘CHECK_INT_GREATER’ FB has to be exchanged by

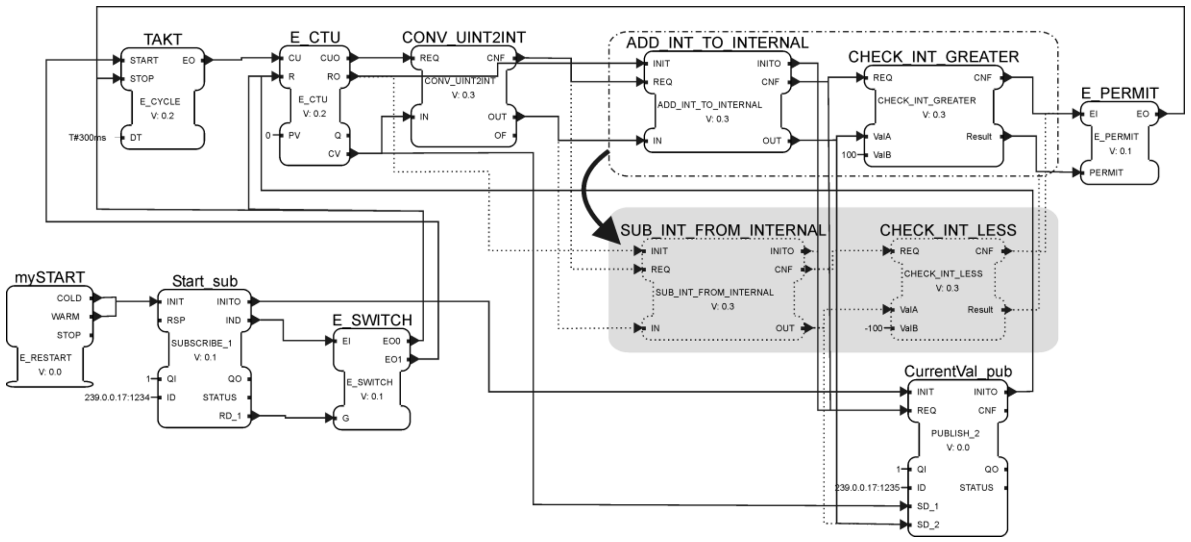

Figure 6. Practical example—mixed representation of current and new system state.

	CTL formula	Description of used model elements
1	**AG** (*p1251* → **AF** (*p1267*))	*p1251* … sending of output event ‘E_SWITCH.EO1’ *p1267* … sending of output event ‘TAKT.EO’
2	**AG** (*p1245* → **AF** (*p1254*))	*p1245* … sending of output event ‘E_SWITCH.EO0’ *p1254* … idle state of ‘TAKT’
3	**AG** (*p6436* → **AX** ((*p2421 = p3593*) ∧ … ∧ (*p2452 = p3624*))	*p6436* … sending of event ‘SET_FBINTVAR_INTERNAL.CNF’, *p2421* to *p2452* ... internal variables ‘ADD_INT_TO_INTERNAL’ *p3593* to *p3624* … internal variable ‘SUB_INT_FROM_INTENRAL’
4	**AG** (*p1267* ∧ ¬ *p3376* →**A** [(¬ *p1257* ∨ **AG** (¬ *p3376*)) **W** *p3376*])	*p1267* … starting point of application flow (sending of ‘TAKT.EO’) *p3376* … end point (triggering of ‘E_PERMIT.EI’) *p1257* … marks the triggering of ‘TAKT’ by the timer
5	**AG** (*p5962* → **5 EF**$_{[0, \alpha]}$(*p6565*))	*p5962* … starting point for the execution of the RECONF sequence *p6565* … end of execution of the RECONF sequence, *α* … end of allowed time frame, e.g. 100000 as equivalent to *100 ms* (*0,1 µs = 1* NCES time step)

Table 1: **Evolution specifications in terms of CTL formulas.**

‘CHECK_INT_LESS’ FB. A real-world motivation for such an example is, for instance, a closed-loop control circuit, in which the controller FB or a filter in the feedback loop can be substituted during the operation. In our example we omitted the model of plant (or a direct closed-loop connection to the plant) for the sake of simplicity. However, from the viewpoint of DSE all necessary elements are included. A detailed description of this example as well as its evaluation can be found in [18].

