8TH IFAC SYMPOSIUM ON COST ORIENTED AUTOMATION
Affordable Automation Systems
Ciudad de la Habana, Cuba, 2007

TOWARDS USAGE OF COMBINED EXERCISES IN LEARNING OF LOW COST CONTROLLERS

Richard Šusta [**] **Jan John** [*]

[*] *DCE-Prague: Department of Control Engineering, Faculty of Electrical Eng., Prague, Czech Republic*
susta@control.felk.cvut.cz
[**] *DCE-Prague,* john@control.felk.cvut.cz

Abstract: The paper outlines our experience with combined educational exercises consisting of virtual and physical parts and they are suitable for training courses of low cost programmable logical controllers (PLCs). First of all, we introduce our training approach based on utilizing real PLCs connected to software simulations of industrial processes. The simulation offers possibility to select the complexity of a controlled task, e.g. a presence of random malfunctions. Tutors assign proper grades to students according to the quality of their programs but their testing often leads to time-consuming operations. For simple educational tasks, we have made successful experiments with formal methods based on partial search of state space of automata, which we present in the second part of the paper. Finally, we consider possible applications of our approach for a remote learning.

Keywords: control program, programmable logical controllers, PLCs, formal methods, case study, soft-commissioning, remote learning

1. INTRODUCTION

It is beyond doubt that experience is the best way to improve our programming skills. Additionally, programs themselves are sometimes considered as a coding of experience. In training laboratories, students may gather both exciting and practical knowledge along with solid preparation for their later successful careers. Laboratory workplaces can be equipped by either physical or virtual exercises.

Physical control exercises offer many unsubstitutable gains in terms of illustrative clearness and motivations. However, they are typically static in structure and unfortunately destructible. Therefore, they always need robust "student-proof" constructions, and also a repeatable behavior that minimizes operational and maintenance costs. To satisfy all conditions, these exercises are often based on simple physical models that represent rather academic processes.

On the contrary, *virtual control exercises* can even supersede real processes because they offer safe simulations of destructive accidents caused by possible improper control actions.

[1] This work was supported by FRVŠ 2150/2005 project and by euSophos 2C06010 project granted by Ministry of education.

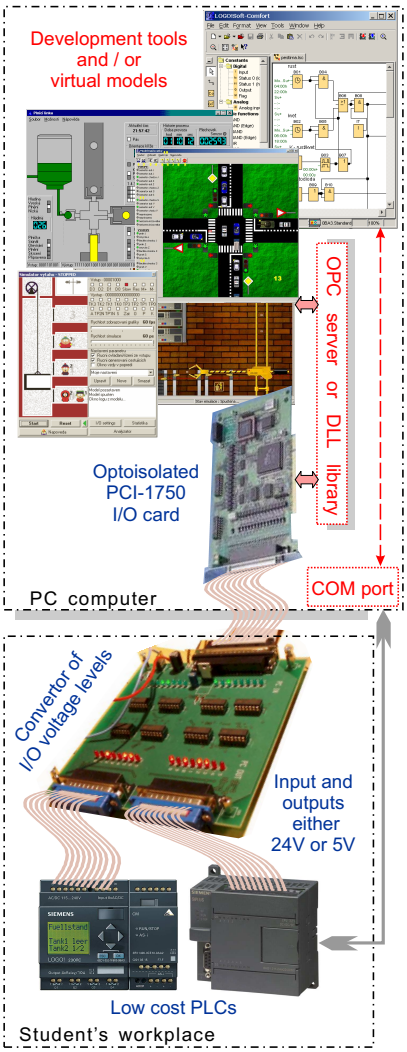

Fig. 1. Workplace organization

However, virtual models have at least one major drawback that will be probably never improvable — users do not operate them directly and it has impact on the amount and type of information gathered.

Several commercial products exist for strictly virtual education where computers emulate whole feedback loops, i.e. technology processes and controllers (for example Automation studio). Such approach offers cost effective and very flexible way, but too virtual, especially for beginners without practical experience.

Combined exercises, i.e. exercises consisting of physical and virtual parts, could represent a middle course. The question is how to demarcate the two parts.

