European Symposium on Computer Aided Process Engineering – 15
L. Puigjaner and A. Espuña (Editors)


# Initiation and inhibiting mechanisms for multi-tasking control in discrete event systems

S. Macchietto*, Nicholas J. Alsop, Ross J. Baird, Zhang P. Feng and Bing H. Chen
Department of Chemical Engineering, Imperial College London,
London SW7 2AZ, UK

## Abstract

A theoretical analysis is presented of sequential controller initiation (pre-check) and inhibition for concurrent operations within a formal Procedural Control Theory (PCT) framework. These aspects are essential to the safety and performance of multitasking sequential controllers at the design and operation stages. The pre–check mechanism ensures the correct initial state of the process when controller is synthesized. The inhibit function prevents nominated controllers potentially using shared devices from starting while another controller is already active. The ideas are explained via a small illustrative example.

**Keywords:** formal methods, procedural control theory, discrete event systems, safety.

## 1. Introduction

Formal techniques to support the design of automation software for discontinuous processes are grounded in logic and discrete systems theory. Their primary goal is the improvement in process safety and operation by eliminating errors in sequential control software that often occur during the design of control logic and its implementation in automation systems. Unlike the *verification* approach (Nimmo 1994, Moon et al 1992), which checks controlled process against specifications, the idea in sequential control *synthesis* methods is to develop "provably correct systems" that can be shown a-priori to meet all specifications and generate no undesired behaviours. The Procedural Control Theory (PCT) (Sanchez and Macchietto, 1995, Sanchez et al., 1999) is one of them.

Industrial sequential controllers are programmed in control languages typically organized into sets of "*operations*" (in the ISA S88 standard sense, ISA 1995). Each *operation* is started by an operator or a supervisory control system. Control architectures are multitasking, i.e. support concurrent control by multiple operations. *Operations* use process related hardware resources (e.g. pumps) that are usually shared with other *operations*. Without adequate safeguards, an *operation* could be started when its resources are in an inappropriate state or already in use by other *operations*. In either case conflicts may arise and at the very least the proper execution of the operation cannot be guaranteed. At worst, unsafe states of the process could be reached. Therefore, suitable initiation and inhibit logic is required in such circumstances, but

* Corresponding author, Tel: +44 20 5945575, Fax: +44 20 75945604, email: s.macchietto@imperial.ac.uk

these issues have been ignored in the literature. This work deals with them by formally introducing controller initiation (pre-check) and inhibition within the PCT framework.

## 2. Process Modelling within PCT

In PCT, a Discrete Event Systems (DESs) model of a process is described as a Finite State Machine (FSM) comprising of a finite set of states and transitions, by 7 tuples:

$$M = \left\{Q, V^{n_v}, \Sigma, \delta, \gamma, q_0, Q_m\right\} \tag{1}$$

where Q is the set of states and $\Sigma = \Sigma_c \cup \Sigma_u$ the set of transitions (other symbols are defined in the nomenclature). Controllable transitions $\Sigma_c$, e.g. turning on/off a pump, are forced by a controller while uncontrollable $\Sigma_u$ transitions, such as the increase/decrease of a level sensor signal in a tank, occur indirectly and may only be observed.
In PCT, an elementary device (e.g. pump or sensor) is represented by a unique "state variable" which may take a number of discrete realizations (states) and by the set of "transitions" that take the device from one state to another. Two operations on FSMs generate the model of a composite system: the "Asynchronous Product" to interleave states from each FSM and the "Synchronous Product" to intersect two FSMs to generate strings common to both original FSMs. The FSM model *M* of a system represents every possible combination of feasible states and process trajectories in the open loop process. Desirable behaviour of the process is defined by imposing the existence or otherwise of states and transitions, or allowable strings of process events, via formal specifications.

