Control Engineering Practice ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at SciVerse ScienceDirect

Control Engineering Practice

journal homepage: www.elsevier.com/locate/conengprac

Transformation of Simulink models to IEC 61499 Function Blocks for verification of distributed control systems

Chia-han Yang*, Valeriy Vyatkin

University of Auckland, Electrical and Computer Engineering, Level 2, Science Centre, 38 Princes Street, Auckland 1041, New Zealand

ARTICLE INFO

Article history:
Received 17 May 2010
Accepted 20 June 2012

Keywords:
Simulation
Distributed systems
IEC 61499
MATLAB Simulink
Function Blocks

ABSTRACT

In this paper, a new model-based engineering approach is introduced by bridging MATLAB Simulink with IEC61499 Function Block models. This is achieved by a transformation between the two block-diagram languages. The transformation supported by the developed tools sets the cornerstone of the verification and validation framework for IEC 61499 Function Blocks in closed-loop with the models of the plant. The framework also paves the way to running distributed simulations of complex hybrid (i.e., continuous-discrete) closed-loop plant-controller systems and building complex models using the efficient object instantiation techniques of IEC 61499.

1. Introduction

The new market demands for flexibility and reconfigurability in manufacturing and process industries motivate the transition from centralised automation systems to the systems with distributed intelligence (Chokshi & Mcfarlane, 2008; Felcht, Darton, Prince, & Wood, 2003; Yang & Vyatkin, 2008). This approach relies on decentralised control architecture which includes multiple control units and each of which controlling a device or a sub-section of the entire system. These controllers may communicate and collaborate with each other through some communication channels, (i.e., Ethernet, Fieldbus, etc.). Although it is possible to implement distributed intelligence using traditional Programmable Logic Controllers (PLC) connected via some communication channels, however, software development for distributed systems using the PLC-oriented centralized software paradigm is cumbersome for a number of reasons, as discussed, for example in Vyatkin (2007a).

Fig. 1 To address these issues, the International Electrotechnical Commission (IEC) in the IEC61499 standard (2005) has defined a new reference software architecture for designing distributed control and automation systems. The standard continues using the event-driven module "Function Blocks" (FBs), which originates from IEC 61131-3 standard. The event-driven execution of components provides for transparent design of distributed systems. It can improve the flexibility and efficiency of system design on account of solutions' re-use. Function Blocks offer a higher level of abstraction than the artefacts of the PLC based software architecture. The modularity concept is the key for easier reconfiguration and software re-usability which are not easily achievable in the traditional IEC 61131-3 compliant PLCs, where re-writing or re-structuring the program code is often needed even in a simple scenario (Vyatkin, 2007b). Even though the modularity concept with FBs is already introduced in IEC 61131-3, the new standard offers completely different design architecture, and it can provide much easier process in reconfiguration, maintenance and deployment process of distributed systems. This new standard sets the common criterion in the implementation of distributed control, allowing the design to be vendor-independent while achieving flexibility in terms of both software and hardware.

However, distributed intelligence systems are difficult to validate and verify (Kshemkalyani & Singhal, 2008) in order to ensure their correctness and robustness. The testing complications originate in the fundamental features of distributed systems, such as lack of global clocks, impossibility of establishing global state and the difficulty to synchronize processes. Even though the control design is more manageable through the software module concept of Function Blocks, it is still challenging to grasp the overall behaviour of the distributed system without computer-aided verification process, especially when each controller in the system network is designed by a different developer. Another challenge comes from the system engineering side. In order to be used in industry, the perceived switching cost to this new Function Blocks approach needs to be less than the perceived benefit. The costs of the change can be very substantial especially in restructuring and retraining to familiarise with the new design approach and new design tools (Peltola, Christensen, Sierla, & Koskinen, 2007). This problem leads to an idea of linking existing tools and languages with Function Blocks. Therefore improving validation and verification by using existing tools can increase the industrial adoption of distributed automation.

* Corresponding author.
E-mail address: cyan034@aucklanduni.ac.nz (C.-h. Yang).

0967-0661/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.conengprac.2012.06.008

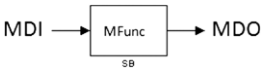

Fig. 1. An example of a Simulink block.

System simulation using closed-loop plant-controller modelling is a standard technique in control engineering and is getting recognized in industrial automation as a way of testing and debugging of automation software (Vyatkin, Hanisch, Pang, & Yang, 2009). So far the Function Blocks architecture of IEC 61499 has been used mainly for implementation of distributed controllers rather than models of the plant. In the authors' view, however, the ability to integrate plant and model in a single execution and development framework is one of the benefits of IEC 61499 architecture. The Model-View-Control (MVC) design pattern has been introduced in the Function Block domain as a way to validate and verify the design (Christensen, 2000). Using this pattern, one can combine simulation model and controller into one component, and the connection of such components results in a working simulation model of the entire distributed system. It has been demonstrated in a number of publications (Black & Vyatkin, 2008; Vyatkin, 2007b; Christensen, 2000), that using Function Blocks as a modelling language is feasible and beneficial. However, the question left is how to obtain these models in a cost-efficient way.

The answer to this question leads to the proposed model transformation approach, where the required models can be obtained by transforming existing models from other popular modelling software tools such as MATLAB Simulink. In this work, MATLAB Simulink is chosen as it is a powerful and well-known tool in the modelling and simulation domain. Also Simulink follows the block-diagram modelling approach which is very close to the modular approach of Function Blocks.

Under assumption of a particular FB execution semantics, the FB standard can be considered as an executable specification for distributed control systems enabling system-level design of distributed controllers with capability of direct deployment, while modelling tools such as MATLAB Simulink are good only for simulation and analysis of control systems.

This paper addresses the formal side of the model transformation method whose idea was earlier presented by the authors in Yang and Vyatkin (2010).

The rest of the paper is organised as follows. Section 2 summarizes the reasons for developing the model-transformation method. Section 3 explains the proposed method in detail. Section 4 discusses the equivalence of the transformation results with the original models. A comparison of the output results from both transformed model and the original Simulink model is presented in Section 5. Finally Section 6 discusses the future direction of this research and Section 7 concludes the paper.

2. Motivation of model transformation

The proposed model transformation aims at providing the necessary plant (and, possibly, controller) models to be used in the simulation environment for validation of distributed systems compliant with IEC 61499. There are also other reasons why the model transformation to Function Block domain is useful:

Easier re-use in FB form: In Simulink, if a same block is used throughout the system many times, and for some reason it requires modification, then this is to be done in its every occurrence. Function Blocks, on the other hand, can be modified more easily and updated to all block instances. In this scenario, Function Block tools have advantage from the software reusability perspective.

Potential to improve performance: The distributed nature of FBs implies the ability to run models in parallel on distributed computers. There is a third-party distributed toolbox for MATLAB/Simulink that enables concurrent simulation on several communicating computers, which justifies the importance of distributed simulation. Translation of MATLAB models to the FB environment allows for more control over the concurrent model execution. In FB one can take advantage of highly efficient compilers of IEC 61499 (Yoong, Roop, Vyatkin, & Salcic, 2009b) and of execution platforms with hard real-time guarantees (Zoitl, 2009).

Reduce the conflict in the nature of the execution between two models: One may argue that the simplest way to connect the two modelling environment can be achieved by a standard communication channel such as UDP or TCP connections, such as the approach introduced in Maturana, Ambre, Staron, Carnahan, and Loparo (2011) where synchronization between MATLAB and SoftPLC is performed. This option was investigated by the authors in Yang and Vyatkin (2008) for IEC 61499 implementation, and it can be a better option for some design scenarios. While, in general, it works and allows for co-simulation of Function Block controllers with plant models in Simulink, there are some limitations and overheads. The execution semantics of both languages is different (and there are variations between different FB implementations), so one has to build a communication channel that does not depend on these differences.

Another important reason for model transformation is to support model-based control design methods, such as model-predictive control (MPC), control by estimation, etc. These are common and advanced control techniques in process industries such as chemical plants and oil refineries automation. The transformation approach can provide an easier way of obtaining necessary models that can assist in making control decisions.

Industrial practices show the great need for comprehensive system level simulation frameworks. Such established modelling frameworks as MATLAB, LabView and PS-CAD provide mechanisms for code generation and deployment as a part of the engineering workflow. What is proposed and investigated in this paper enables a similar block diagram modelling approach, but based on an open international standard and with native support of distributed architectures. This would enable a very smooth pathway to the deployment of distributed control code upon the validation with less disturbances than in the code generation, e.g., from MATLAB.

3. Model transformation method

The syntax and semantics of both MATLAB Simulink and Function Blocks must be understood, formally defined and taken into consideration before transforming the models between the two. It must be done in a way that the behaviour of the models remains the same after switching to another platform. To ensure the semantic equivalence, a formal mapping between both languages will be established in this section. First, Simulink and Function blocks objects will be defined using some mathematical notation and then the mapping will be described.