A. *Analytic evaluations*

The evaluation of system integrity characteristics based on the analytic calculations according to Table 1 is implemented as evaluation wizard in the 4DIAC IDE [24]. As our example is not real-time constrained, the corresponding properties for global and local consistency are not addressed.

The check for ***dependent DSE operation*** is split into two parts. Static dependency check is included in the modeling activities within the engineering tool. Dynamic dependency check is done by first identifying the execution order of the basic reconfiguration services. For each of these commands the runtime environment has to be able to execute this type of service and the parameters of the service need to be valid (e.g., the resource/FB which is addressed by the command). After each basic reconfiguration service within this execution order the virtual KAPPA vector has to be updated according to the effects of the command.

The ***requirements of resources*** belong to those properties, which may be changed during the execution of a system evolution step. In our case, these are FB types and memory consumption. Based on the representation of the necessary information within the KAPPA vector, the wizard is able to apply these checks for the different sequences of the system evolution step.

B. *Verification by model checking*

The developed set of properties, capturing the characteristics for the RECONF sequence, have been formalized in the Computation Tree Logic (CTL) language and checked against the NCES model. The total model combines the old and new FB applications, ECA, runtime environment and characteristics of the control device. The size of the NCES model, obtained as a result of flattening, is 6672 net places and 10563 net transitions. The reachability graph includes 11116 states when using the “Maximum set of spontaneous” firing rule of NCES. This is a very reasonable number, as ViVe model-checker is capable of dealing with reachability spaces of several millions of states. Verification is applied to each step of the verification sequence one-by-one, therefore model-checking performance does not seem to restrict scalability of this approach.

In this paper, we focus mainly on the applied checks, referring the reader to [18] and [23] for the details of the system model. The checked properties are discussed in detail as follows:

Global and local consistency: As our example is purely computational and does not interact with a plant, the specifications refer only to the model of the FB application, for instance as follows:

- “If the user interface sends a start command, the FB ‘TAKT’ has to send at least one output event some time afterwards.” This property may be formulated as the CTL formula as depicted in Table 1 (row 1).
- “If the user interface sends a stop command, the FB ‘TAKT’ has to be set to its idle state some time afterwards.” This property may be formulated in a similar manner as depicted in Table 1 (row 2).

Active references: This property is not applicable in our example because no references to underlying services (e.g. system timer) are included in the DSE.

State management: Within the control application the internal state of FB ‘ADD_INT_TO_INTERNAL’ has to be transferred to FB ‘SUB_INT_FROM_INTERNAL’ without any additional calculations (this can be a common scenario in the evolution of closed-loop control applications). In general, the transition management policy may be evaluated according to the effects on the plant. Since our control example does not use a model of the plant, the evaluation of state management is focused on the control application itself, expressed as the following specification:

“After the execution of the ECA the internal variables of the two FBs ‘ADD_INT_TO_INTERNAL’ and ‘SUB_INT_FROM_INTERNAL’ need to have the same value”.

This criterion has to be fulfilled exactly after the execution of the corresponding basic reconfiguration service, or, more precisely, right after finishing of the WRITE command. A possible formulation is given in Table 1 (row 3).

The evaluation result of this property is a very important measure for the quality of the whole system evolution step as it reflects the level of disturbances to the system under control. A more detailed discussion on the formulation of the state management property can be found in [27].

Real-time constrained operation: Possible specifications for the check of this property may be:

- “The execution of the calculations within the control application has to be finished before a new trigger occurs.” This property would have been a part of the global and local consistency properties if a real-time constrained execution had been modeled with the event source ‘TAKT.EO’. This property is formulated as depicted in Table 1 (row 4).
- “The execution of the RECONF sequence within the ECA has to happen within a given time frame.” This property is formulated as given in Table 1 (row 5).