## 3. The Closed Loop Controller Initiation with Pre–checks

The closed loop behaviour obtained by a controller $C_x$ that implements given specifications on model *M* is denoted by ($L(C_x / M)$. In general, controller performance is sensitive to any departure in the initial state of process *M* from its nominal initial state $q_0$'. Thus, additional functionality is required to handle possible variability in the initial state of the process. The simplest mechanism is a pre–check that ensures that an operator request for starting controller $C_x$ from a process state $q$ is granted only if $q$ is the nominal initial state $q_0$' of the process. A sufficient condition for $q = q_0$' is that every elementary component of *M* is in its nominal initial state. We define a new transition $\sigma_x$ to represent the actual start of controller $C_x$. Strictly, this is not an elementary process event (i.e. $\sigma_x \notin \Sigma$) and therefore does not alter the process state. Then the pre-check mechanism is to ensure that the process is in state $q_0$ for event $\sigma_x$ to occur (i.e. for the controller $C_x$ to start). Therefore, $\sigma_x$ is included in the model of the process as a self-looped transition at each potential starting state $q \in Q_x$. The closed loop behaviour of a system *M* under control of $C_x$ with pre–checks is then formulated as:

$$L(C_x / M) = \Sigma^* \cdot \sigma_x \cdot L(S^{X_w}_{\Sigma_u} C_x) \cap L(S^{Q_x}_{\sigma_x} M) \tag{2}$$

This can be read as: the closed loop language generated by $C_x$ on *M* with pre–checks is the set of physically possible strings made up from the concatenation of an open loop

trajectory and the starting event $\sigma_x$ plus the closed loop behaviour. $M$ is not under control from $C_x$ until the event $\sigma_x$ at a potential starting state $q \in Q_x$. Similarly, the marked closed loop response is given by:

$$L_m(C_x / M) = \Sigma^* \cdot \sigma_x \cdot L_m(S_{\Sigma_u}^{X_w} C_x) \cap L_m(S_{\sigma_x}^{Q_x} M) \tag{3}$$

## 4. Controller Inhibiting

A controller inhibit mechanism is defined to disable a controller from starting if the system does not satisfy a given set of conditions. In particular, this mechanism checks the state of nominated controllers in a multitasking control system. Checks are performed before a controller is started so that abortive action can be taken to avoid the concurrent operation of controllers that compete for the same resource. The approach is based on the concept of *non-cooperation* (Alsop, 1996) and requires the closed loop language generated by the process under parallel control by controllers $C_x$ and $C_y$ each with pre–checks. The *cooperative* and *non-cooperative* properties are used as follows: if $C_y$ is non-cooperative with $C_x$, then $C_x$ must inhibit $C_y$, and if $C_y$ cooperates with $C_x$ then no inhibition mechanism is necessary.

### 4.1 Closed loop language with pre–checks for two controllers in parallel

Using the closed loop language generated by a single controller (equation 2), the closed loop language generated by two controllers $C_x$ and $C_y$ on process $M$ augmented with the self-loop $\sigma_x$ at each possible initial state $q \in Q_x$ and $\sigma_y$ at each possible initial state $q \in Q_y$ with a pre–check mechanism is given by:

$$L(C_x \uparrow C_y / M) = \mathcal{P}_x^{-1}\{\Sigma_x^* \cdot \sigma_x \cdot L(S_{\Sigma_u x}^{X_w} C_x)\} \cap \mathcal{P}_y^{-1}\{\Sigma_y^* \cdot \sigma_y \cdot L(S_{\Sigma_u y}^{X_w} C_y)\} \cap L(S_{\sigma_x}^{Q_x} S_{\sigma_y}^{Q_y} M)\} \tag{4}$$

where $\Sigma = \Sigma_x \cup \Sigma_y$ and $\Sigma_x \cap \Sigma_y \subseteq \Sigma_u$. The first term of equation 4 is equivalent to the first term of equation 2 with the addition of the projection operation $\mathcal{P}_x^{-1}$. This operation permits any event from $(\Sigma - \Sigma_x) \cup \sigma_y$, and thereby restricts no events generated by $C_y$. Similarly, the second term represents the set of strings generated by controller $C_y$, which permits any event generated by $C_x$. The third term is the set of physically possible strings (i.e. the process model), augmented with self-loops $\sigma_x$ at each state $q \in Q_x$ and $\sigma_y$ at each state $q \in Q_y$. The augmented self-looped transitions allow both controllers to start when the process is appropriately initialised.