3.1. MATLAB Simulink

Simulink is a software package from Mathworks Inc., which provides an environment for simulation and model-based design

Please cite this article as: Yang, C., & Vyatkin, V. Transformation of Simulink models to IEC 61499 Function Blocks for verification of distributed control systems. *Control Engineering Practice* (2012), http://dx.doi.org/10.1016/j.conengprac.2012.06.008

of dynamic control or embedded systems. It is tightly integrated with MATLAB. Simulink has a library storing many standard blocks that are often used in modelling.

The syntax and semantics of Simulink and Stateflow have already been defined and well-documented in Clawz (2003) and Mathworks, 2010. Further it will be defined using the following mathematical notation, which will be used throughout the paper.

A Simulink block can be defined as a tuple SB=(MInterface,MFunc,MIV), where MInteface is the data interface of the Simulink block, MFunc defines the behaviour of the block as a function, and MIV defines the internal parameters of the block:

MIV={m*iv*1,*miv*2,*miv*3,…} is a set of internal variables;

MInterface=(MDI,MDO), where

MDI={m*di*1,*mdi*2,*mdi*3,…}=is a set of data inputs

MDO={m*do*1,*mdo*2,*mdo*3,…}=is a set of data outputs

$$\text{MFunc}: \prod_{\text{mdi}\in\text{MDI}} \text{Dom(mdi)} \times \prod_{\text{miv}\in\text{MIV}} \text{Dom(miv)} \rightarrow \prod_{\text{mdo}\in\text{MDO}} \text{Dom(mdo)}$$

where Dom(x) represents the domain of variable x.

Stateflow is one of such packages that allows specifying customized blocks with finite state machine (FSM) diagram. Stateflow is an interactive graphical design tool that works with Simulink to model and simulate event-driven systems. Fig. 2(a) shows an example of a Stateflow block, and the encapsulated state charts are shown in Fig. 2(b). It allows modelling event-driven systems in a State Chart dialect whose syntax and semantics is extended from the traditional Finite State Machines (FSM) by allowing hierarchical charts, parallel states, and using temporal logic to schedule events.

So, instead of having MFunc as in a Simulink block, Stateflow has a FSD, which represents the Finite State Diagram. Therefore, if SFB is a single Stateflow block, then SFB is determined by:

SFB=(MInterface,MIV,FSD),where:

FSD=(MState,MTrans,Func,MCond) is finite state diagram of a Stateflow block,

Func={*f*0,*f*1,*f*2,*f*3,…}is a set of functions (i.e., algorithms),

MState={*ms*0,*ms*1,*ms*2,*ms*3,…} is a set of states, where *s*0 is the initial state,

MTrans=MState × MState is a set of transition between states,

MCond is a set of the conditions for the transition to take place, where:

$$\text{MCond}: \text{MTrans} \rightarrow \left[\prod_{\text{mei}\in\text{MEI}} \text{Dom(mei)} \times \prod_{\text{mdi}\in\text{MDI}} \text{Dom(mdi)} \times \prod_{\text{meo}\in\text{MEO}} \text{Dom(meo)} \times \prod_{\text{miv}\in\text{MIV}} \text{Dom(miv)} \rightarrow \text{true, false}\right]$$

Modelling in Simulink is done by creating a network of "blocks," stored in the Simulink library. Simulink supports hierarchical structuring of models by grouping the related blocks into "subsystems." However the subsystem layout only groups the selected blocks together for the better display purpose only. It does not affect the order of blocks' execution.

Simulink supports simulation for discrete, continuous and even hybrid systems through its corresponding solvers which compute the states of the system at successive time steps over a specified time span (Ray, 2007). These time steps can be of fixed or variable duration. The fixed-step simulation solver calculates the states of the system at fixed time intervals. The variable-step simulation relies on the solver to determine the length of the time steps, and the time steps will vary over time. When the parameters are changing rapidly, the step size will be decreased, and vice versa. The solvers are generally categorised into continuous solvers and discrete solvers. Continuous solvers compute the state of a system in the current time step by using numerical integration from the state of the system in the previous time step and the state derivatives. Discrete solvers primarily solve only discrete models. They rely on the model blocks to update the discrete states of the models. There is no single method that can solve all types of models.

Simulink automatically assigns execution priority of the blocks, based on a set of fundamental rules. These rules state that a block generally has a higher priority than the one where its output data is connecting to. But sometimes there are exceptions where some Simulink blocks automatically have higher priority than other blocks (i.e., Integrator, etc.). It also allows assignment of user-defined priorities to each block in order to determine execution order as long as the fundamental rules are not breached. But if the priority is not defined, the system automatically assigns priority to each single block. State of each block (i.e., input and output values of the block) will be updated during a single time step of execution, in the order following the priority.

Sometimes Simulink prohibits closed-loop data connection because it is not able to determine the execution priority in this case. The recommendation is to add a "Memory" block to break down the loop (see Fig. 3).

Despite the extensive description of Simulink syntax and semantics, there is still one concern related to determining the

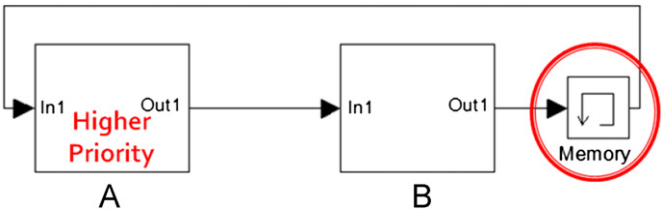

Fig. 3. Memory block allows closed-loop connection in Simulink.

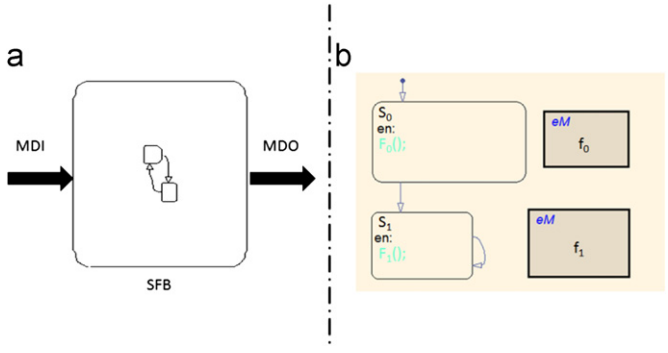

Fig. 2. (a) An example of a Stateflow block. (b) The finite state chart inside the Stateflow block.

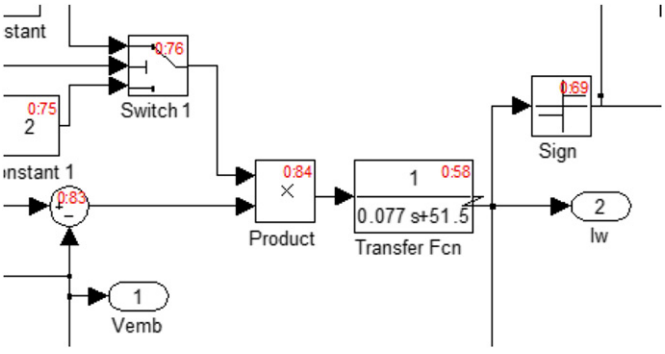

Fig. 4. An exception case for the priority order.

Please cite this article as: Yang, C., & Vyatkin, V. Transformation of Simulink models to IEC 61499 Function Blocks for verification of distributed control systems. *Control Engineering Practice* (2012), http://dx.doi.org/10.1016/j.conengprac.2012.06.008

execution order. For example, in the situation shown in Fig. 4, the Transfer Fcn block actually has a higher priority than Product block. This is because blocks such as Transfer Fcn have non-direct feedthrough inputs, and therefore this block may be placed anywhere in the execution order. This affects the decision on the priority order in the transformation process. The priority order information is determined by Simulink and this information cannot be automatically extracted, therefore the transformation software has to determine the priority on its own. Models, which contain these blocks having non-direct feedthrough inputs, cannot be sorted in the same order as the Simulink model, and a manual adjustment has to be performed.

More details about MATLAB Simulink and its packages can be found in Mathworks' official documentation and website (Mathworks, 2010).

3.2. Function Blocks of IEC 61499

3.2.1. Introduction

The IEC 61499 standard introduces Function Blocks as a new modular and event-driven way of designing controllers and modelling distributed control systems. It is believed that modelling systems with Function Blocks will improve the flexibility, software reusability and reconfigurability in distributed control systems design from both software and hardware perspective (Vyatkin, 2007b).

The IEC 61499 standard defines a few design artefacts, such as basic and composite Function Blocks. A basic Function Block is a single event-driven module whose logic is specified by Execution Control Chart (ECC)—a kind of finite-state machine. Therefore the structure of a basic Function Block is very similar to a single MATLAB Stateflow block. ECC describes the conditions of transitions between states and algorithms associated with each state. An example of a basic Function Block and ECC can be found in Fig. 5. A Function Block system can be hierarchically organized through the use of the "composite Function Blocks," which describe a subsystem of the whole Function Block network. However all the Function Blocks inside a composite Function Block have exactly the same priority status comparing to the ones at the same hierarchy level as the composite Function Block. This is different to the hierarchy of subsystems in Simulink therefore it would require special attention in the transformation.