Our approach, depicted in Figure 1, differs from strict virtual reality by physical PLCs, which are fully new control elements for beginners in industrial control technologies, whereas PC simulated processes are mainly taken from everyday life. Thus every student workplace represents an analogy to a remote control where responses of a distant process are only monitored, for instance by web cameras.

Students work with real PLCs and see their input and output signals, either 24 V industrial I/O or 5 V logic circuits. These signals are converted to voltage levels of standard I/O PCI cards. To protect PC computers as much as possible, we have employed PCI-1750 cards with 16 inputs and 16 outputs that are all fully independent and optically isolated.

Our older models communicate with PCI-1750 cards via a DLL library, which is supplied by PCI-1750 manufacture, and more recent models prefer more flexible OPC server created for these purposes. Development environments for programmable logical controllers can certainly run on the same or different computers, as required.

The experience from last years shows that students like our combined exercises, not only for attractive design, but also for their better usability. Our approach offers many pedagogical contributions:

- Students concentrate more on control operations, which is a primary goal of our training course. They have no trouble with bothersome mechanical malfunctions because simulated models are always in a perfect shape.
- Students train their skills to manipulate properly with real PLCs.
- All consequences of wrong programs are shown on simulations, including crashes and destructive accidents.
- The use of laboratory equipments is better. Each of ten work places in our laboratory can operate with any virtual industrial process, which reduces sharing problems with reservations of hardware equipments.

The popularity of combined exercise is also confirmed by the fact that some students volunteer to design and to program new models as their individual projects in higher classes. Up to now, they have built nine virtual models. It is possible to control five of them by low cost PLC controllers; the remaining models are directed to advance courses.

1.1 Testing of Student Programs

We try to automate the validation of student works for many reasons, but mostly to improve the quality of education. Erroneous student programs frequently pass simple tests and fail only in situations that are more complex.

Their manual in-depth "commissioning" leads to a time-consuming and boring procedure. Our semi-automatic tests of the programs can be divided into two cases: a soft-commissioning and a partial verification.

The *soft-commissioning* means debugging a real PLC program with the aid of the virtual model, which practically leads to testing of a proper behavior of a control program by the sequences of events generated by the model. The soft-commissioning is quite illustrative and students usually perform it by themselves; on the other hand, such tests are slow down by responses of models and PLCs, thus, the main question is how to select proper test scenario.

The *partial verification* can reveal more errors. It begins by converting the PLC program into some automaton and closing feedback loop by adding controlled model. Finally, required criteria are checked by NuSMV model checker Cimatti and at al. (2002). We chose only partial searches because well-known state explosion problems frequently occur even when verifying seemingly simple programs.

Automatic tests of programs have several pedagogical contributions:

- Students must write good programs. There is no space for any tricks how to persuade their tutor to give them grades for partially functioning programs.
- Tutors can concentrate more on technical details because they do not lose time with exhausting tests that should be completely repeated after correcting each found bug.

2. THEORETICAL MODEL OF COMBINED EXERCISES

In this section, we will create a theoretical model for our combined exercises. We present conditions that allow representing the virtual model of an industrial process $\mathcal{P}$ as an untimed finite state machine even if it simulates a complex hybrid timed process.

A virtual process $\mathcal{P}$ is simulated by a PC computer. It generates input events $x \in X$ for a tested control program $\mathcal{C}$ that runs on a PLC, see Figure 2. Control actions $u \in U$ outputted by $\mathcal{C}$ are processed in several $\mathcal{P}$ blocks, which we include in the following quadruple:

$$\mathcal{P} \stackrel{df}{=} \langle \mathcal{M}, \mathcal{R}, \mathcal{J}, \mathcal{D} \rangle$$