For this example, the model-checking has confirmed the validity of all properties listed above.

VIII. Conclusion

This paper presents a new methodology for the evaluation of dynamic systems evolution. It was illustrated using a simple but practical example. The evaluation is the key element for the practical use of DSE, because the top-priority goal of DSE is to keep the system running. The application of dynamic reconfiguration can be useless if its correctness cannot be proved before starting the execution of DSE. Further research work should be focused on support during the engineering process: standardized description of devices and the used parameters or automatic establishment of formal models for the overall control device are required. Otherwise the ACS user will expend too much effort for the evaluation, what would be the main disadvantage of this approach. Automatic ECA creation can be based on this approach, too. A possible next step for enhancing the theory for DSE would be the provision of metrics for special application classes in order to provide a means to measure property coverage for a given evolution step.

References

[1] T. Baier, J. Fritsche, G. Keintzel, D. Loy, R. Schranz, H. Steininger, T. Strasser and C. Sünder, "Future scenarios for application of downtimeless reconfiguration in industrial practice", *IEEE Int. Conf. on Industrial Informatics (INDIN'07)*, Vienna, Austria, July, 2007, pp. 1129-1134

[2] M. N. Rooker, C. Sünder, T. Strasser, A. Zoitl, O. Hummer, G. Ebenhofer, "Zero Downtime Reconfiguration of Distributed Automation Systems: The εCEDAC Approach", in *Proc. 3rd Int. Conf. on Industrial Applications of Holonic and Multi-Agent Systems*, Regensburg (Germany), Sept. 2007, Springer Verlag Berlin Heidelberg (LNCS 4659), pp. 326-337

[3] IEC 61499-1 Function blocks—Part 1: Architecture, Int. Standard, International Electrotechnical Commission (IEC), 2005

[4] M. M. Lehmann and J. F. Ramil, "Software evolution in the age of component-based software engineering", *IEEE Proceedings Software*, vol. 147, nb. 6, Dec. 2000, ISSN 1462-5970

[5] T. Mens, M. Wermelinger, S. Ducasse, S. Demeyer, R. Hirschfeld, M. Jazayeri, "Challenges in Software Evolution", in *Proc. 8th IEEE Int. Workshop on Principles of Software Evolution*, Lisbon (Portugal), Sept. 2005, pp. 13-22

[6] J. Kramer, J. Magee, "Dynamic Configuration for Distributed Systems", *IEEE Trans. Software Engineering*, vol. SE-11, nb. 4, April 1985, pp. 424-436

[7] J. D. Walsh, F. Bordeleau, B. Selic, "Domain analysis of dynamic system reconfiguration", *Software and Systems Modeling*, vol. 6, no. 4, Springer Verlag, 2007, pp. 355-380, ISSN 1619-1366

[8] ICS Triplex. ISaGRAF Workbench for IEC 61499/ 61131, v.5.1 [Online]. Available: http://www.icstriplex.com/

[9] nxtControl GmbH, nxtControl - Next generation software for next generation customers [Online, 2009, June]. Available: http://www.nxtcontrol.com/

[10] R. W. Brennan, X. Zhang, Y. Xu, D. H. Norrie, "A reconfigurable concurrent function block model and its implementation in real-time Java", *Integrated Computer-Aided Engineering*, IOS Press, vol. 9, nb. 3, 2002, pp.263-279, ISSN 1069-2509

[11] A. Tešanović, S. Nadjm-Tehrani, J. Hansson, "Modular Verification of Reconfigurable Components", in *Component-Based Software Development for Embedded Systems*, LNCS 3778, C. Atkinson et al. (Eds.), Springer Verlag Berlin Heidelberg, 2005, pp. 59-81, ISBN 978-3-540-30644-3