There are numerous works on verification and validation of IEC 61499 based systems, which can be classified in three categories: simulation, formal verification and specification compliance. Implementation of simulation in FB environment was demonstrated in Christensen (2000). The specification compliance at an abstract level before actual design and implementation can be done by bridging Function Blocks with UML or SysML, e.g., Thramboulidis (2004), Dubinin and Vyatkin (2008), Dubinin, Vyatkin, and Pfeiffer (2005), Panjaitan (2008), Christensen (2000) and Hirsch (2010). The formal verification of Function Blocks is done through their modelling in various formal languages, such as NCES, Timed automata, State charts, etc. which can be verified using model-checkers such as SESA, ViVe, SMV, (Cheng & Vyatkin, 2008; Hagge & Wagner, 2005; Vyatkin, Hanisch, & Pfeiffer 2003, Cengic & Akesson, 2010, Gerber & Hanisch, 2010).

3.2.2. Formal definition

The formal notation of IEC 61499 from Dubinin and Vyatkin (2008) is followed in this paper, according to which a Basic Function Block is determined by a tuple FB=(Interface, ECC, ALG, IV), where:

ALG={*alg*1,*alg*2,*alg*3,…} is a set of algorithms, and it is possible that ALG= QUOTE .

Interface=(EI,EO,DI,DO,IA,OA), where:

EI {*ei*1,*ei*2,*ei*3,…} is a set of event inputs, where ∀ei ∈ EI[Dom(ei) = true, false],

EO={*eo*1,*eo*2,*eo*3,…} is a set of event outputs,

DI = *di*1,*di*2,*di*3,… is a set of data inputs,

DO = *do*1,*do*2,*do*3,… is a set of data outputs,

IV={*iv*1,*iv*2,*iv*3,…} is a set of internal variables.

IA is a set of input event/data associations, where $IA \subseteq EI \times DI$ *and* $VI\backslash IA = \emptyset$.

OA=a set of output event/data associations, where $OA \subseteq EO \times DO$ *and* $VO\backslash OA = \emptyset$.

ECC is the execution control chart of a basic Function Block, and it is determined by tuple (ECState, ECAction, ECTransition, ECCondition).

ECState = *s*0,*s*1,*s*2,*s*3,…..=a set of states, where s0 is the start state.

ECTransition=ECState × ECState=a set of transition between states.

ECAction : ECState\s0→ALGxEO∪ALG∪EO.

ECCondition is a set of the transition conditions. A transition condition is a predicate over event and data inputs, where:

$$\text{ECCondition} : \text{ECTransition} \rightarrow [\prod_{ei \in EI} \text{Dom(ei)} \times \prod_{di \in DI} \text{Dom(di)} \times \prod_{eo \in EO} \text{Dom(eo)} \times \prod_{iv \in IV} \text{Dom(iv)} \rightarrow \text{true, false}]$$

Execution of Function Blocks is achieved by compiling them into executable code which works in conjunction with some predefined libraries. The code generation model plus the libraries are commonly referred to as a run-time environment. One of its important tasks is to dispatch events among Function Blocks. Currently, there are several different run-time environments which are implemented with different execution models. In the current exercise, the Function Block Run Time (FBRT) and

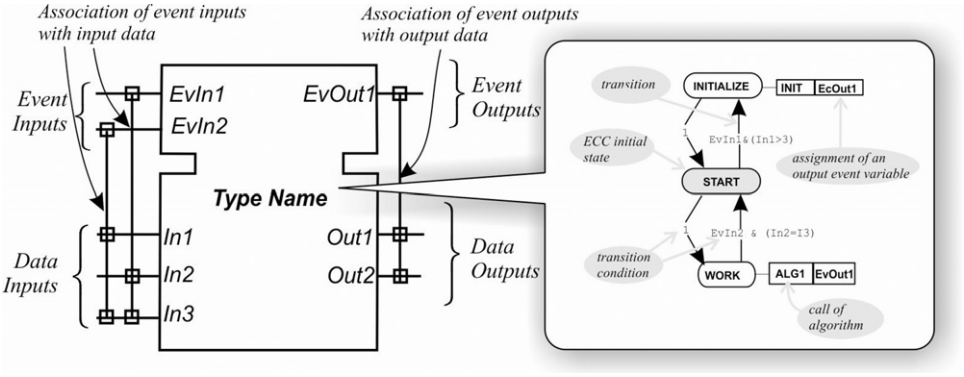

Fig. 5. Interface and ECC of the basic Function Block.

Please cite this article as: Yang, C., & Vyatkin, V. Transformation of Simulink models to IEC 61499 Function Blocks for verification of distributed control systems. *Control Engineering Practice* (2012), http://dx.doi.org/10.1016/j.conengprac.2012.06.008

Function Block Development Kit (FBDK) (Holobloc, 2009) are used for demonstration purposes. The Function Block Run Time (FBRT) has the longest history in the IEC61499 community. It uses "direct function calls" execution semantics where an output event triggers the successive Function Blocks in a single thread. However, this mechanism may result in stack overflow in case of event feedback loop, or in starvation of some blocks because the execution process of the caller block will be halted until all other Function Blocks along the event propagation path have completed execution. There is one solution to this problem by the use of LOOP_END block, which will be described later in the paper. FBRT is written in Java, and is the built-in run-time of FBDK.

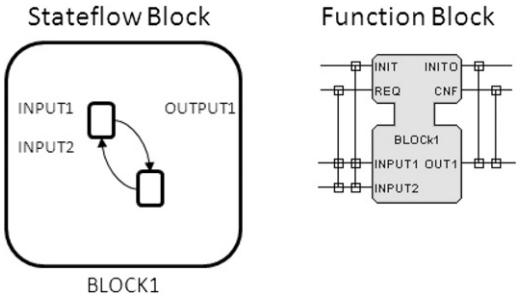

Fig. 6. A Stateflow block is transformed into a basic FB.

3.3. Block-to-block mapping principle

The main concept of this transformation methodology is through a block-to-block mapping. The transformation must be done in a way that preserves structural properties of the original model.

For every Simulink block (SB) a corresponding FB is created. The model transformation method is defined as a function M, where:

$M : \mathrm{SB} \rightarrow \mathrm{FB}$

There is one-to-one correspondence between input sets of both blocks, i.e.,:

$\forall m\mathrm{di} \in \mathrm{MDI} \ni !M(\mathrm{mdi}) = \mathrm{di} \in \mathrm{DI}$

In this model, internal variables are included as a part of data input because it is not easy to modify internal variables in the current Function Block development tools. This may be changed to be mapped directly to IV in the Function Block side.

$\forall \mathrm{miv} \in \mathrm{MIV} \ni !\mathrm{M}(\mathrm{miv}) = \mathrm{di} \in \mathrm{DI}$

This only changes the interface so that:

$$\prod_{\mathrm{mdi} \in \mathrm{MDI}} \mathrm{Dom}(\mathrm{mdi}) \times \prod_{\mathrm{miv} \in \mathrm{MIV}} \mathrm{Dom}(\mathrm{miv}) = \prod_{\mathrm{di} \in \mathrm{DI}} \mathrm{Dom}(\mathrm{di}) \times \prod_{\mathrm{iv} \in \mathrm{IV}} \mathrm{Dom}(\mathrm{iv})$$

In order to implement the logic of algorithms' invocation, the ECC of FB is created as:

ALG$=\{alg_1, alg_2\}$, where alg1 handles initialisation and $alg_2 =$ MFunc;

ECState$=\{S_0, S_1, S_2\}$, S_0 is the initial state

Two pairs of event inputs and outputs are added. INIT and INT0 handle the initialisation, while REQ and CNF handle the execution of the blocks:

$\mathrm{EI} = \mathrm{INIT,REQ}$ and $\mathrm{EO} = \mathrm{INIT0,CNF}$

All data is associated with EI and EO: IA$=$EI $\times$ DI,OA$=$EO $\times$ DO

After the initialisation, FB is activated when:

$$\mathrm{REQ} = \mathrm{true} \Rightarrow \mathrm{CNF} = \mathrm{true} \cap [alg_2 : \prod_{\mathrm{di} \in \mathrm{DI}} \mathrm{Dom}(\mathrm{di}) \times \prod_{\mathrm{iv} \in \mathrm{IV}} \mathrm{Dom}(\mathrm{iv}) \rightarrow \prod_{\mathrm{do} \in \mathrm{DO}} \mathrm{Dom}(\mathrm{do})]$$

But on the Simulink side,

$$\mathrm{MFunc} : \prod_{\mathrm{mdi} \in \mathrm{MDI}} \mathrm{Dom}(\mathrm{mdi}) \times \prod_{\mathrm{miv} \in \mathrm{MIV}} \mathrm{Dom}(\mathrm{miv}) \rightarrow \prod_{\mathrm{mdo} \in \mathrm{MDO}} \mathrm{Dom}(\mathrm{mdo}).$$

Since MFunc$=alg_2$, if t denotes an instance of time where $\mathrm{t} \in \Re \cap \mathrm{t} \geq 0$,

$$\forall \mathrm{t} : \mathrm{DO}^{\mathrm{t}} = \prod_{\mathrm{do} \in \mathrm{DO}^{\mathrm{t}}} \mathrm{Dom}(\mathrm{do}) = \prod_{\mathrm{mdo} \in \mathrm{MDO}^{\mathrm{t}}} \mathrm{Dom}(\mathrm{mdo}) = \mathrm{MDO}^{\mathrm{t}},$$

where $t = 1, 2, 3, \ldots, n$ (for discrete-time state).

