Annual Reviews in Control 36 (2012) 338–345

Contents lists available at SciVerse ScienceDirect

Annual Reviews in Control

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

# Transformation of Function Block Diagrams to UPPAAL timed automata for the verification of safety applications

Doaa Soliman [a,*], Kleanthis Thramboulidis [b], Georg Frey [a]

[a] *Chair of Automation, Saarland University, 66123 Saarbrücken, Germany*
[b] *Electrical and Computer Engineering, University of Patras, Greece*

ARTICLE INFO

*Article history:*
Received 16 July 2012
Accepted 15 September 2012
Available online 31 October 2012

*Keywords:*
Function Block Diagram
Timed automata
IEC 61131-3
UPPAAL
Verification
Safety applications
PLCopen

ABSTRACT

Verification of IEC 61131-3 based safety applications is a challenge in the industrial automation domain. In this paper, the transformation of FBD diagrams to UPPAAL formal models was adopted to address this challenge. A set of transformation rules are defined for the automatic transformation of IEC 61131-3 Function Block based safety applications to UPPAAL timed automata models. These models are next used for the verification of the safety application. Both the source and the target domain models have been formally defined and these definitions are used for the definition of the transformation rules. Based on this a prototype model transformer was developed using Java. The transformer was used with various safety applications to check the efficiency of the transformation process. A laboratory system is presented as a case study to highlight the proposed approach.

## 1. Introduction

The IEC 61131-3 (IEC, 2003) set of programming languages is widely adopted by industry and the majority of industrial automation systems are based on these languages. Many of these systems are safety critical and should conform to various safety standards defined by international standard organizations, such as ISO and IEC. For example, the IEC 61508 (IEC, 1998) and its implementation for the machinery sector, i.e., the IEC 62061 (IEC, 2000), impose manufacturing industries to certify that their systems are safe for the human life and the environment. This is why safety issues of IEC 61131 have already been examined by the research community, e.g., Toon (2002) and Lewis (2002). Moreover, PLCopen (http://plcopen.org) has developed a library of Safety Function Blocks (SFBs) to facilitate the development of safety applications. Safety applications that are developed using verified libraries, such as the one of PLCopen, are suitable to meet the requirements of the second upper Safety Integrity Level, i.e., SIL 3, as defined by IEC 61508 (IEC, 1998).

Among the challenges that engineers face, during the development process of safety critical industrial systems, is the verification of the safety application before implementing it. Over recent years the complexity of safety applications has increased dramatically. This makes the formalization of safety applications a very difficult task. Formalization is important in order to verify that the design meets the specified safety requirements. There are several research works that try to facilitate the transformation of PLC code into formal models supported by available model checker tools. UPPAAL (Larsen et al., 1997; Behrmann et al., 2004) was selected, in this work, to be used for the verification process of safety applications based on Function Block Diagrams (FBDs). UPPAAL is a good choice for formal verification of systems that can be modeled as a collection of non-deterministic processes with real valued clocks. This makes the tool suitable for the verification of safety applications built from Function Blocks (FBs) triggered by timers.

In this paper, we have adopted as format of the IEC 61131-3 FBD the PLCopen XML one, defined by PLCopen (2009). This allows our approach to be used by any IEC 61131-3 commercial development tool that supports the PLCopen XML specification. The FBD design models of the safety application are transformed to UPPAAL models, which are next imported into the UPPAAL model checker. In Soliman and Frey (2009, 2011) the validation via simulation and the verification via model checking based on these models, is presented. In this paper, we formally define the models of the two domains. A set of formal definitions for the IEC 61131-3 FBD and the UPPAAL system are given. Then, based on these models we formally define the set of transformation rules. A set of mapping rules for the transformation process based on the meta-models of the source and target domains have been informally defined in Thramboulidis et al. (2011). A prototype model transformer was developed using the Java language and its behavior was checked

* Corresponding author.
*E-mail address:* doaa.soliman@aut.uni-saarland.de (D. Soliman).

on several safety applications. An example safety application for an *XY* coordinated table is used through the paper to illustrate the proposed transformation and verification process. This kind of precision-controlled automated movement system is broadly used in mechanical processes and applications including material handling and pharmaceutical.

The remainder of this paper is organized as follows: In Section 2, related work is discussed. Section 3 briefly describes the proposed semi-automated verification process. In Section 4, the formal models of the source and target domains are given and based on these the transformation rules are defined. In Section 5, the whole process of developing and verifying the example safety application is presented, to demonstrate the proposed approach. Summary and future work are given in Section 6.

## 2. Related work

Nowadays, there is an increasing interest of automating the verification process of applications in the industrial automation domain. Several research groups have already published the results of their work to this direction. These works are mainly based on the transformation of the FBD models of the safety application to formal models, which are next used for verification purposes exploiting model checking tools. For example, Pavlovic' and Ehrich (2010) present an approach for the automated formal verification of PLC software, which is written in FBD, using the NUSMV model checker. The FBD should be constructed from basic FBs such as bit operations, comparators and jumps. The IL representation of the FB body, which is generated from the corresponding FBD with the help of the PLC programming tool, is transformed to a NUSMV model. This approach cannot be used for the verification of safety applications built up from PLCopen SFBs, since SFBs are not basic FBs.

Yoo et al. (2008) propose a way to formally verify FBD programs. They adopt an IEC 61131-3 compliant syntax for the FBD and propose the translation of this specification into Verilog models. The so generated Verilog models are verified using the Cadence SMV model checker. The FBD2V tool has been implemented to automatically perform this transformation. A case study is used to show the effectiveness of the proposed approach. As in Pavlovic' and Ehrich (2010) the transformation approach is applied only for systems composed finally of basic FBs. SFBs of the safety library of PLCopen are not supported.

A formal verification approach of a safety procedure of a nuclear power plant written in FBD language is presented by Németh and Bartha (2009). The approach is based on the Colored Petri Net (CPN) representation and it is similar to our approach in that it uses a pre-developed library of CPN subnets to represent the FBD specification in an identical structural way. However, an automatic transformation process is not supported; the transformation of FBD diagrams to CPN is carried out manually.

Silva et al. (2008) describe the automatic generation of timed automata (TA) models from FBD models. This allows testing of FBDs against their specifications in the case that there is no PLC available to run them. The UPPAAL TRON tool is used to test simulation scenarios on the generated TA models. The FBD elements, that this approach deals with, are logical AND, OR, SR/RS flip-flops and time elements TON, TOFF and TP. The test of the model is performed by feeding some simulation traces to UPPAAL TRON tool. Traces are automatically generated from cause/effect matrix specified by experts. This approach can be considered similar to the one presented in this paper, but it is considerably different if we consider the details. Moreover, this approach cannot be applied to the safety applications considered in our approach, since it only supports logical and time FBs.