### 4.2 Inhibit design for controllers with shared controllable items

In the formulation of the combined closed loop language (equation 4), it is necessary for the two controller languages to not share controllable transitions (i.e. $\Sigma_x \cap \Sigma_y \subseteq \Sigma_u$). At first this may appear to reject from the inhibit analysis the case in which $C_x$ and $C_y$ share elementary components generating controllable events (e.g. driving the same valve). Typically, inhibits are necessary between pairs of controllers which drive the same equipment item. However situations may arise in which two controllers cooperatively employ the same equipment item. A special modelling technique is

employed to handle this case to ensure that controllable transitions will not be generated concurrently by both controllers. In other words, it is not possible for the controllers to synchronise on a common controllable event. Therefore two common controllable transitions generated by the two controllers are unique, and must be labelled accordingly. The labels identify an event with the controller from which it is generated, i.e. transitions in $C_x$ and $C_y$ need be relabelled to respect the condition that $\Sigma_x$ and $\Sigma_y$ share only uncontrollable events. In this work, we use a 'convert' technique (Wonham, 1996) for re-labelling transitions. For convenience, the set $\Sigma_{xy}$ is defined as all those controllable events in $\Sigma_y$ which have an equivalent $\Sigma_x$:

$$\Sigma_{xy} = \{\sigma \in \Sigma_{cy} / \forall_q \in Q, \delta(\sigma,q)!, \exists \sigma' \in \Sigma_{cx} \text{ s.t. } \delta(\sigma',q)! \wedge \delta(\sigma',q)=\delta(\sigma,q)\} \quad (5)$$

Events in $\Sigma_{xy}$ can occur in the open loop process or when generated by $C_y$, but not when controller $C_x$ is the only active controller. Similar occurrences apply to events in $\Sigma_{yx}$. A regulator *R* is therefore introduced to meet this requirement by interlocking shared controllable items. Its language is given by:

$$L(R) = \overline{\Sigma^* \cdot \sigma_x \cdot [\Sigma - \Sigma_{xy}]^* \cdot \sigma_y \cdot \Sigma^* \cup \Sigma^* \cdot \sigma_y \cdot [\Sigma - \Sigma_{yx}]^* \cdot \sigma_x \cdot \Sigma^*} \quad (6)$$

The first term prohibits events not in $[\Sigma - \Sigma_{xy}]$ after starting $C_x$ and before starting $C_y$ with the reverse for the second term. The closed loop language generated by process *M* under parallel control from $C_x$ and $C_y$ with pre–checks where $C_x$ and $C_y$ share elementary components generating controllable events is given by:

$$L(C_x \uparrow C_y / M) = \mathcal{P}_x^{-1}\{\Sigma_x^* \cdot \sigma_x \cdot L(S_{\Sigma_u x}^{X_w} C_x)\} \cap \mathcal{P}_y^{-1}\{\Sigma_y^* \cdot \sigma_y \cdot L(S_{\Sigma_u y}^{X_w} C_y)\} \cap L(S_{\sigma_x}^{Q_x} S_{\sigma_y}^{Q_y} M) \cap L(R) \quad (7)$$

Two controllers are cooperative (can be concurrent without inhibits) if the condition $L(C_x \uparrow C_y / M)$ is a subset of $\Sigma^* \cdot \sigma_x \cdot \mathcal{P}_x^{-1} L(S_x)$ holds true (Alsop, 1966) and vice versa.

## 5. An Example

A liquid ingredient is dosed in a tank (Figure 1) using Feed Valve (FV), Drain Valve (DV) and level sensors LV1 and LV2. These devices are modelled as two state FSMs (Figure 2). Nodes on the graph represent states and edges transitions. Controllable transitions of FV and DV are duplicated for the use of inhibitions. Uncontrollable transitions are shown as dashed arcs, while controllable ones as solid arcs.