A single Stateflow block is transformed into a basic Function Block. Fig. 6 shows the interfaces of both blocks.

Instead of MFunc mapping in a normal Simulink block, actually it is possible to directly map SFD to ECC as follows:

$\mathrm{M} : \mathrm{FSD} \rightarrow \mathrm{ECC}, M : \mathrm{MState} \rightarrow \mathrm{ECState}, M : \mathrm{MCon} \rightarrow \mathrm{ECCondition}, M : \mathrm{Func} \rightarrow \mathrm{ALG}, M : \mathrm{MTrans} \rightarrow \mathrm{ECTransition}$

Transforming Stateflow diagram from Simulink to the ECC of the basic Function Blocks is almost straightforward as long as the classic Stateflow semantics is used, that only takes one transition at each simulation time step. Fig. 7(a) presents an example of such a transformation. Each ECC transition is "clocked" with an input event REQ that is connected to the tick generator implementing an equivalent of the Simulink's time step. Upon one execution the FB emits CNF event which can be connected to the REQ of next FB in the row. Another possible semantic option of Stateflow is known as "super step" in which case the execution of SFD continues as long as there are enabled transitions. In the corresponding ECC this may involve visiting several EC states after invocation by REQ, but CNF needs to be emitted once in the end of this execution chain. For this semantic option another ECC pattern is suggested as shown in Fig. 7(b). Each SFD state MStatei is mapped onto two ECC states Si and tSi. To enable a series of state visits, another event output REP is introduced which is looped back to the same FB's input CHK. If MStatei is the last state in such a series, the ECC would stop its run in tSi. Other Simulink semantics options and FSD features (e.g., hierarchical states) can be addressed in a similar way through the development of the corresponding ECC patterns.

The algorithm inside Stateflow's FSM is written in MATLAB code, which needs to be translated into a language that can be recognized by the Function Block tools. The MATLAB programming language is fairly similar to the Structured Text (ST) language used in PLC programming and in algorithms of Function Blocks. In this work, a simple algorithm translator has been developed, which can translate basic equivalence statements, math equations and IF statement. Mathworks (2011) provides "Simulink PLC Coder" allowing generation of hardware independent IEC 61131-3 ST code from Simulink blocks and Stateflow blocks. However, due to the difference in the execution of FBs in the IEC 61131-3 and IEC 61499 standards, the usability of this generated ST code requires extensive investigation to work properly in the IEC 61499 context.

A subsystem of Simulink is be mapped to a composite Function Block. Eventually the Simulink model will be transformed into a Function Block application. Fig. 8 presents a simple example of

Please cite this article as: Yang, C., & Vyatkin, V. Transformation of Simulink models to IEC 61499 Function Blocks for verification of distributed control systems. *Control Engineering Practice* (2012), http://dx.doi.org/10.1016/j.conengprac.2012.06.008

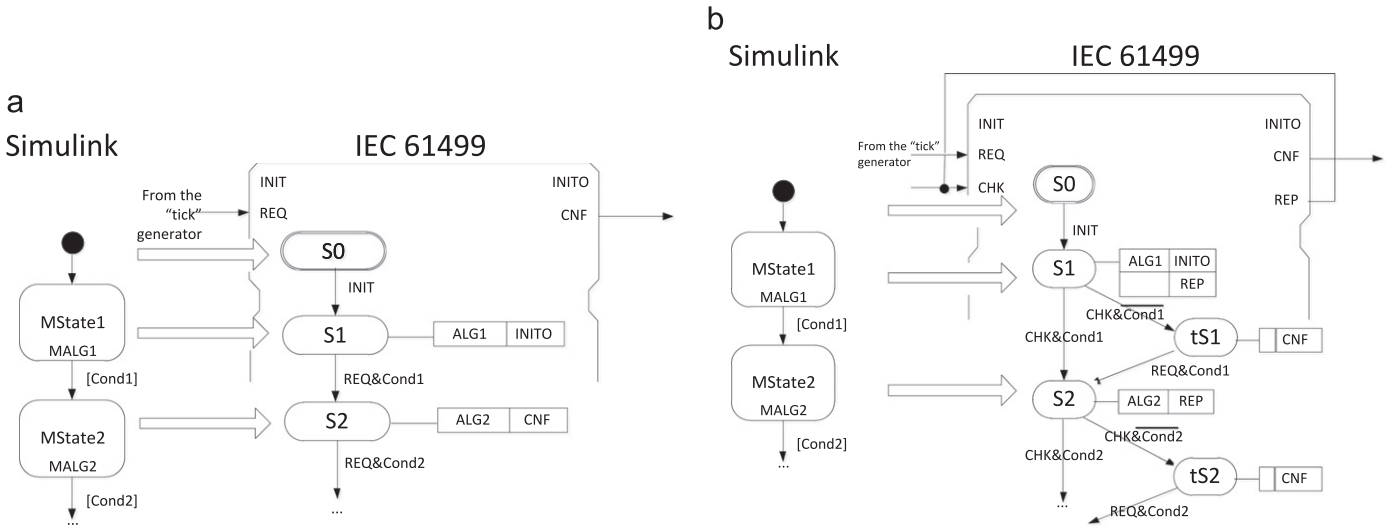

Fig. 7. Stateflow's FSM is mapped to FB's ECC: (a) classic semantics; (b) "super step" semantics.

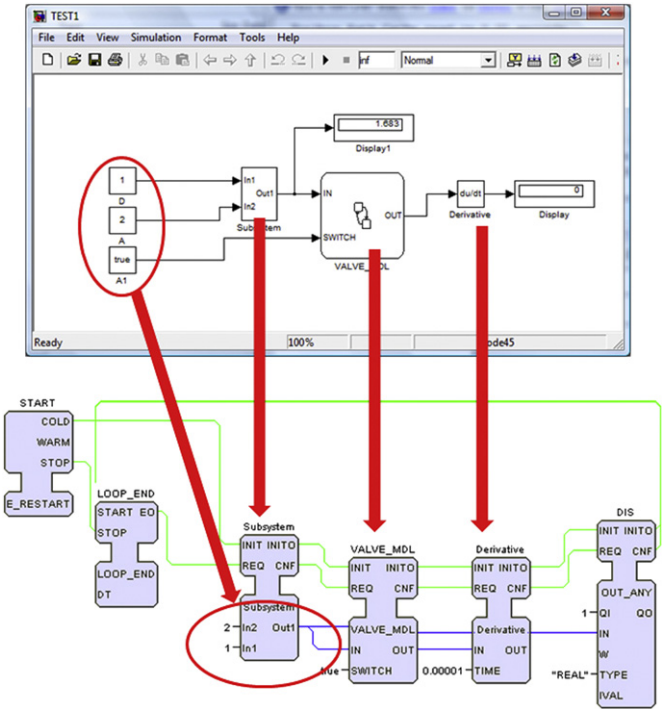

Fig. 8. The transformed Function Block System.

model transformation, following the block-to-block transformation approach. In order to implement the cyclic execution order of Simulink in the Function Blocks model, a "LOOP_END" block is introduced in FBDK environment with all the other Function Blocks connected in a sequential order (also see Fig. 8). This LOOP_END acts just like the E_DELAY Function Block but with zero delay time. Because of the threading effect of the E_DELAY block, it manages to set a break between function calls and therefore allows events to be connected in closed-loop in FBDK.

3.4. Data handling

Table 1 shows the data types mapping between Function Blocks (in FBDK) and MATLAB Simulink/Stateflow at this stage of development.

Table 1
Data types mapping between Function Blocks (in FBDK) and MATLAB Simulink.