Wardana et al. (2009) present an automatic verification approach to ensure the correctness of the continuous function chart (CFC) programs using UPPAAL model checker. A drawback of this approach is the absence of standard for CFC, which leads to different implementations by vendors. The formal definition of CFC adopted by the Emerson DeltaV tool is used in this work. In a similar way with our approach, transformation rules are formalized based on formal definitions of both CFC and UPPAAL TA. However, this approach cannot be generalized to other implementation tools since it is not based on standards.

We are not aware of any other approach that automatically transforms IEC 61131-3 FBD specifications to formal models for their automatic verification as part of a systematic approach for upgrading legacy industrial systems to meet safety regulations defined by standards.

## 3. Automating the verification process

The design model of the safety application has to be verified before integrating the safety application in the safety critical industrial system. UPPAAL was selected to be used for the verification process of the safety application. This means that the FBD design models of the safety application have to be transformed to UPPAAL models. In this section, a brief description of our proposal to semi-automate this verification process is given. A detailed description is given in Thramboulidis et al. (2011).

We have adopted as notation to express the design model of the safety application the widely used in the industrial automation domain IEC 61131-3 FBD programming language. This decision allows the proposed transformation process to be used with any PLC development tool that is compliant with the IEC 61131-3 standard. Moreover, it provides support for the use of the certified Safety Function Block (SFB) library of PLCopen, and for model exchange between commercial tools that support the PLCopen XML specification. The basic idea of the transformation process is depicted in Fig. 1. SFBs of the PLCopen safety library have been transformed to corresponding UPPALL TA that were used to form the safety UPPAAL library. For every SFB, a TA model was graphically constructed via the UPPAAL editor; it was validated via simulator and finally verified against formal properties via the verifier (Soliman and Frey, 2009, 2011). This has simplified the building of the required model-to-model transformer that will transform the FBD design model of the safety application to UPPAAL TA model. The model-to-model transformer uses as source information the design model of the safety application, which will be exported in PLCopen XML format, and automatically generates the equivalent UPPAAL model in XML format. The result of the

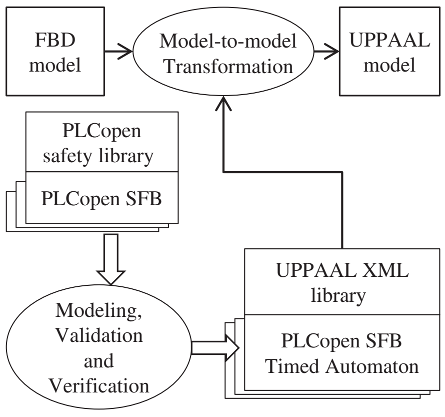

FBD model
Model-to-model Transformation
UPPAAL model
PLCopen safety library
PLCopen SFB
Modeling, Validation and Verification
UPPAAL XML library
PLCopen SFB Timed Automaton

**Fig. 1.** The FBD to UPPAAL transformation process.

transformation process is imported into the UPPAAL model checker for verification against the requirements of the safety application. Information on the unsatisfied properties is forwarded to the simulator to identify the TA state in which the properties are not satisfied. This helps the developer to identify the source of the problem and fix it.

## 4. Formal definition of transformation rules

In order to formally define the transformation rules we first formally define the models of the two domains, i.e., the Function Block Diagram domain and the UPPAAL TA domain. Then, based on these two formal models, we define the transformation rules.

### *4.1. Function Block Diagram (FBD)*

The FBD language, which has its origins in the field of signal processing, has introduced very early in the industrial automation domain basic concepts of the model-driven development paradigm (Thramboulidis, 2010). The standard defines that zero or more function block instances and other language elements can be used for the definition of the behavior captured by an IEC 61131-3 program. However, we consider in the context of this work only programs that their behavior is expressed using Function Block instances. Moreover, we use only two predefined libraries for the development of safety applications. The first one is the SFB library of PLCopen (2006) defined specifically for safety applications; it contains 20 SFB types. This library is widely accepted and is already supported by many commercial PLC programming tools. The second is a library of Logic gate Function Blocks (LFBs), such as AND, OR, XOR and NOT. PLCopen (2008) has already presented safety applications built up from both libraries. These assumptions are not restrictive to implement safety applications (PLCopen, 2006). The new SFBs introduced this year by PLCopen will be supported by the next version of the transformation tool.

According to IEC 61131-3, the FB is defined as an independent, encapsulated data structure with a defined algorithm working on this data. The algorithm is represented by the body of the FB type. Every FB: (a) has a name, (b) defines the interface of its FB instances by a set of input parameters and a set of output parameters, and (c) defines the behavior of its instances by the body of the FB type. Programs, as defined by the standard, can only be instantiated within resources, while Function Blocks can only be instantiated within programs or other Function Blocks. The objective of a program is to capture the signal processing required for the control of a machine or a process by a programmable controller system. In this section, we formally define programs defined by the FBD language, i.e., FBD programs, with the objective to use this definition for the transformation process of the FBD representation of the safety application to a UPPAAL TA representation. The FB network shown in Fig. 2 is part of the safety application developed for the *XY* coordinated table and is used in the following as an example. This case study is presented in detail in Section 5.

An FBD program consists of variable declarations and one or several FB networks. Every network is mainly built up from SFB and LFB instances connected to each other via five types of connections. For the transformation process, LFBs are considered logic operators since they have simple internal behavior.

**Definition 1** (*FBDProgram*). An FBD program is defined as a tuple FBDProgram = ⟨Name, IP, OP, IOP, VL, VG, VE, VA, Body⟩, where:

Name: the name of the FBDProgram that is defined by its developer,
IP: a set of input variables that represent monitored parameters of the controlled machine or process, $\{ip_1, ip_2, \ldots, ip_n\}$,
OP: a set of output variables that represent controlled parameters of the machine or process, $\{op_1, op_2, \ldots, op_m\}$,
IOP: a set of input/output variables,
VL: a set of local variables of this FBDProgram,
VG: a set of global variables defined by the FBDProgram,
VE: a set of global variables of other POUs used by this FBDProgram,
VA: a set of access variables used by the FBDProgram as named variables; these can be accessed by some of the communication services specified in part 5 of IEC61131,
Body: it is defined as a set of FBNs which are interconnected using the local variables (VL) of the FBDProgram. IP, OP and IOP of the FBDProgram are used as inputs, outputs and input–outputs of the FBNs. The body defines the behavior of the FBD program instances.