[12] D. Kalita, P. P. Khargonekar, "Formal Verification for Analysis and Design of Logic Controllers for Reconfigurable Manufacturing Systems", *IEEE Trans. Robotics and Automation*, 2002, 18(4), pp. 463-474

[13] J. Li, X. Dai, Z. Meng, "Dynamic Reconfiguration of Petri Net Logic Controllers Based on Modified Net Rewriting Systems", in: *Proc. IEEE Int. Conf. on Mechatronics and Automation*, Niagara Falls (Canada), July 2005, pp. 592-567

[14] M. Alcaraz-Mejía, E. López-Mellado, "Petri Net Model Reconfiguration of Discrete Manufacturing Systems", in: *Proc. 12th IFAC Symp. on Information Control Problems in Manufacturing*, vol. 1, Saint-Etienne (France), May 2006, pp. 547-552

[15] E. Park, D. M. Tilbury, P. P. Khargonekar, "A Modeling and Analysis Methodology for Modular Logic Controllers of Machining Systems Using Petri Net Formalism", *IEEE Trans. Systems, Man, and Cybernetics—Part C: Applications and Reviews*, 2001, vol. 31, nb. 2, pp. 168-188

[16] A. Zoitl, *Real-time execution for IEC 61499*, ISA and O^{3}neida, USA, 2009, ISBN 978-1-934394-27-4

[17] M. Guler, S. Clements, L. M. Wills, B. S. Heck, G. J. Vachtsevanos, "Transition Management for Reconfigurable Hybrid Control Systems", *IEEE Control Systems Magazine*, vol. 23, nb. 1, Feb. 2003, pp. 36-49, ISSN 0272-1708

[18] C. Sünder, "Evaluation of Downtimeless System Evolution in Automation and Control Systems", Ph.D. dissertation, Automation and Control Institute, Vienna University of Technology, Vienna (Austria), 2008 (online available via http://aleph.ub.tuwien.ac.at/ALEPH)

[19] M. Rausch, H.-M. Hanisch, "Net Condition/Event Systems with Multiple Condition Outputs", in: *Proc. INRA/IEEE Symp. on Emerging Technologies and Factory Automation*, vol. 1, Magdeburg , 1995, pp. 592-600

[20] V. Vyatkin, G. Bouzon, "Using Visual Specifications in Verification of Industrial Automation Controllers," *EURASIP Journal on Embedded Systems*, vol. 2008, Article ID 251957, 2008, 9 p.

[21] H.-M. Hanisch, A. Lobov, J.L. Martinez Lastra, R. Tuokko, V. Vyatkin, "Formal Validation of Intelligent Automated Production Systems towards Industrial Applications", *Int. Journal of Manufacturing Technology and Management*, vol. 8, nb. 1/2/3, 2006, pp.75-106

[22] V. Vyatkin, H.-M. Hanisch, "Verification of Distributed Control Systems in Intelligent Manufacturing", *Journal of International Manufacturing*, vol. 14, nb. 1, 2003, pp. 123-136

[23] C. Sünder, V. Vyatkin, "Functional and temporal formal modeling of embedded controllers for intelligent mechatronic systems", *Intl. J. Mechatronics and Manufacturing Systems*, Vol. 2, Nb. 1/2, 2009, pp.215-235

[24] 4DIAC IDE, Online, http://www.fordiac.org/

[25] C. Pang ,V. Vyatkin, "Formal modelling of IEC61499 systems following the Sequential Hypothesis", *5th IEEE Int. Conf. on Industrial Informatics* (*INDIN'07)*, Vienna, Austria, July 2007, pp.879-884

[26] V. Vyatkin, Visual Verification Framework, Version 0.35a, Online: http://www.ece.auckland.ac.nz/~vyatkin/vive/ViVe.zip , June 2009

[27] T. Kovácsházy, G. Péceli, G. Simon, "Transients in Reconfigurable Signal Processing Channels", *IEEE Transactions on Instrumentation and Measurements*, vol. 50, nb. 4, pp. 936-940