Data type	Comment	Memory space	FBDK support	MATLAB support	Both supported
Byte	Short integer	8 bits	SINT	Int8	Yes
Int	Integer	16 bits	INT	Int16	Yes
Int	Double integer	32 bits	DINT	Int32	Yes
Short	Unsigned short integer	8 bits	USINT	Uint8	Yes
Int	Unsigned integer	16 bits	YUBT	Uint16	Yes
Long	Unsigned double integer	32 bits	UDINT	Uint32	Yes
Single		32 bits	REAL	Single	Yes
Boolean		8 bits	BOOL	Boolean	Yes
Double		64 bits	LREAL	Double	Yes
String		Variable size	WSTRING	–	No
Byte	Unsigned	8 bits	BYTE	–	No

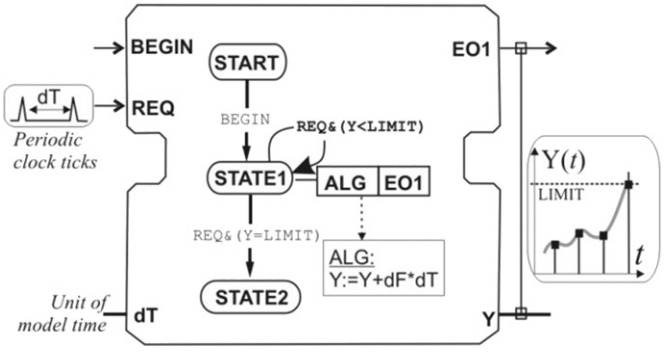

Fig. 9. FB implementation of a discretized computation of a continuous function *F* given its derivative d*F*.

In general, control systems use both discrete and continuous data. A system may contain discrete operations (such as on and off switches) as well as continuous control (such as movement or flows of objects). Such systems are also known as "hybrid" (Lunze, 2002). In process industry, a batch processing is one example of such a system. The process in the batch process reactor, for example, can be described by both continuous variables (e.g., temperature) and discrete variables (e.g., switches). Modelling, verification and validation of hybrid systems, i.e., continuous and discrete dynamics, executing intelligent control algorithms in

Please cite this article as: Yang, C., & Vyatkin, V. Transformation of Simulink models to IEC 61499 Function Blocks for verification of distributed control systems. *Control Engineering Practice* (2012), http://dx.doi.org/10.1016/j.conengprac.2012.06.008

decentralised nodes, are known to be computationally hard. This again shows the importance of software environment linking the Function Block design with MATLAB Simulink which has readily validation and verification tools.

Some Simulink blocks are associated with the sampling time of the system execution especially in the continuous time modelling. The transformed Function Block model must use the same "time step," which can be achieved by setting the "dT" parameter of model FBs. It is illustrated in Fig. 9. The model FB is activated by an external "clock" signal. If simulation in real time is required (e.g., for animation of some processes) then the E_CYCLE or RT_E_CYCLE Function Blocks, as suggested in Zoitl (2009), can be used as the clock generator. However, in the scenario using sequential event connection (with scheduler if necessary) it is enough for achieving model-time simulation.

For these types of blocks, the "dT" input parameter must be given for their corresponding Function Blocks. Simulink supports a variable time-step solver and a so-called continuous solver. Thus in this case, these data must be discretised in order to perform the matching solution in the Function Block domain. The "dT" parameter indicates the discretisation rate (or sampling rate).

4. Transformation equivalence

In this section, a proof of semantic equivalence of the generated FB model with original Simulink model is provided. Two blocks are equivalent if they produce equal output values at all discrete time moments. One unit of time, *t*, is defined as the time period when all blocks are executed exactly once. The following assumptions are made:

The same algorithm expression will produce exactly the same output from both models.
Simulink is assumed to be executing in a sequential and cyclic order with certain priorities assigned to each block.
The Function Block models are forced to execute in a sequential and cyclic order by the use of event connections, despite the original execution semantics (i.e., after the last FB is executed, the next cycle starts by executing the first block with all the updated data I/Os.)
Data types are all mapped properly.
Descretization is handled properly causing no difference to the Simulink model.

The proof of equivalence resulting from the model transformation is done by mathematical induction. For a single block, there is no proof required since the matching is already done by the above rules and assumptions. So the base of induction can start with two blocks (see Fig. 10). By the previous block-to-block mapping in Section 2.4:

$$DI_1^t = MDI_1^t \Rightarrow DO_1^t = MDO_1^t, \forall t \in \Re \cap t \geq 0$$

Assuming sb1 is always executed before sb2 in Simulink model, and *mo* denotes a set of data outputs connected to sb1:

$$MDI_2^t = mo \cup mc \cup mk, \text{ where } mo\ MDO_1^t, mk\ MDO_2^{t-1}$$

and *mc* is a subset of constantsfb1 and fb2 are forced to be executed in the same order by:

$$EI_2^t = EO_1^t.$$

Since all data are associated with all events in the block-to-block mapping,

$$DI_2^t = o \cup c \cup k,$$

where $o \subseteq DO_1^t$, $k \subseteq DO_2^{t-1}$ and c is a set of constants

But the transformation is making:

$o = mo$, $k = mk$, and $c = mc$,

$$\therefore DI_2^t = mo \cup mc \cup mk = MDI_2^t.$$

Again, by the block-to-block mapping:

$$DI_2^t = MDI_2^t DO_2^t = MDO_2^t.$$

Now the proof of equivalence is required for the case with *N* numbers of blocks (see Fig. 11). Assuming the equivalence is true for *N* blocks, by the block-to-block mapping:

$$DI_N^t = MDI_N^t \Rightarrow DO_N^t = MDO_N^t, \forall t \in \Re \cap t \geq 0.$$

Assuming sb*N* is always executed before sbN+1 in Simulink model, and mo denotes a subset of data outputs connected to sbN:

$$MDI_{N+1}^t = mo \cup mc \cup mk, \text{ where } mo \subseteq MDO_1^t, mk \subseteq MDO_2^{t-1}$$

and *mc* is a set of constants

Assuming Function Block models are now all executed in a sequential order, forcing fb*N* executed before fb*N*+1 (like the order in Simulink side) by using events:

$$EI_{N+1}^t = EO_N^t.$$

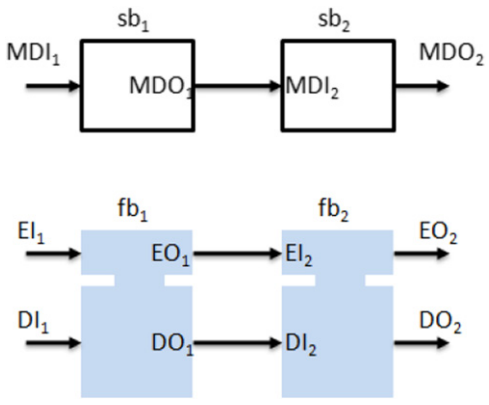

Fig. 10. Mapping for a system with two blocks.

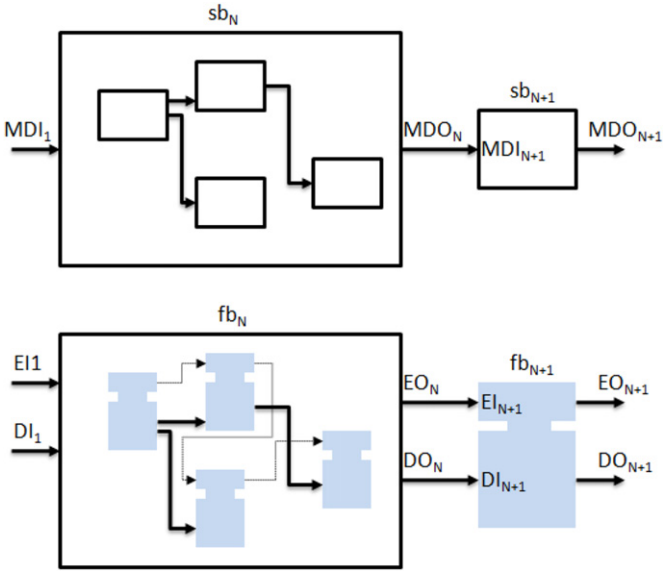

Fig. 11. Mapping for system with *N*+1 blocks.

Please cite this article as: Yang, C., & Vyatkin, V. Transformation of Simulink models to IEC 61499 Function Blocks for verification of distributed control systems. *Control Engineering Practice* (2012), http://dx.doi.org/10.1016/j.conengprac.2012.06.008

Since all data are associated with all events,

$DI^t_{N+1} = o \cup c \cup k$, where $o \subseteq DO^t_N, k \subseteq DO^{t-1}_2$ and c is a set of constants.

But the transformation rules are making:
$o = mo, k = mk$ and $c = mc$,

$\therefore DI^t_{N+1} = mo \cup mc \cup mk = MDI^t_{N+1}$.

Again, by the block-to-block mapping:

$DI^t_{N+1} = MDI^t_{N+1} DO^t_{N+1} = MDO^t_{N+1}$

$\therefore$ the equivalence still holds for $N+1$ blocks.

5. Implementation results

The model transformation has been implemented in Java and FBDK is chosen as the target Function Block development environment in this work, even though its execution semantics has known issues (Cengic, Ljungkrantz, & Akesson, 2006). However it is sufficient enough for academic demonstration, at such early stage of the development. The required modifications for other semantic models are expected to be minor.

