# A Formal Specification Method for PLC-Based Applications

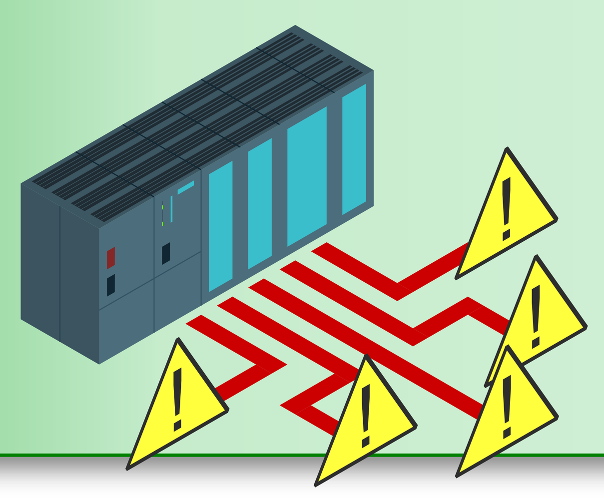
2015 ICALEPCS melbourne • australia

WEPGF091

**D. Darvas, E. Blanco Viñuela**, CERN, Geneva, Switzerland
**I. Majzik**, Budapest University of Technology and Economics, Budapest, Hungary
daniel.darvas@cern.ch | enrique.blanco@cern.ch | majzik@mit.bme.hu

## 1) Motivation

### critical systems ⇒ need for verification ⇒ need for convenient specification



- Programmable Logic Controllers (PLCs) are often used to control **critical systems**
- **Formal verification** (e.g. model checking) can improve the quality of critical systems: It checks if the designed/implemented system satisfies its requirements

⇒ **A specification method** is needed that can describe the requirements of the system

**But:**
- Available *formal methods* are typically **not adapted to the PLC domain**
  ⇒ difficult usage, need for extensive training or external expertise
- The *PLC specification methods* are typically **not formal or not convenient**

## 2) Our Proposed Solution: PLCspecif

### 2a) Main requirements

A **formal**, **lightweight behaviour description** method is needed that is **useful in practice**.

Therefore we need:
- *simple semantics* adapted to the PLC domain
- specific treatment for the *I/O handling* (to keep the core logic clean)
- *multiple formalisms* for the different behaviours
- *limited expressivity* to limit the possible errors

### 2b) Structure of the specification

*Each module (both composite and leaf modules) is further decomposed into three main parts: (1) input definitions, (2) core logic, and (3) output definitions. According to the semantics of PLCspecif, these parts are executed sequentially.*

**ExampleModule**

declarations

**Assigned inputs:**
- ValueReq : INT16
- EnableReq_fromLogic : BOOL
- EnableReq_fromScada : BOOL
- EnableReq_fromField : BOOL
- DisableReq : BOOL
- PMin : INT16 param
- PMax : INT16 param

**Assigned outputs:**
- Value : INT16
- Status : BOOL

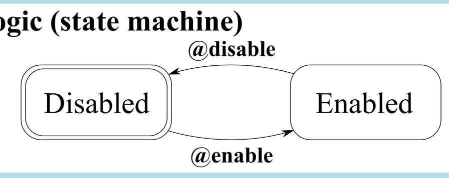
input definitions

**Input definitions:** — (none)

**Event definitions:**
- @disable ⇐ rising_edge(DisableReq) (pri=1)
- @enable ⇐ EnableReq_fromLogic OR EnableReq_fromScada OR EnableReq_fromField (pri=2)

core logic

**Core logic (state machine)**




output definitions

**Output definitions:**
- _Value =

| ValueReq < PMin | ValueReq > PMax | result |
|---|---|---|
| T | · | PMin |
| F | T | PMax |
| F | F | ValueReq |

- Value =

| in_state(Enabled) | result |
|---|---|
| T | _Value |
| F | 0 |

- Status = in_state(Enabled)