where

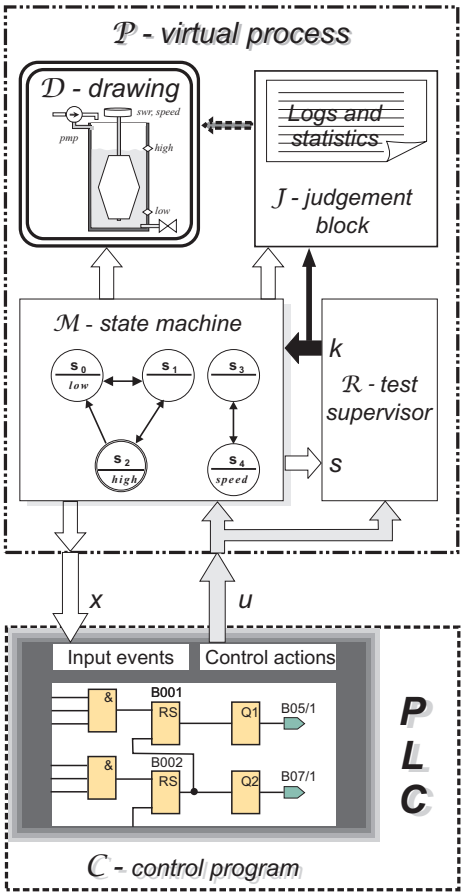

Fig. 2. Virtual process and tested program

- $\mathcal{M}$ represents some state machine, usually a timed hybrid automaton with some finite set S of discrete states.
- $\mathcal{R}$ supervises $\mathcal{M}$. It detects states that violate specified by safety properties and modifies $\mathcal{M}$ behavior by blocking or enabling some transitions to next states according to a momentary performed test scenario. Its current $k \in K$ output serves as a supplementary control input signal.
- $\mathcal{J}$ judgment block measures the quality of $\mathcal{C}$ control by some given criteria and it provides statistic data. In our models, $\mathcal{J}$ also creates $\mathcal{C}$ behavior log, so operators can be temporary absent when running long tests.
- Finally, $\mathcal{D}$ drawing unit animates the behavior of $\mathcal{M}$ and creates illusions of controlling of some distant industrial process.

If we represent $\mathcal{M}$ as a hybrid timed automaton, or eventually as a hybrid timed Petri net, we will obtain very accurate model, but with less decidability. Therefore, we describe $\mathcal{P}$ as a binary finite automaton, which is possible due to the following facts:

(1) $\mathcal{C}$ runs on a classical PLCs that operates in the well-known cyclic manner, which consists from 3 phases, see Figure 3: sampling

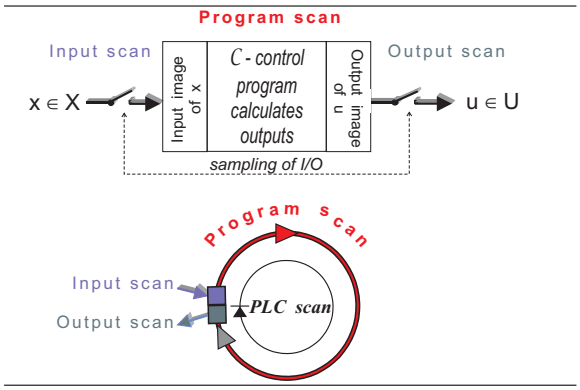

Fig. 3. PLC cycle

inputs, executing $\mathcal{C}$ program, and writing outputs to peripherals.

(2) Low cost PLCs utilized in this training course (*Siemens LOGO! OBA3*) have few inputs, output, and they operated in relatively long scan cycles. Their I/O sampling has nearly regular 1 second period.

(3) Our exercises are based on models with short delay or waiting times, at most 10 seconds, because we train programming skills, not a patience waiting.

All responses of $\mathcal{P}$ are sampled by PLC scan, therefore, $\mathcal{M}$ operations can be modeled as a synchronous automaton performing an analogy of PLC scan cycle where times are converted into sequences of transitions to next states.

$\mathcal{M}$ behavior consists of several mutually unsynchronized automata, which proper synchronization is solved by students in $\mathcal{C}$ programs as an integral part of training exercises, as also indicated in Figure 2.

A composition of all partial automata into one huge finite state machine would neither express internal architecture of $\mathcal{P}$, nor it would reflect utilized testing procedures. We base on definition on a network of automata.