The developed software implementation of the transformation can be categorized into three phases (see Fig. 12). The first phase is a model translator that is able to convert all the relevant elements of the Simulink models into software objects, referred as the "Simulink Parser." The second one is the translator that can construct the Function Block models from software objects in a specific format, referred as the "Function Block Model Generator." The last one acts like a middle-ware that reconstructs the software objects from the first translator into the specific format that can be imported to the second translator.

The mapping between the two models can be represented using the corresponding meta-models, shown in Fig. 13. The Function Block meta-model is inherited from the meta-model of basic Function Block presented in Hussain (2006).

A motor Simulink model (Fig. 14) provided by an industrial partner has been experimented with this transformation approach. The motor example is composed of a controller and a motor driving a fan. The sensor readings from the motor and the fan are collected by the controller before passing the resultant control command to the motor.

The output result from the motor Simulink model can be seen in Fig. 15. The most important output here is the EMF reading from the motor. In this particular motor design, this data indicates the desired behaviour expected from the motor. The sine wave signal is the main voltage just for reference purpose. The pulse-like signal is the output signal from controller to the motor.

The Simulink model of the motor has been successfully transformed into a Function Block model that can be executed under FBDK. The resultant Function Block model after the transformation can be seen in Fig. 16. The execution order of the

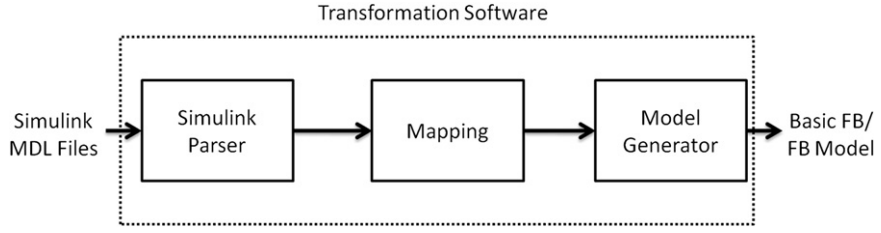

Fig. 12. The phases of the model transformation in software implementation.

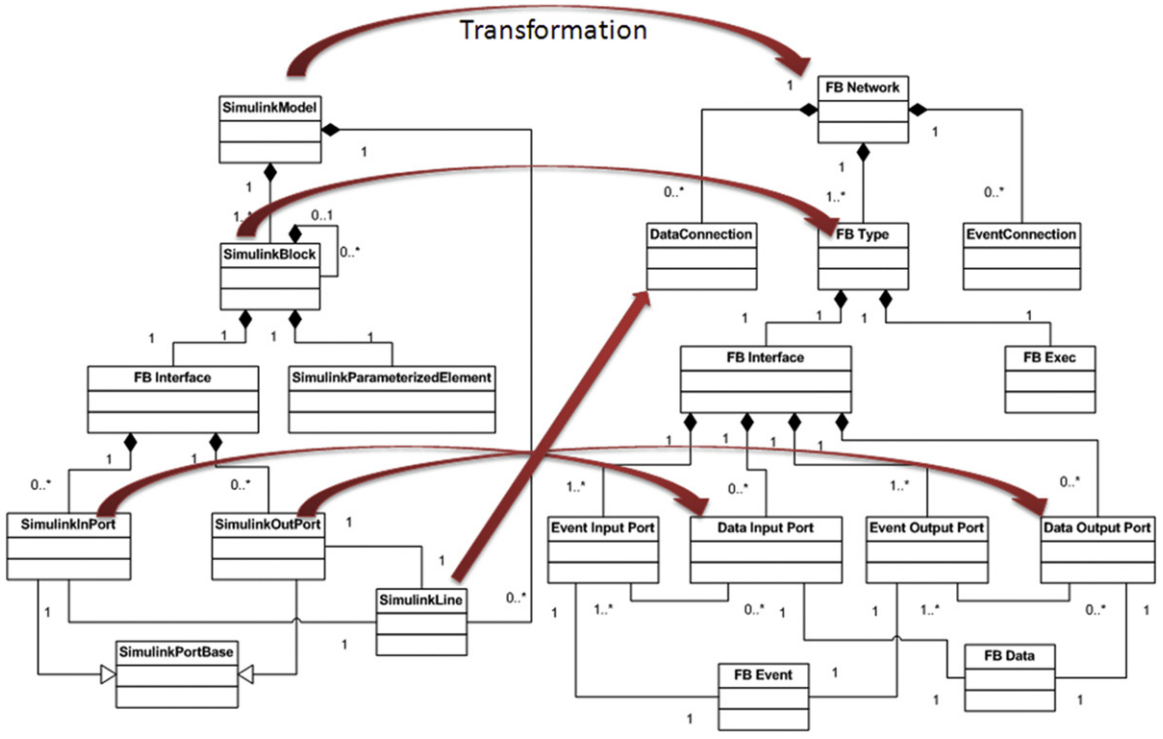

Fig. 13. Metamodel mapping between Simulink and Function Block.

Please cite this article as: Yang, C., & Vyatkin, V. Transformation of Simulink models to IEC 61499 Function Blocks for verification of distributed control systems. *Control Engineering Practice* (2012), http://dx.doi.org/10.1016/j.conengprac.2012.06.008

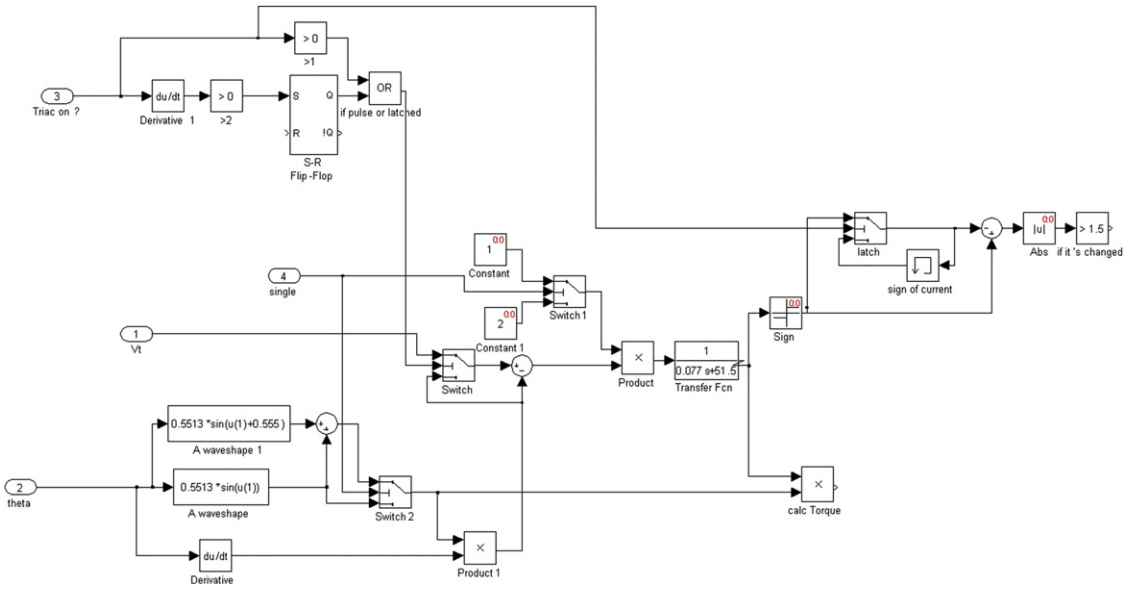

Fig. 14. An example of motor Simulink model.

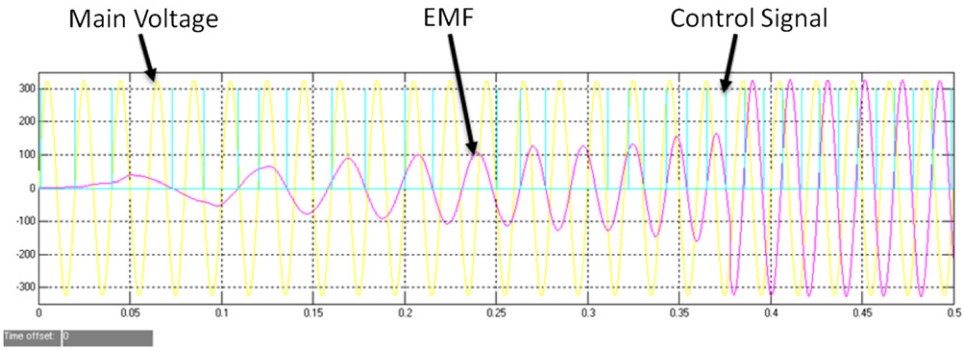

Fig. 15. Output results from the motor Simulink model.