invariants

**Invariant properties:**
- ALWAYS *PMin* ≤ *Value* ≤ *PMax* ASSUMING *PMin* ≤ *PMax*

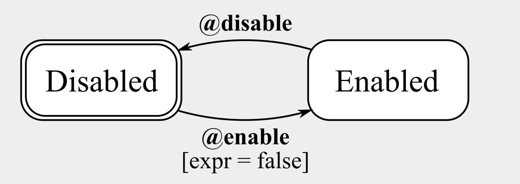
### 2c) Core logic description

core logic

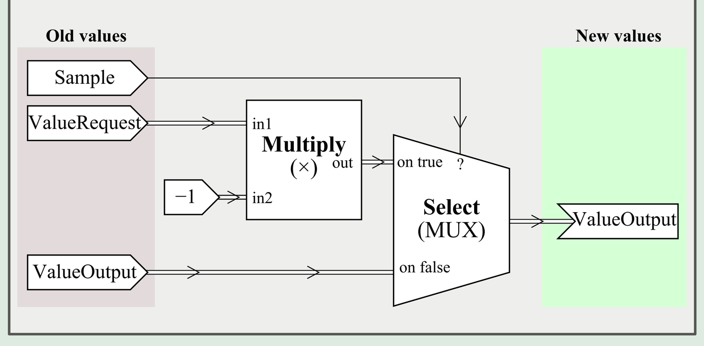
*One single formalism cannot conveniently fit the different types of modules (with state-based, data-flow-oriented and time-dependent behaviour). Therefore we introduced three types of core logic descriptions: **state machine modules**, **input-output connection modules** and **PLC timer modules**.*

3+3 different logic description method for different behaviours:

**State machines**
*For modules with enumerable states*
Main concepts: states, transitions, guards, triggers




**I/O connection modules**
*For modules with complex data-flow or integer states*
Main concepts: blocks, input/output pins, data edges




**PLC timers**
*For modules with timed behaviour*
Represents TON, TOFF or TP

**TON**
**expression to delay:** x AND y
**delay length:** dT
**reset events:** @reset1 @reset2

**Composite module**
*To group different behaviour descriptions together*

**Stateless module**
*Module without state, only input-output definitions (e.g. stateless safety logic)*

**Alternative module**
*To have different behaviour depending on some inputs*

### 2d) Expression description

input definitions | output definitions

*The input or output definitions may contain complex expressions. While the **arithmetic form** is suitable to describe simple expressions ("a OR b"), it does not scale up well. PLCspecif supports the usage of other expression description methods: **AND/OR tables** and **switch-case tables**.*

**Arithmetic form** *a AND NOT b AND c* **or** *in1 * 10 + offset*

**AND/OR table**
*To describe complex Boolean expressions*

x =

| | Case 1 | Case 2 |
|---|---|---|
| a | true | true |
| b | false | false |
| c | true | · |
| d | · | false |

The **x** will be set to true if
**a AND NOT b AND c** (Case 1),
or if
**a AND NOT b AND NOT d** (Case 2).
Otherwise, **x** will be false.
*x = a AND NOT b AND (c OR NOT d)*

**Switch-case table**
*To treat complex assignments*

_Value =

| V < PMin | V > PMax | result |
|---|---|---|
| T | · | PMin |
| F | T | PMax |
| F | F | V |

**_Value** will be:
- PMin, if V < PMin,
- PMax, if V > PMax,
- ValueReq, if PMin ≤ V ≤ PMax.

*Ambiguity, incompleteness, consistency can be checked automatically*

## 3) Benefits

### Improved understanding

- Specification helps both **reuse** and **software evolution**
- **Decoupling I/O handling** helps to focus on the core logic
- A **restricted method** helps to make unambiguous and implementable specifications
- The **simple semantics**, well-adapted to the PLC domain makes the specifiers confident

### Code generation

- It is possible to **automatically generate PLC code** based on a PLCspecif specification
- The generation process can be **configured** and **fine-tuned** to match to the user's expectations
- The generated code **preserves the properties** satisfied by the specification
  *If a property is checked on the specification, it does not have to be checked on the implementation*

### Consistency checking and formal verification

- The **consistency (ambiguity, completeness)** of the specification **can be checked**
- Formal argumentation about correctness is possible (e.g. **model checking,** cf. PLCverif)
- The user can extend the specification with **invariant properties**
  E.g. *X and Y outputs should never be true at the same time.*

invariants

### Equivalence check, refinements

*Equivalence and conformance checking is the process of comparing the behaviours of models with formal semantics, e.g. if they provide the same outputs for the same input sequences. This can show the equivalence of two modules with different internal structure.*

It is possible to **compare the behaviours** (not the syntax!) of two specifications/implementations
**Use cases:**
- Comparing two versions of the same specification (refinement)
- Comparing specification to legacy or manual implementation

The user can define for each variable the required **level of conformance**

You can find this poster, the paper and more information at
**http://go.cern.ch/9JVk**
**http://cern.ch/plcspecif**

CERN | EN ENGINEERING DEPARTMENT | ICE INDUSTRIAL CONTROL