For example, for the Declaration part of FBD program body of Fig. 2:

VL = {Reset, SafeStandstill_M1,_M2},
IP = {SI_1_0, SI_1_2, ..., SI_3_2, SI_3_4, SI_3_6},
OP = {SO_1_1, SQ_1_3, SQ_1_7},

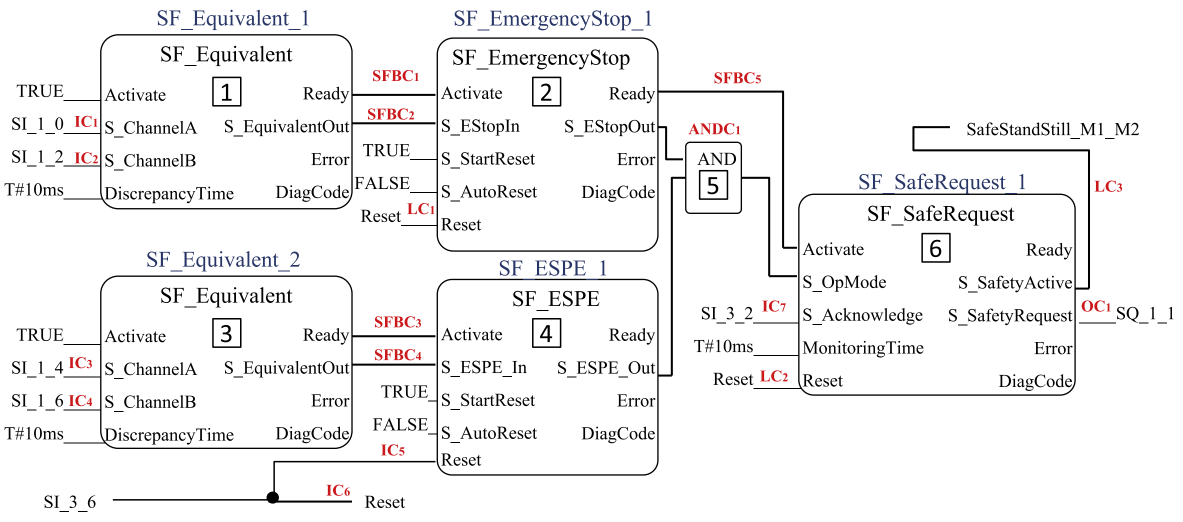


**Fig. 2.** The example FB network.

**Definition 2** (*FBN*). A Function Block Network is defined as a tuple FBN = ⟨FBIs, C⟩, where:

FBIs: a set of FB instances (see Definition 3),
C: a set of connections (see Definition 5).

**Definition 3** (*FBI*). A Function Block Instance is defined as a tuple FBI = ⟨InstanceName, FBTypeName, EO⟩, where:

InstanceName: user defined name of the specific instance,
FBTypeName: the name of the corresponding FB type,
EO: it is an integer that defines the execution order of the FB instance in the context of the FBN. The execution order usually appears in the middle of every instance FB, as shown in Fig. 2.

**Definition 4** (*FBT*). A Function Block Type is defined as a tuple FBT = ⟨FBTypeName, IP, OP, IOP, VL, VE, Body⟩, where:

FBTypeName: the name of the Function Block Type defined by the developer,
IP: a set of input parameters, $\{ip_1, ip_2, \ldots, ip_n\}$,
OP: a set of output parameters, $\{op_1, op_2, \ldots, op_m\}$,
IOP: a set of input/output parameters,
VL: the set of local variables of this FBT,
VE: the set of global variables of other POUs used by this FBT,
Body: The body can be written in one of the five programming languages defined by IEC 61131-3. When FBD is used the body is defined as a set of FBNs which are interconnected using the local variables (VL) of the FBT. IP, OP and IOP of the FBT are used as inputs, outputs and input-outputs of the FBNs. The body defines the behavior of the instances of the FBT.

**Definition 5** (*C*). Connections is defined as C = {SFBC, IC, LC, LogicC, OC}, where: **SFBC** is a set of direct connections between OPs of SFB instances and IPs of SFB instances. It is defined as:

SFBC$_n$: InstanceName$_i$.op$_j$ → InstanceName$_k$.ip$_l$,
e.g., *SFBC*$_5$:*SF_EmergencyStop_*1.*Ready* → *SF_SafetyRequest_*1.*Activate*.

**IC** is a set of connections that connects input variables of the FBD program, i.e., elements of IP, to other variables or parameters. The input variable ip$_j$ ∈ IP of FBDProgram may be connected to an input parameter ip$_l$ ∈ IP of the *k*th FB instance or to a local variable $vl_k$ ∈ VL. It can be defined as:

IC$_n$: FBDProgram.ip$_j$ → InstanceName$_k$.ip$_l$, or
IC$_n$: FBDProgram.ip$_j$ → FBDProgram.vl$_k$,
e.g.: *IC*4: *SI_*1_6 → *SF_Equivalent_*2.*S_ChannelB*,
*IC*6: *SI_*3_6 → *Reset*.

**LC** is a set of connections that connects local variables in VL of FBD program that do not appear in IC, to other variables or parameters. The local variables vl$_i$ ∈ VL of FBDProgram may be connected to an input port ip$_j$ ∈ IP of the *k*th FB instance, an output port op$_l$ ∈ OP of the *k*th FB instance or an output variable vbo$_m$ ∈ OP of FBDProgram. It can be defined as:

LC$_n$: FBDProgram.vl$_i$ → InstanceName$_k$.ip$_j$, or
LC$_n$: InstanceName$_k$.op$_l$ → FBDProgram.vl$_i$, or
LC$_n$: FBDProgram.vl$_i$ → FBDProgram.op$_m$,
e.g.: *LC*2: *Reset* → *SF_SafetyRequest_*1.*Reset*,
*LC*3: *SF_SafetyRequest_*1.*S_SafetyActive* → *SafeStandstill_M*1*_M*.

**LogicC** is a set of indirect connections between SFBs via one or more LFBs. It is defined as:

$$\begin{aligned}LogicC_n :\ & f(\text{InstanceName}_i.\text{op}_j,\ \text{FBDProgram.ip}_k,\\ & \text{FBDProgram.vl}_l, \text{Const}_m)\\ & \rightarrow \text{InstanceName}_o.\text{ip}_p/\text{FBDProgram.op}_q/\text{FBDProgram.vl}_r\end{aligned}$$

where *f* is a logic function expression involving AND, OR, XOR and Not. The "/" symbol means "and/or", e.g.,

$$\begin{aligned}ANDC_1 :\ & SF\_EmergencyStop\_1.S\_EStopOut \& SF\_ESPE\_1.S\_ESPE\_Out\\ & \rightarrow SF\_SafetyRequest\_1.S\_OpMode.\end{aligned}$$

**OC** is a set of output connections between op$_i$ ∈ OP of the *k*th FB instance and output variables op$_j$ ∈ OP of FBDProgram. It is defined as:

OC$_n$: InstanceName$_k$.op$_i$ → FBDProgram.op$_j$, e.g.:
*OC*$_1$: *SF_SafeRequest_*1.*S_SafetyRequest* → *SQ_*1_1.

### 4.2. The UPPAAL TA system

The UPPAAL system consists of three main parts which are: declarations, TA models and system declarations. TA models are separately created and are synchronized via shared variables. All variables are globally declared in the declarations part, to allow the synchronization of TA models. The execution of these models is specified in the system declarations part. The UPPAAL TA system presented in this paper is formalized based on the work of Alur and Dill (1994). In the following the UPPAAL system is defined to facilitate the transformation process. This means that all parts, which are not used by the transformation process, are not included in the definition.