Let us write $\mathbb{I}$ for the set of all integer numbers, $\forall i \in \mathbb{I}; i > 1$, and

$$\alpha(B) \stackrel{df}{=} \{0,1\}^{|B|}$$

for a Cartesian product where B is any finite nonempty ordered set of binary variables. We utilize $\Pi_n Q_i$ as abbreviation for Cartesian product of some non empty finite mutually disjunctive Q_i sets:

$$\Pi_n Q_i \stackrel{df}{=} \Pi_{i=1}^{n} Q_i = Q_1 \times Q_2 \times \cdots \times Q_n$$

where $i, n \in \mathbb{I}, i \leq n$ and $j \neq k$ always implies $Q_j \cap Q_k = \emptyset, \forall Q_j, Q_k \in \Pi_n Q_i$.

We will write $\Delta(\overline{\tau}, B)$ for a *latched memory set* that stores current values of B variables when $\tau = 0$, and it holds their values if $\tau = 1$. It behaves as a set similar to B, $|B| = |\Delta(\overline{\tau}, B)|$. The memory can be constructed as an output of proper automaton, but let us omit its definition for abbreviations.

First we define *finite synchronous state machine* (FSSM), which transitions are enabled by external binary clock τ, as a quadruple

$$\mathcal{G}(\tau) \stackrel{df}{=} \langle D, Q, \phi(\tau) \rangle$$

where D and Q are a nonempty finite ordered sets of binary input data and states. To abbreviate definitions, let us suppose that Q also contains some predefined initial state. Mapping $\phi(\tau)$ is specified by:

$$\phi(\tau) \stackrel{df}{=} \begin{cases} \alpha(\Delta(\overline{\tau}, D)) \times \alpha(\Delta(\overline{\tau}, Q)) \rightarrow \alpha(Q) \\ \qquad \text{if } \tau = 1 \\ q \rightarrow q, \forall q \in Q \qquad \text{otherwise} \end{cases}$$

In the words, any transition to next states occurs only if τ signal equals to 1, otherwise $\mathcal{G}(\tau)$ remains in its current state.

Because all transitions depend only on values that were stored when $\tau = 0$, $\mathcal{G}(\tau)$ *performs at most one transition* on $\tau = 1$. It is our main reasons for its definition.

We define a *finite nonempty set of FSSMs* as

$$\Gamma(\tau, D, \Pi_n Q_i) \stackrel{df}{=} \{\mathcal{G}_i(\tau) \langle D_i, Q_i, \phi_i(\tau) \rangle ; i = 1..n\}$$

where $i, n \in \mathbb{I}, i \leq n$, $D_i \in D$ are input data sets, and Q_i are mutually disjunctive finite sets of states.

We have utilized Cartesian product to emphasize that all decisions inside model will be are derived from the states of individual automata.

Finally, we have all construction necessary for *network of unsynchronized FSSMs*. We define $\mathbb{N}$ as tuple:

$$\mathbb{N} \stackrel{df}{=} \left\langle \tau, U, \Gamma(\tau, U \cup K, \widehat{S}), \mathbb{R}(\widehat{S}, K), X, \omega \right\rangle$$

where

- τ is an binary internal clock;
- U is a finite nonempty ordered set of binary variables that represent inputs of $\mathbb{N}$;
- Γ is a finite nonempty set of FSSMs, which was presented in the previous definition.
- $\widehat{S}$ denotes the set of vectors created as $\widehat{S} = \Pi_n Q_i$ Cartesian product of Γ states.
- $\mathbb{R}(\widehat{S}, K)$ supervisor calculates blocking or enabling conditions according to a given criteria and test scenarios.
- Finally, X is a finite set of binary outputs to external systems created by ξ mapping $\xi : \widehat{S} \rightarrow \alpha(X)$.