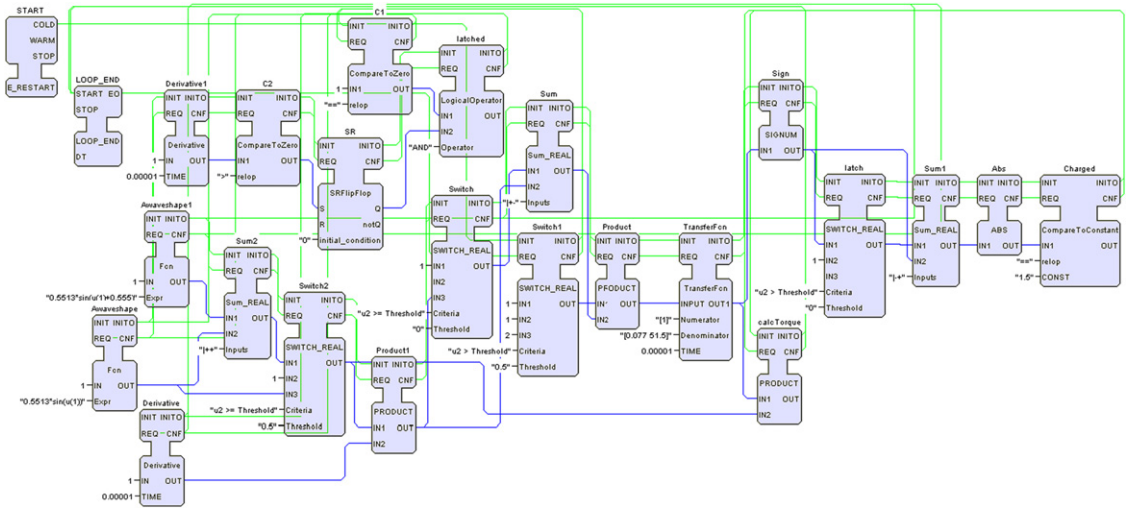

Fig. 16. The transformed FB System.

resultant transformed Function Block model may differ from the one in the original Simulink model due to the exception cases mentioned previously. Therefore some modification to the order of blocks is required before producing the desired results.

An experiment has been set up to feed exactly the same data input to both the original Simulink model and the transformed model. The output result from the transformed FB model is illustrated in Fig. 17. From the visual inspection, the results (EMF readings) are fairly identical.

There are still some slight differences in the results, even with matching execution order and semantics. In order to explain this difference, a simple experiment with the derivative Function Block in

Please cite this article as: Yang, C., & Vyatkin, V. Transformation of Simulink models to IEC 61499 Function Blocks for verification of distributed control systems. *Control Engineering Practice* (2012), http://dx.doi.org/10.1016/j.conengprac.2012.06.008

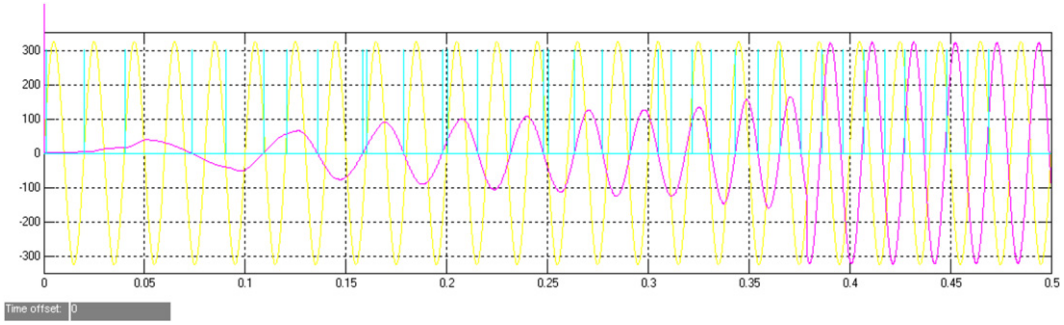

Fig. 17. Output results from the transformed Function Block model.

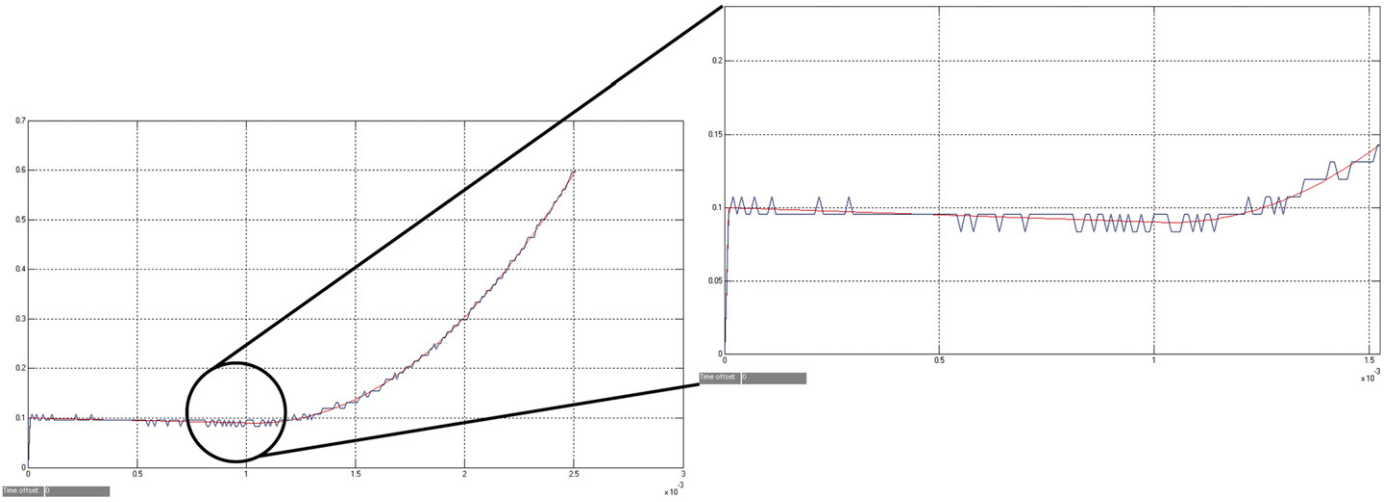

Fig. 18. Result comparison of the derivative block in Simulink and FBDK.

the motor model has been conducted using a comparator block. Fig. 18 shows the difference in the outputs of the derivative blocks. Simulink model produces a smooth output, while the output from FBDK results in some spikes. This is an expected influence of the discretization. This may be improved with more advanced modelling techniques or some averaging operation, when constructing the corresponding Function Blocks library. This exercise also demonstrated the importance of the execution priority. The result varies (i.e., out of phase) if the block are executed in a different order.

Although this methodology is based on a hypothesis that the behaviour of each basic Function Block used in the transformation is identical in execution to its corresponding Simulink block (i.e., the outputs are the same when provided exactly the same input at a specific time stamp), this is not fully true in this exercise. This is why the output results may seem to be slightly different. Some continuous blocks such as "Derivative" block and "Transfer Function" block can be created by more sophisticated modelling techniques. The source code of the library Simulink blocks is not publicly available, therefore the corresponding Function Block can only be created based on the behaviour expected and observed in simulation. This can surely be overcome by an expert in the field of control system modelling to create block in industrially acceptable standard. However in this exercise, these blocks are constructed in a simplified way that is sufficient and fairly close the "original" blocks from Simulink and are able to generate acceptable results for the proof-of-concept purpose.

6. Future work

Three future work directions are envisaged. First, it will include experiments with other Function Block tools, run-times and compilers, in order to expand the usage of this new approach with the proposed simulation environment. In particular, this approach will be applied in the model-predictive control scenarios, when the model is to be executed along with the controller on embedded devices. An example of such "embedded modelling" was provided by Yan and Vyatkin (2011). The embedded modelling has great potential, for example, for implementing unobservable control parameters as virtual sensors. Another direction of the work consists in building libraries of basic Function Blocks equivalent to various standard MATLAB/Simulink packages. And, finally, distributed simulation methods will be further developed, both from methodological and tool support perspectives.

One of such new FB implementation is "synchronous compiler" (Yoong et al., 2009b) that compiles Function Blocks into C code. This compiler is based on the synchronous execution model and is proven to be very efficient in terms of the target code performance (Yoong, Roop, & Salcic, 2009a). It can be very useful in the context of distributed control where the model can sometimes be big in size and resource consuming in simulation. Such models can be transformed to the open IEC 61499 form and to be run efficiently on distributed or centralized platforms.

Secondly, the authors' research group is developing a cyclic execution Function Block run-time (Tata and Vyatkin, 2009). This cyclic nature of execution immediately eliminates the stack overflow problem with FBDK. This is executed under FBench, which is another Function Block development tool (FBench).

The open nature of IEC 61499 allows application of the proposed simulation environment with any compliant IEC 61499 tool/runtime. The potential targeted tools are nxtControl (Nxtcontrol.COM., 2010) and 4DIAC (4DIAC, 2010). These Function Block development tools are based on FORTE runtime,

Please cite this article as: Yang, C., & Vyatkin, V. Transformation of Simulink models to IEC 61499 Function Blocks for verification of distributed control systems. *Control Engineering Practice* (2012), http://dx.doi.org/10.1016/j.conengprac.2012.06.008

following a First-In-First-Out (FIFO) sequential execution order (PROFACTOR). The proposed transformation approach would immediately provide a simulation add-on to these tools. ISaGRAF (IsaGRAF) is yet another company that is developing Function Block tools based on IEC61499 standard. However, ISaGRAF has own specification of the Function Block syntax, so a converter from the standard XML to ISaGRAF format would be required when performing the model transformation.