**Definition 6** (*UPPAALSys*). A UPPAAL system can be defined as a tuple UPPAALSys = ⟨Declarations, TAModels, SystemDeclarations⟩ where:

**Declarations** is a set of triples ⟨type, variableName, value⟩where:
*type* can be bool, int, const bool, const int, or clock,
*variableName* represents the name of the variable, and*value* is considered either the initial value or the constant according to the type parameter.

The declaration part contains all input/output parameters (IP, OP) of all SFB instances and all input/output/local variables (IP, OP, VL) of the FBD body. Fig. 3 shows part of the Declarations of our safety application example. The left part of figure declares the variables used within the SF_Equivalent_1 SFB instance. The right part declares input, output and local variables used within the FBD body.

**TAModels** is a set of TA models that constitute the UPPAAL system. The TA model is defined as a tuple TAModel = (TAName, TA) where:

TAName is the name of the TA model which appears in the UPPAAL project tree and in the SystemDeclarations part to arrange priorities on TAModels, and

```
//SF_Equivalent_1                          bool SI_1_0;
// VAR_INPUT                               bool SI_1_2;
const bool Activate_Eq_1=1;                .
bool S_ChannelA_Eq_1;                      .
bool S_ChannelB_Eq_1;                      bool SQ_1_3;
const int DiscrepancyTime_Eq_1=10;         bool SQ_1_7;
clock DT_Eq_1;                             bool Reset;
//VAR_OUTPUT                               bool Automatic;
bool Ready_Eq_1;                           bool Manual;
bool S_EquivalentOut_Eq_1;                 bool DoorsLocked;
bool Error_Eq_1;                           bool SafeStandStill_M1_M2;
int DiagCode_Eq_1;                         bool SLS_M1_M2; 0
```

**Fig. 3.** Declarations part in the UPPAAL system (part).

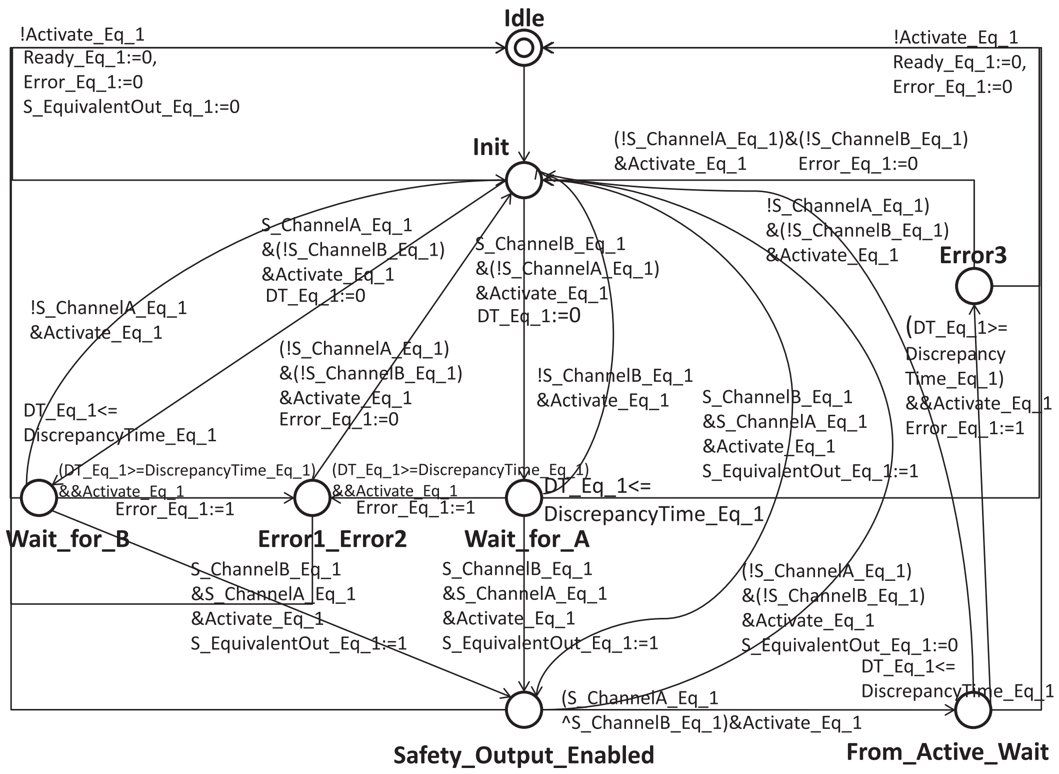


**Fig. 4.** The TA resulted from the SF_Equivalent_1 FB instance.

TA is the state-transition graphical description of TAModel.

Fig. 4 presents the UPPAAL TA of the SF_Equivalent_1 SFB instance shown in Fig. 2 which is defined as $(L, l_0, E, V, C, Init, Inv, T_L)$, where:

$L$ = {Idle, *Init*, Wait_for_A, Error1_Error2, Wait_for_B, Error3, From_Active_Wait, Safety_Output_Enabled},
$L_0$ = {Idle},
$E = \{E1, E2, \ldots\}$. An example of an edge is $E1$ = {Idle, Activate_Eq_1, –, Ready_Eq_1 = 1, Init},
$V$={Activate_Eq_1, S_ChannelA_Eq_1, S_ChannelB_Eq_1, DiscrepancyTime_Eq_1, Ready_Eq_1, S_EquivalentOut_Eq_1, Error_Eq_1},
$C$ = {DT_Eq_1},
*Init* = {Ready_Eq_1: = 0, S_EquivalentOut_Eq_1: = 0, Error_Eq_Eq_1: = 0},
*Inv*({Wait_for_A}) = *Inv*({Wait_for_B}) = Inv({From_Active_Wait}) = DT_Eq_1 < =DiscrepancyTime_Eq_1,
$T_L$ for all locations is {o}.

There are four types of TAModels: (a) TA corresponding to SFB instances; we call these SFB TA, (b) Connections TA, (c) R_TRIG/F_TRIG TA, and (d) INPUTs TA. A SFB TA captures the internal behavior of an SFB instance. Connections TA capture all direct/indirect connections in FBNs (C). R_TRIG/F_TRIG TA are associated to SFBs IPs. Their function is to reset rising/falling edges detectors after every execution cycle. INPUTs TA are to allow altering external input variables (FBDProgram.IP) which are physically connected to input modules of safety PLC. Fig. 5 shows examples of TA models. All TA models, unless SFB TA, are composed of a single location and a single self-edge.

**SystemDeclarations** in which execution order is defined by assigning priorities to TA models. In the example of Fig. 2, input connection TA (ICs) has the highest priority, followed by FBs and other connection types according to defined execution order and finally output connection TA (OCs). A complete definition and the semantics of UPPAAL TA network are given in Soliman and Frey (2011).