Notice, each FSSM in Γ performs at most one transition on $\tau = 1$ that is determined by data

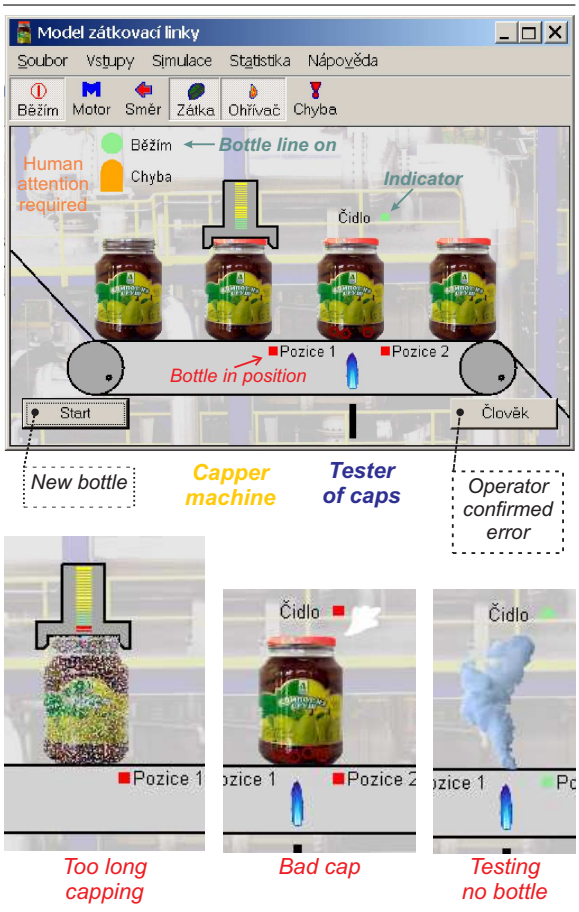

Fig. 4. Example: Capping line in manual mode

latched when τ was 0. Thus, while $\tau = 1$, no FSSM can influence other elements of Γ.

The previous definition allows building binary untimed analogies of some hybrid systems. Resulting models are easily expressible in NuSMV checker language, which was our main reason for building $\mathbb{N}$ representation by an unsynchronized network of FSSMs.

2.1 Example: Capping line

We demonstrate our approach with the aid of a capping line, see Figure 4. Its virtual model simulates a simple assembly operation suitable for learning purposes.

A bottle moves on a conveyor and its positions are indicated by two sensors, at a capping machine and at a tightness tester. Each bottle firstly slides from left chute. When the bottle arrives below the capper, the conveyor is stopped and 2 second capping phase begins. Capped bottle is moved to the tightness tester, where 5 second checking operation is performed . If it was successful, finished bottle continues towards the right chute, where it disappears.

Bad bottle is moved back below the capper, where one attempt of its recapping is performed followed by new tightness test. If the test fails again, the wrong bottle is rejected.

Figure 4 shows the screen shot of our virtual model (programmed in C# language). The model was switched into a manual mode, where it is possible to add more bottles. However, the exercises given to students suppose that only one bottle moves on the conveyor, because our low cost PLCs are not capable to control operations that are more complex.

The model indicates also erroneous control actions, some of them are depicted on Figure 4. In addition to already mentioned bad capping, the model simulates jamming of bottles. Because there are no sensors indicating such accidents, students must detect too long transport times between the places. It is a general practice in industry applications for substituting of expensive additional sensors.

NuSMV model checker allows only finite types. Therefore, position variables and delay times are quantified to enumerated types. The bottle position was quantified to 16 levels. For time variables, we have chosen the scale of 1 second as 1 state transitions that corresponds to LOGO! scan. We have selected 4 states for the capper and 8 states for measuring testing phase (to add possibility of erroneous overruns).

In NuSMV, we utilize enumerated types, e.g. capping time will be *VAR captm : 0..7;*. Internal clock τ is a Boolean variable that periodically flip-flops: *next(tau) := !tau;*. The capper was expressed by one FSSM machine with variable *qcaptm : 0..7;* as capping time memory. It gives the following NuSMV code:

```
init(captm) := 0;
init(qcaptm) := 0;
next(qcaptm) :=
 case
  !tau: captm;
  1: qcaptm;
 esac;
next(captm) :=
 case
   tau & runcapper & (qcaptm¡7) : qcaptm+1;
   tau & runcapper & (qcaptm = 7): 7;
   tau & !runcapper & (qcaptm>0) : qcaptm -1;
   tau & !runcapper & (qcaptm=0): 0;
   1: captm;
 esac;
```

The whole model is easily built in similar way as several independent processes. They are supplemented by $\mathbb{R}(\widehat{S}, K)$ supervisor, which consists of logical conditions for detecting er-

rors, for instance if *runcapper* reaches state greater than 3 (it will run more than 3 seconds), then the bottle is broken.