7. Conclusion

In this paper a mathematically rigorous transformation method from MATLAB Simulink for IEC61499 Function Blocks has been described. This method aims to help in design of complex distributed systems by allowing to take advantage of the simulation and analysis capability of MATLAB Simulink. The proposed approach has been validated on a number of industrial use-cases.

Main benefits of the proposed approach are:

– Automated model transformation reduces time and effort on model development and greatly helps in validating the designs based on IEC61499 Function Blocks.

– Embedded models extend the developers capabilities in implementation of model-predictive controls in the IEC 61499 environment;

These and other motivations listed in Section 2 can make the use of the proposed method attractive as a part of IEC 61499 commercial development environments.

Acknowledgements

This work was supported in part by the research grants of the University of Auckland: FRDF 3622763 and PRESS account of the University of Auckland.

References

2005.Function blocks: Internatinal Standard IEC61499. International Electrotechnical Commission. 4DIAC. 2010. Available: 〈http://www.fordiac.org/〉.

Black, G., & Vyatkin, V. (2008). Intelligent component-based automation of baggage handling systems with IEC 61499. *IEEE Transactions on Automation Science and Engineering, 7*(2), 337–351.

Cengic, G., & Akesson, K. (2010). On formal analysis of IEC 61499 applications, Part A: Modeling. *Industrial Informatics, IEEE Transactions on*, *6*, 136–144.

Cengic, G., Ljungkrantz, O., & Akesson, K. Formal modeling of Function Block applications running in IEC 61499 execution runtime. Emerging technologies and factory automation, 2006. ETFA '06. *IEEE Conference on*, 2006. pp. 1269–1276.

Cheng, P., & Vyatkin, V. 2008. Automatic model generation of IEC 61499 Function Block using net condition/event systems. *6th IEEE International Conference on Industrial Informatics.*

Chokshi, N., & Mcfarlane, D. C. (2008). *A distributed coordination approach to reconfigurable process control*. Berlin ; London, Berlin ; London: Springer.

Christensen, J.H. 2000. Design patterns for systems engineering with IEC 61499. In: DÖSCHNER, C. (editor) *Verteilte Automatisierung—Modelle und Methoden für Entwurf, Verifikation, Engineering und Instrumentierung.* Magdeburg, Germany.

Clawz (2003). *The Semantics of Simulink Diagrams*. Lemma 1 Ltd.

Dubinin, V., & Vyatkin, V. (2008). On definition of a formal Semantic model for IEC 61499 Function Blocks. *EURASIP Journal of Embedded Systems*, *2008*, 10.

Dubinin, V., Vyatkin, V., & Pfeiffer, T. 2005. Engineering of validatable automation systems based on an extension of UML combined with Function Blocks of IEC 61499. *Proceedings of the 2005 IEEE International Conference on Robotics and Automation.*

Fbench. *FBench Project: Open Tool for IEC 61499 Function Block Engineering* [Online]. FBench Project Team, . Available: http://www.ece.auckland.ac.nz/~vyatkin/fbench/.

Felcht, U. H., Darton, R. C., Prince, R. G. H., & Wood, D. G. (2003). *The future shape of the process industries. Chemical engineering: visions of the world*. Amsterdam: Elsevier Science B.V..

Gerber, C., & Hanisch, H. M. (2010). Does portability of IEC 61499 mean that once programmed control software runs everywhere?. *10th IFAC Workshop on Intelligent Manufacturing Systems* Lisbon, Portugal

Hagge, N., & Wagner, B. 2005. Java code patterns for Petri net based behavioral models. *3rd IEEE International Conference on Industrial Informatics.*

Hirsch, M. 2010. *Systematic Design of Distributed Industrial Manufacturing Control Systems Logos Verlag Berlin*.

Holobloc, I. 2009. *Function Block Development Kit* [Online]. Holobloc inc., http://www.holobloc.com. Available: http://www.holobloc.com.

Hussain, T. & FREY, G. 2006. UML-based Development Process for IEC 61499 with Automatic Test-case Generation. *Emerging Technologies and Factory Automation, 2006. ETFA '06. IEEE Conference on.*

Isagraf. *ICS Triplex ISaGRAF Inc.—leading IEC 61131 and IEC 61499 software* [Online]. Available: http://www.isagraf.com.

Kshemkalyani, A. D., & Singhal, M. (2008). *Distributed computing: principles, algorithms, and systems*. Cambridge University Press.

Lunze, J. (2002). *What is a hybrid system? Lecture notes in control and information science 279: modelling, analysis, and design of hybrid systems*. Berlin Heidelberg: Springer-Verlag.

Mathworks. 2011. *Simulink PLC Coder* [Online]. Available: http://www.mathworks.com/products/sl-plc-coder/.

Mathworks, T. 2010. *The MathWorks—MATLAB and Simulink for Technical Computing* [Online]. Available: http://www.mathworks.com.

Maturana, F., Ambre, R., Staron, R., Carnahan, D., & Loparo, K. 2011. Simulation-based environment for modeling distributed agents for smart grid energy management. *Emerging Technologies & Factory Automation (ETFA), 2011 IEEE 16th Conference on.*

Nxtcontrol.COM. 2010. Available: http://www.nxtcontrol.com/.

Panjaitan, S. D. 2008. *Development Process for Distributed Automation Systems based on Elementary Mechatronic Functions*, Shaker Verlag GmbH, Germany.

Peltola, J., Christensen, J., Sierla, S., & Koskinen, K. 2007. A migration path to IEC 61499 for the batch process industry. *5th IEEE International Conference on Industrial Informatics, 2007* Vienna, Austria.

Profactor. *4DIAC-RTE (FORTE): IEC 61499 Compliant Runtime Environment* [Online]. PROFACTOR Produktionsforschungs GmbH, Available: http://www.fordiac.org.

Ray, R. 2007. *Automated Translation of MATLAB Simulink/Stateflow Models to an Intermediate Format in HyVisual.* M.Sc. Degree, Chennai Mathematical Insitute.

Tata, P., & Vyatkin, V. 2009. Proposing a novel IEC61499 runtime framework implementing the Cyclic Execution semantics. *7th IEEE International Conference on Industrial Informatics (INDIN 2009).* Cardiff UK.

Thramboulidis, K. C. Using UML in control and automation: a model driven approach. Industrial informatics, 2004. INDIN '04. 2004 *2nd IEEE International Conference on*, 2004. pp. 587–593.

Vyatkin, V. 2007a. *IEC 61499 Function Blocks for embedded and distributed control systems design*, Instrumentation Society of America.

Vyatkin, V. 2007b. *IEC 61499 Function Blocks for embedded and distributed control systems design,* Research Triangle Park, NC, ISA-Instrumentation, Systems, and Automation Society.

Vyatkin, V., Hanisch, H. M., Pang, C., & Yang, C.-H. (2009). Closed-loop modeling in future automation system engineering and validation. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, *39*, 17–28.

Vyatkin, V., Hanisch, H. M., & Pfeiffer, T. 2003. Object-oriented modular place/transition formalism for systematic modeling and validation of industrial automation systems. *Industrial Informatics, 2003. INDIN 2003. Proceedings. IEEE International Conference on.*

Yan, J., & Vyatkin V. Distributed Execution and Cyber-Physical Design of Baggage Handling Automation with IEC 61499. *9th International IEEE Conference on Industrial Informatics (INDIN'11)*, 2011 Lisbon, Portugal.

Yang, C.-H., & Vyatkin, V. Model transformation between MATLAB simulink and Function Blocks. *Industrial Informatics (INDIN), 2010 8th IEEE International Conference on,* 13–16 July 2010 2010. pp. 1130–1135.

Yang, C.-H., & Vyatkin, V. 2008. Design and validation of distributed control with decentralized intelligence in process industries: A survey. *6th IEEE International Conference on Industrial Informatics, 2008. INDIN 2008.* Daejeon, Korea.

Yoong, L. H., Roop, P. S., & Salcic, Z. 2009a. Efficient implementation of IEC 61499 Function Blocks. *IEEE International Conference on Industrial Technology (ICIT).* Gippsland.

Yoong, L. H., Roop, P. S., Vyatkin, V., & Salcic, Z. (2009b). A synchronous approach for IEC 61499 Function Block implementation. *IEEE Transactions on Computers*, *58*, 1599–1614.

Zoitl, A. (2009). *Real-Time Execution for IEC*, 61499 ISA.

Please cite this article as: Yang, C., & Vyatkin, V. Transformation of Simulink models to IEC 61499 Function Blocks for verification of distributed control systems. *Control Engineering Practice* (2012), http://dx.doi.org/10.1016/j.conengprac.2012.06.008