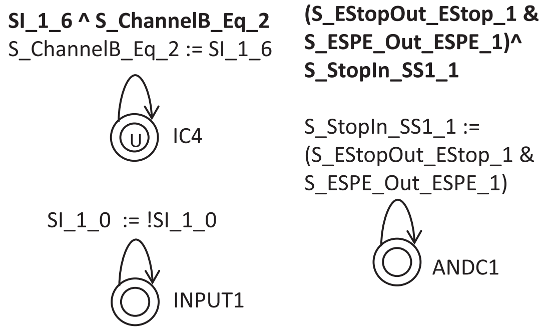


**Fig. 5.** UPPAAL TA models graphical representation.

## 4.3. Transformation rules

In this sub-section the transformation rules that are based on the above defined formal models of the FBD and UPPAAL systems are given.

*Transformation Rule 1 (Declarations mapping Rule)*: The objective of this rule is to transform the variables declared in the FBD program and the input and output parameters of each SFB instance to UPPAAL artefacts. It is composed of two parts:

1. For each FBI = ⟨InstanceName, FBTypeName, EO⟩where FBT = ⟨FBTypeName, IP, OP, IOP, VL, VE, Body⟩:

Insert InstanceName$_i$, IP_i, OP_i in Declarations of UPPAALSys, $i = 1, 2, \ldots, n$, where $n$ is the number of FBIs that have similar

FBTypeName (i.e., SF_Equivalent_1, SF_Equivalent_2 in Fig. 2), taking into account constants associated to subset of IP_i, as shown in the left side of Fig. 3.

2. For each FBDProgram.ip$_m \in$ IP, FBDProgram.op$_n \in$ OP and FBDProgram.vl$_p \in$ VL,

Insert corresponding vbi$_m$, vbo$_n$ and vbl$_p$ in Declarations (see right side of Fig. 3).

*Transformation Rule 2 (FBIs mapping Rule)*: The objective of this rule is to transform FBIs to UPPAAL TA models.

For each SFB instance = ⟨InstanceName, FBTypeName, EO⟩,

Insert the corresponding TA model to UPPAAL with TA model name: = InstanceName using a pre-defined SFB TA model library (see Fig. 4).

*Transformation Rule 3 (Connections mapping Rule)*: The objective of this rule is to transform connections. For each connection in $C$ = {SFBC, IC, LC, LogicC, OC}, a TA model in UPPAAL is inserted. This rule is composed of eight parts:

1. For each SFBC$_n$: InstanceName$_i$.op$_j$ → InstanceName$_k$.ip$_l$,
   Insert a TA model with name $C_n$. This TA has one initial urgent location and self-edge where: $L = L_0 = \{C_n\}$, $E = \{E_1\} = \{C_n,\ op_j \wedge\ ip_l,\ -\ ,\ ip_l := op_j,\ C_n\}$, where is for XOR, $V = \{ip_l, op_j\}$, $C = \{\}$, $Init = \{\}$, No invariant, and $T_L = \{u\}$.
2. For each IC$_n$: FBDProgram.ip$_j$ → InstanceName$_k$.ip$_l$,
   Insert a TA model with name IC$_n$ where: L=L$_0$ = {IC$_n$}, $E = \{E_1\}$ = {IC$_n$, $ip_j \wedge$ ip$_l$, – , $ip_l := ip_j$, IC$_n$}, $V = \{ip_j, ip_l\}$, $C = \{\}$, Init={}, No invariant, and $T_L$ = {u}. (see IC4 in Fig. 5),
3. For each IC$_n$: FBDProgram.ip$_j$ → FBDProgram.vl$_k$,
   Insert a TA model with name IC$_n$ where: $L = L_0$ = {IC$_n$}, $E = \{E_1\}$ = {IC$_n$, $ip_j \wedge vl_k$, – , $vl_k := ip_j$, IC$_n$}, $V$ = {ip$_j$, $vl_k$}, $C = \{\}$, $Init = \{\}$, No invariant, and $T_L$ = {u}.
4. For each LC$_n$:
   FBDProgram.vl$_i$ → InstanceName$_k$.ip$_j$,
   Insert a TA model with name $C_n$ where: $L = L_0 = \{C_n\}$, $E = \{E_1\} = \{C_n,\ vl_i \wedge ip_j,\ -\ ,\ ip_j := vl_i,\ C_n\}$, $V = \{vl_i, ip_j\}$, $C = \{\}$, $Init = \{\}$, No invariant, and $T_L = \{u\}$.
5. For each LC$_n$: InstanceName$_k$.op$_l$ → FBDProgram.vl$_i$,
   Insert a TA model with name $C_n$ where: $L = L_0 = \{C_n\}$, $E = \{E_1\} = \{C_n,\ op_l \wedge vl_i,\ -\ ,\ vl_i := op_l,\ C_n\}$, $V = \{op_l, vl_i\}$, $C = \{\}$, $Init = \{\}$, No invariant, and $T_L = \{u\}$.
6. For each LC$_n$: FBDProgram.vl$_i$ → FBDProgram.op$_m$,
   Insert a TA model with name $C_n$, where: $L = L_0 = \{C_n\}$, $E = \{E_1\} = \{C_n,\ vl_i \wedge op_m,\ -\ ,\ op_m := vl_i,\ C_n\}$, $V = \{vl_i, op_m\}$, $C = \{\}$, $Init = \{\}$, No invariant, and $T_L = \{u\}$.
7. For each LogicC$_n$:$f$(InstanceName$_i$.op$_j$, FBDProgram.ip$_k$, FBDProgram.vl$_l$, Const$_m$) → InstanceName$_o$.ip$_p$/FBDProgram.op$_q$/FBDProgram.vl$_r$,
   Insert a TA model, where: $L$ = L$_0$ = {LogicC$_n$}, $E = \{E_1\}$ = {LogicC$_n$, $f(op_j,\ ip_k,\ vl_l,$ Const$_m) \wedge\ ip_p/op_q/vl_r$, – , $ip_p/op_q/vl_r := f$(op$_j$, $ip_k$, $vl_l$, Const$_m$), LogicC$_n$}, $V$ = {$op_j$, $ip_k$, $vl_l$, Const$_m$, $ip_p/op_q/vl_r$}, $C = \{\}$, $Init = \{\}$, No invariant, and $T_L = \{o\}$ (see ANDC1 in Fig. 5)
8. For each OC$_n$: InstanceName$_k$.op$_i$ → FBDProgram.op$_j$,
   Insert a TA model, where: $L = L_0$ = {OC$_n$}, $E = \{E_1\}$= {OC$_n$, $op_i \wedge\ op_j$, – , $op_j := op_i$, OC$_n$}, $V = \{op_i, op_j\}$, $C = \{\}$, $Init = \{\}$, No invariant, and $T_L = \{u\}$.