2.2 Example of Verifications

Here, we present the verification of one student's program that controlled the capping line. The program contained 52 logics blocks and five timers with time constants in seconds. It utilized five bit inputs, 6 bit outputs, and three internal flags. LOGO! program (stored in ladder diagram format) was transformed by our external tool into a NuSMV input file and was joined with our prepared model of the capping line. Its timers were replaced by corresponding finite state machines.

We verified the states of bottles at right end of the conveyor, if they are capped, properly checked, and unbroken. Unbound verification created the state explosion problem and NuSMV run had to be aborted. A bounded check found counterexample after 19 minutes when checking the model with bound 123.

Of course, there are many possibilities how to improve the conversions of PLC programs into more effective NuSMV models.

3. CONCLUSION AND FUTURE WORK

In this paper, we have presented probably fully new contribution to software testing based on "PLC scan" conversion of a nondeterministic hybrid timed automaton into deterministic automaton. Unfortunately, this method is applicable only to narrow set of educational task running on slow low cost PLCs.

In any case, our work may be considered as a suggestion for similar studies, especially for remote learning, which is one of main topics supported by our long term project. Students will debug their codes in combined exercises without risks of damage. The soft-commissioning and partial verifications can either draw student's attention to errors or serve as a basis for giving grades.

Eventually, PC simulations in combined exercises can also mirror physical models to protect their construction against shock stresses in extreme states. After control programs have successfully passed all tests, their I/O signals are connected to real systems.

4. APPLIED METHODS AND RELATED WORKS

We surveyed many related works in this area. Some of them have dealt with similar problems, but we have found out no paper that specialized in low cost controllers with long scan cycle, which brought new possibility how to simplify validation of programs. For completeness, we mention some papers in software testing that deal with similar problems. An overview of software testing can be found in Muccini (2002). The verification of student programs utilized by Vienna University of Technology is presented in Legourski *et al.* (2005), but the authors specialize in Atmel ATmega128 microcontrollers, which require different methods than slow PLCs. Our soft-commissioning is based on data flow techniques are described in Tretmans and Belinfante (1999), Hong *et al.* (2003), and Bernardo and Inverardi (2003). Their results are partially similar to simpler $\mathcal{R}$ approach presented in this paper, but they are based on different sets of states.

REFERENCES

Bernardo, Marco and Paola Inverardi (2003). Formal methods in testing software architectures. In: *SFM*. Vol. 2804 of *Lecture Notes in Computer Science*. Springer. pp. 122–147.

Cimatti, A. and at al. (2002). NuSMV 2: An opensource tool for symbolic model checking. In: *Proceeding of the 14th International Conference on Computer-Aided Verification (CAV'2002)*.

Hong, Hyoung Seok, Sung Deok, Insup Lee, Oleg Sokolsky and Hasan Ural (2003). Data flow testing as model checking. In: *Proceedings of Internal Conference on Software Engineering (ICSE'03), Portland, Oregon*.

Kovchegov, Vladislav B. (2005). Modelling of human society as a locally interacting product-potential networks of automata. In: *The 7th Understanding Complex Systems Symposium*. University of Illinois.

Legourski, V., Ch. Trödhandl and B. Weiss (2005). A system for automatic testing of embedded software in undergraduate study exercises. *SIGBED Rev.* **2**(4), 48–55.

Muccini, H. (2002). Software Architecture for Testing, Coordination and Views Model Checking". PhD thesis. Universita degli Studi di Roma.

Tretmans, J. and A. Belinfante (1999). Automatic testing with formal methods.. In: *EuroSTAR'99: 7 European Int. Conference on Software Testing, Analysis and Review, Barcelona, Spain*.