*Transformation Rule 4 (rising/falling edge triggers mapping rule)*: There is a set of input parameters FBT.ip$_i$ that need to reset an R_TRIG/F_TRIG signal associated to them. For example, the rising trigger edge at Reset input of SF_SafetyRequest, which is called R_TRIGatReset, is a pulse signal that goes to true when Reset is true and immediately reset to false after executing the SFB. These rising/falling signals have formal names of type R_TRIGatip$_i$/FTRIGatip$_i$, specified by PLCopen 2006. Their names are transformed to R_TRIGip$_i$_TypeName_n/F_TRIGip$_i$_TypeName_n in the UPPAAL system, e.g., R_TRIGReset_SR_1, where TypeName represents a SFB type and n represents the SFB instance. This rule is composed of two parts:

1. For each R_TRIGat_ip$_i$ related to FBI, Insert a TA model with name R_TRIGip$_i$_FBTypeName_n , where: $L = L_0$ = {R_TRIG}, $E = \{E_1\}$ = {–, R_TRIGip$_i$_n, –, R_TRIGip$_i$_n := 0, –}, $V$ = {R_TRIGip$_i$_n}, $C = \{\}$, $Init = \{\}$, No invariant, and $T_L = \{u\}$.
2. For each F_TRIGat_ip$_i$, Insert a TA model with name F_TRIGip$_i$_FBTypeName_n, where $L = L_0$ = {F_TRIG}, $E = \{E_1\}$ = {–, F_TRIGip$_i$_n, –, F_TRIGip$_i$_n:= 0, –}, $V$ = {F_TRIGip$_i$_n}, $C = \{\}$, $Init = \{\}$, No invariant, and $T_L = \{u\}$.

*Transformation Rule 5 (INPUTs mapping rule)*: The objective of this rule is to allow manual validation against simulation scenarios using the UPPAAL simulator. For this, external variables IP of the FBD program are allowed to be changed by the user.

For each FBDProgram.ip$_m$ in IP,

Insert TA model with name INPUT$_n$, where: $L = L_0$ = {–}, $E = \{E_1\}$ = {–, –, –, $ip_m$ :=! $ip_m$, –}, $V = \{ip_m\}$, $C = \{\}$, $Init = \{\}$, No invariant, and $T_L = \{o\}$ (INPUT1 in Fig. 5)

*Transformation Rule 6 (execution flow mapping rule)*: The objective of this rule is to define the execution flow of the UPPAAL model using priorities on TA models.

For each FBI

Assign priority to the corresponding TA model in UPPAAL SystemDeclarations part, based on EO defined in FBI = ⟨InstanceName, FBTypeName, EO⟩.

## 5. Case study

The design, implementation and verification of safety applications can be considered as part of a methodology to upgrade legacy systems to conform to safety standards. The *XY* coordinated table of the Automation laboratory at Saarland University, shown in Fig. 6 is used as a case study in this paper. This type of precision-controlled automated movement is broadly used in mechanical processes and applications including material handling and pharmaceutical. The system consists of an automatically moved *XY* drawing table with manually adjustable marking pen mounted above an Aluminum base. The two axes are operated with Siemens synchronous motors and precision linear guides from Bosch Rexroth. A soft-PLC WinLC-T from Siemens equipped with S7-technology is used and the motors are controlled using Sinamics S120 servo drives. The user interface is based on Wincc HMI that allows the operator to perform the required motion tasks through simple graphical screens. The control system equipments, i.e., PLC and Servo drivers, are connected through Profibus DP. For safety reasons the system is protected by a glassy cell with two doors on the

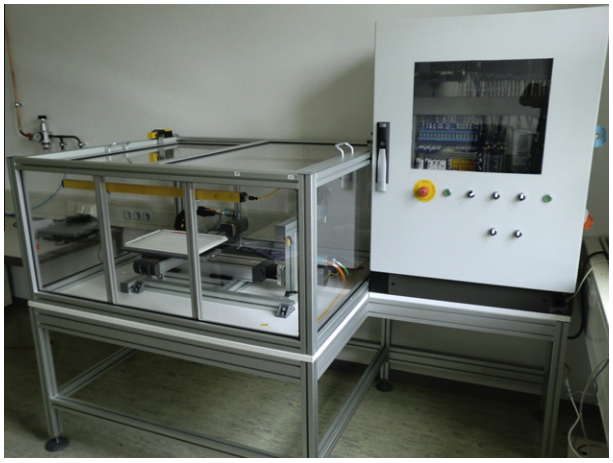


**Fig. 6.** XY table system after upgrading.

top. The methodology for upgrading legacy systems is described in (Soliman et al., 2011). This section focuses on the validation of the UPPAAL TA model via simulation and the examination of specific properties of the system.

The area surrounded by the glass cell is considered a dangerous area for the operator. The access to the working area is restricted by two monitored guard doors with interlock and a light curtain. The operator is allowed to access the working area only in the manual mode of operation. In the automatic mode of operation, it is assumed that the operator needs not to access the working area. For this reason an unlock request to unlock doors is neglected by the safety system. In an emergency situation the motors need to safely stop in accordance with stop category 1. In this case study, the safety drive system supports the modes safe stop and safe motion.

The design of the safety application was performed in FBD language using the MultiProg PLC editing tool from KW software. The FBD program body consists of three FB networks (FBNs), twelve input variables (IP), three output variables (OP) and four local variables (VL). The FB networks of this program contain, eight SFB, five LFB instances (FBIs) and thirty two connections (C).

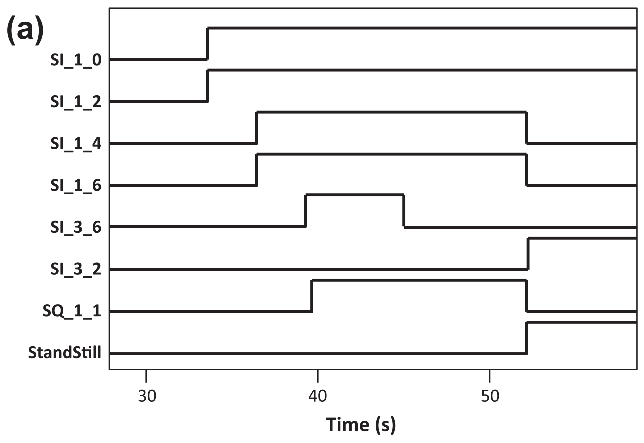
The FBD program was exported in PLCopen XML format. The FBD to Timed Automata (FBD2TA) prototype transformation tool was used to automatically transform the FBD design model of the safety application into a UPPAAL TA model. The result output file can be loaded by the UPPAAL editor.

### 5.1. Validation of the UPPAAL system

#### 5.1.1. Extracting simulation scenarios with the help of the PLC tool

Using simulation we record the response of the system in terms of output variables and local variables, as a function of system inputs. The Logic Analyzer software tool supported by MultiProg was used to extract the timing diagrams of different simulation scenarios of the FBD safety application. The external input variables defined in the FBD network of Fig. 2 are used to extract the timing diagrams shown in Fig. 7.

The two timing diagrams of Fig. 7, capture the output SQ_1_1 (safe stop request) and variable SafeStandStill_M1_M2 (safe stop achievement) as a function of the inputs SI_1_0, SI_1_2 (contacts of emergency stop), SI_1_4, SI_1_6 (contacts of light curtain), SI_3_6 (contact of Reset) and SI_3_2 (safe stop acknowledgment). In the first simulation scenario, the discrepancy time between emergency stop switch contacts and light curtain contacts is equal to zero (Fig. 7a). Safe stop acknowledgment is received after safe stop request within the specified monitoring time. In the second simulation scenario, the safe stop acknowledge signal is received after the expiration of the desired monitoring time (Fig. 7b).



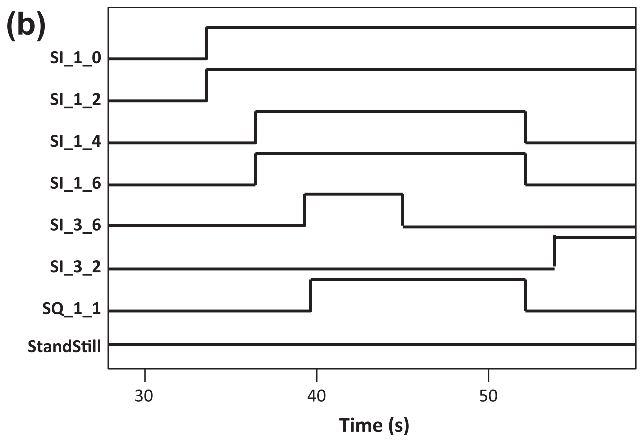


**Fig. 7.** Timing diagrams corresponding to two different simulation scenarios.

#### 5.1.2. Converting simulation scenarios to UPPAAL TA models

In this step, each simulation scenario is manually transformed to UPPAAL TA model. Fig. 8 presents the UPPAAL TA models of the two simulation scenarios for our case study. Then, they are imported to the UPPAAL TA system that was constructed from the transformer using as input the FBD program of the safety

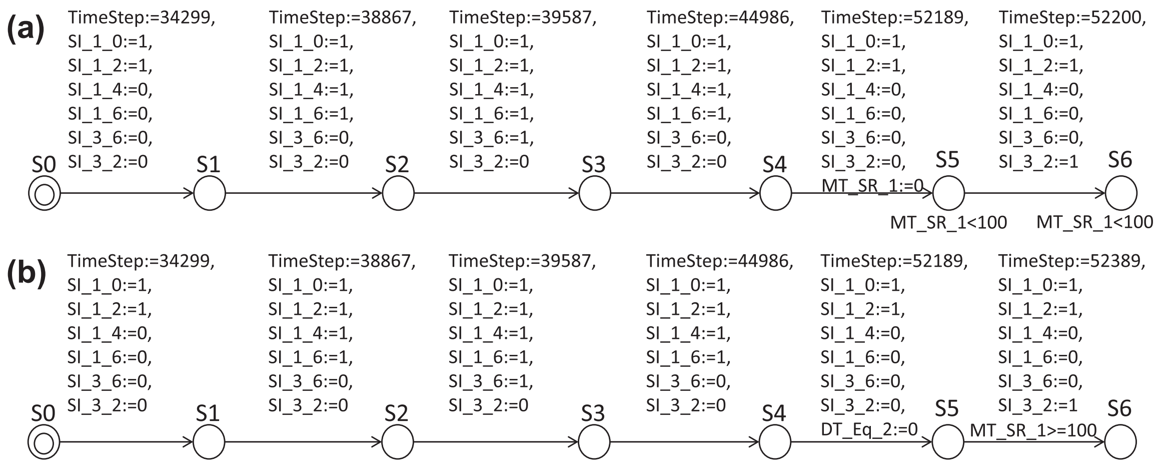

(a)
S0
TimeStep:=34299, SI_1_0:=1, SI_1_2:=1, SI_1_4:=0, SI_1_6:=0, SI_3_6:=0, SI_3_2:=0
S1
TimeStep:=38867, SI_1_0:=1, SI_1_2:=1, SI_1_4:=1, SI_1_6:=1, SI_3_6:=0, SI_3_2:=0
S2
TimeStep:=39587, SI_1_0:=1, SI_1_2:=1, SI_1_4:=1, SI_1_6:=1, SI_3_6:=1, SI_3_2:=0
S3
TimeStep:=44986, SI_1_0:=1, SI_1_2:=1, SI_1_4:=1, SI_1_6:=1, SI_3_6:=0, SI_3_2:=0
S4
TimeStep:=52189, SI_1_0:=1, SI_1_2:=1, SI_1_4:=0, SI_1_6:=0, SI_3_6:=0, SI_3_2:=0, MT_SR_1:=0
S5
MT_SR_1<100
TimeStep:=52200, SI_1_0:=1, SI_1_2:=1, SI_1_4:=0, SI_1_6:=0, SI_3_6:=0, SI_3_2:=1
S6
MT_SR_1<100
(b)
S0
TimeStep:=34299, SI_1_0:=1, SI_1_2:=1, SI_1_4:=0, SI_1_6:=0, SI_3_6:=0, SI_3_2:=0
S1
TimeStep:=38867, SI_1_0:=1, SI_1_2:=1, SI_1_4:=1, SI_1_6:=1, SI_3_6:=0, SI_3_2:=0
S2
TimeStep:=39587, SI_1_0:=1, SI_1_2:=1, SI_1_4:=1, SI_1_6:=1, SI_3_6:=1, SI_3_2:=0
S3
TimeStep:=44986, SI_1_0:=1, SI_1_2:=1, SI_1_4:=1, SI_1_6:=1, SI_3_6:=0, SI_3_2:=0
S4
TimeStep:=52189, SI_1_0:=1, SI_1_2:=1, SI_1_4:=0, SI_1_6:=0, SI_3_6:=0, SI_3_2:=0, DT_Eq_2:=0
S5
TimeStep:=52389, SI_1_0:=1, SI_1_2:=1, SI_1_4:=0, SI_1_6:=0, SI_3_6:=0, SI_3_2:=1
MT_SR_1>=100
S6

**Fig. 8.** UPPAAL TA models of simulation scenarios shown in Fig. 7.

application. Each TA model controls input variables and also clocks defined through UPPAAL TA system.

### *5.1.3. Applying automatic simulation of UPPAAL TA system*

Using the simulator tool of UPPAAL, automatic simulation is applied for every scenario. A comparison between the resulted internal states of all SFB TA models in UPPAAL and the Diagnostic Code (DiagCode) outputs of SFB instances in FBD shows that the UPPAAL TA system is identical to the FBD program.

## *5.2. Verification of the UPPAAL system*

After the validation of the UPPAAL TA system via simulation, the verification process can be applied via model checking. Textual safety functionalities defined by the developer are converted to CTL properties and are checked on UPPAAL TA system using the verifier tool. For example, the following safety requirement,

In case of emergency stop request or light curtain interrupting, safety system enables Safe Stop1 (SS1) safety function,

is converted to the following CTL statement

"$A[](!SI\_1\_0)!SI\_1\_2)!SI\_1\_4)!SI\_1\_6)imply(!SQ\_1\_1)$",

where A[] indicates the invariantly property. This property is satisfied, however, some other properties may not. The verification results help the developer to identify the software components that should be modified.

## 6. Summary and future work

In this paper, formal models of IEC 61131-3 FBD and UPPAAL TA have been presented with the objective to automate the transformation process between the two representations. Transformation rules were then defined based on these models. A prototype model-to-model transformer was developed using the Java language. The prototype transformer was used to test several real safety applications and prove its applicability. A real case study is used through the paper to demonstrate the applicability of the proposed approach and the understanding of the transformation, validation and verification processes. The results of applying this approach to several case example applications are very encouraging for the efficiency of this process.

We are currently working on the automation of the validation process and the development of an approach to automate the transformation of SFBs to UPPAAL TA models.

## References

Alur, R., & Dill, D. (1994). A theory of timed automata. *Theoretical Computer Science, 126*, 183–236.

Behrmann, G., David, A., Larsen, G. (2004). A tutorial on UPPAAL, UPPAAL web site, Documentation, Tutorials.

International Electrotechnical Commission (IEC) (1998). IEC 61508 – functional safety of electrical/electronic/programmable electronic safety-related systems.

International Electrotechnical Commission (IEC) (2000). *IEC 62061 – safety of machinery – functional safety – electrical, electronic and programmable electronic control systems*. Committee draft.

International Electrotechnical Commission (IEC) (2003). IEC 61131-3 programmable controllers: – Part 3: Programing languages.

Larsen, K., Pettersson, P., & Yi, W. (1997). UPPAAL in a nutshell. *International Journal of Software Tools for Technology Transfer, 1*, 134–153.

Lewis, R. (2002). *Can IEC 61131 graphical languages be used for safety related PLC applications?* The application of IEC 61131 in industrial control, UK.

Németh, E., & Bartha, T. (2009). Formal verification of safety functions by reinterpretation of functional block based specifications. *Formal Methods for Industrial Critical Systems, LNCS, 5596*, 199–214.

Pavlovic', O., & Ehrich, H. (2010). Model Checking PLC Software Written in Function Block Diagram. In *International Conference on Software Testing, Verification and Validation* (pp. 439–448). Paris, France.

PLCopen (2006). *Part 1: Concepts and function blocks*. TC5 safety software technical specification, Ver.1.0, Germany.

PLCopen (2008). *Part 2: User examples*. TC5 safety software technical specification, Ver.1.0, Germany.

PLCopen (2009). *XML Formats for IEC 61131-3*. TC6, technical paper, Ver.2.01, Germany.

Silva, L., Barbosa, L., Gorgonio, K., Perkusich, A. and Lima, A. (2008). On the automatic generation of timed automata models from function block diagrams for safety instrumented systems. In *34th Annual Conf. of the IEEE Industrial Electronics Society* (pp. 291–29.

Soliman, D., and Frey, G. (2009). Verification and validation of safety applications based on plcopen safety function blocks using timed automata in UPPAAL. In *2nd IFAC Workshop on Dependable Control of Discrete Systems (DCDS)* (pp. 39–44) Bari, Italy.

Soliman, D., Thramboulidis, K., & Frey, G. (2011). A methodology to upgrade legacy industrial systems to meet safety regulations. In *Proceedings of the 3rd International Workshop on Dependable Control of Discrete Systems* (pp. 141–147). Saarbrücken, Germany.

Soliman, D., & Frey, G. (2011). Verifications and validations of safety applications based on PLCopen safety function blocks. *Control Engineering Practice*, 929–946.

Thramboulidis, K. (2010). The 3+1 SysML view-model in model integrated mechatronics. *Journal of Software Engineering and Applications (JSEA), 3*(2), 109–118.

Thramboulidis, K., Soliman, D., & Frey, G. (2011). Towards an automated verification process for industrial safety applications. In *IEEE 7th International conference on Automation Science and Engineering (IEEE CASE 2011)*. August 24–27, Trieste, Italy.

Toon, K. (2002). *IEC 61131-3 in safety applications*. The application of IEC 61131 in industrial control, UK.

Wardana, A., Folmer, J., & Vogel-Heuser, B. (2009). Automatic program verification of continuous function chart based on model checking. In *The 35th Annual Conference of the IEEE Industrial Electronics Society (IECON 09)* (pp. 2422–2427).

Yoo, J., Cha, S., & Jee, E. (2008). A verification framework for FBD based software in nuclear power plants. In *15th Asia-Pacific Software Engineering Conference* (pp. 385–392).

## Further reading

International Electrotechnical Commission (IEC) (2003/2004). IEC 61511 – functional safety: Safety Instrumented systems for the process industry sector.

John, K., & Tiegelkamp, M. (2001). *IEC 61131-3: Programming Industrial Automation System: Concepts and Programming Languages, Requirements for Programming Systems, Aids to Decision-Making Tools*. Springer. Germany.

**Doaa Soliman** received the B.Sc. degree from Department of Electrical Engineering and Technology, High Institute of Technology, Benha University, in 1996 and received the M.Sc. degree from the same institution in 2003. She is currently a Ph.D. candidate in the chair of Automation, Saarland University, Germany. Her research interests include verification and validation of logic control safety applications.

**Kleanthis Thramboulidis** is a Professor in Software Engineering at the Department of ECE at University of Patras, Greece, where he is leading the Software Engineering Group (http://seg.ece.upatras.gr/seg). He was a Visiting Professor at Saarland University, Germany in 2010, and TKK Finland in 2009. He is the Designer of REDOM, CORFU, and he has proposed Model Integrated Mechatronics (MIM), a new paradigm for the model driven development of mechatronic systems. He has organized several special sessions in the domain of industrial automation. His research areas cover object-technology, model-driven development, meta-modeling, distributed control and automation systems, embedded systems, CASE tools, component and service-based development, service-oriented architectures, semantic web and mechatronics.

**Georg Frey** is a Full Professor and holds the Chair of Automation at Saarland University, Saarbrücken, Germany. Dr. Frey coauthored 160+ peer-reviewed papers covering his research interests from modeling and simulation of complex automation and energy systems over development processes for distributed and safe automation systems to design and formal validation of logic controllers. He co-chairs the "Information Technology in Industrial and Factory Automation" sub-committee of the IEEE/IES TC on Factory Automation and member of IFAC TCs 1.3 "Discrete Event and Hybrid Systems" and 5.1 "Manufacturing Plant Control" in GMA (German IFAC NMO) he chairs TC1.50 "Methods of Logic Control".