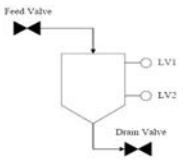



*Figure 1. Dosing tank*

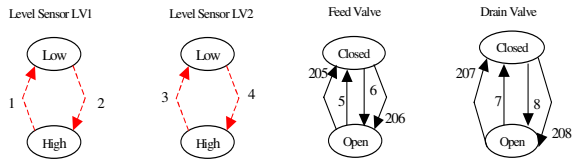



*Figure 2. Elementary components modelled as FSMs*

We wish to design a procedure is to fill the tank to level 2, ready for the next stage of the process. The initial and goal (marked) states are:

- Initial state: FV is closed, DV is closed, LV1 is low and LV2 is low.
- Goal state: FV is closed, DV is closed, LV1 is low and LV2 is high.

The overall process model (Figure 3), obtained by the asynchronous operator of the components of Figure 2, includes all possible combinations of state variables {FV,DV,LV1,LV2} and transitions. A standard controller (i.e. before pre-checks and inhibits) for filling the tank up to level sensor 2 is obtained (Figure 4 (a)) using the following specifications:

*Static constraints:*

1. The process should never reach a state where LV1 is high
2. When LV2 is low, LV1 cannot be high

*Dynamic constraints:*

3. If DV is closed it is impossible for LV1 or LV2 to fall
4. If FV is closed it is impossible for LV1 or LV2 to rise
5. If FV is open, do not allow DV to open
6. If LV2 is low, do not allow FV to close
7. If FV and DV are closed and LV1 and LV2 are low, then open FV
8. If FV and DV are closed, LV1 is low and LV2 is high, do not open DV
9. If FV and DV are closed, LV1 is low and LV2 is high, do not open FV
10. If LV2 is low, do not allow DV to open
11. If valve DV is closed, do not open it

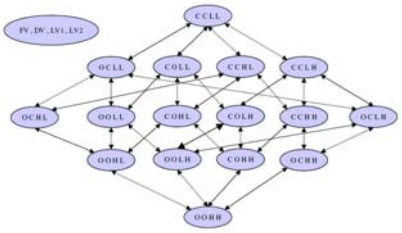
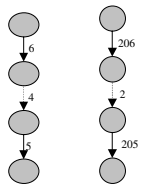


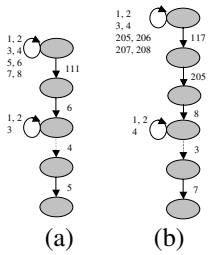

*Figure 3. Asynchronous product for the Dosing tank, where C for 'closed', O for 'open', L for 'low', H for 'high' respectively*




(a) (b)

*Figure 4. Controllers for filling tank up to level 2 (a) and level 1 (b) without pre-check*

Adding pre-checks to ensure that the initial state is consistent with the nominal initial state gives the close loop language of Figure 5(a), where $\sigma_x = 111$ is defined as the starting event of this controller, which we call $C_x$. A similar procedure to fill up to level 1 can be obtained (Figure 4 (b)) which uses the same valve and sensor devices. The closed loop language of two parallel controllers with pre-check is shown in Figure 5(a) and (b). The regulator language for the concurrent use of the two parallel controllers is shown in Figure 6. It can now be formally established that the two controllers are non-cooperative and hence $C_x$ must inhibit $C_y$ and vice versa.




(a) (b)

*Figure 5. FSM with pre-checks for two controllers in parallel*

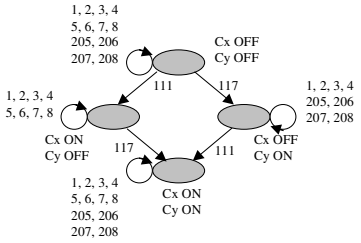



*Figure 6. Regulator language for controllers with shared controllable items*

## 6. Conclusion

The novel theoretical development presented allows incorporating controller initiation and controller inhibiting features within a PCT framework. These two mechanisms guarantee essential safety properties of a multitasking sequential control system, prohibiting the start of control actions in inappropriate situations. The pre–check mechanism ensures that control is initiated only when the state of the process is compatible with predefined nominal initial state(s). Similarly, the inhibit function prevents nominated controllers from starting while a given controller is already active. The formal closed loop language generated by a controller with a pre–check mechanism on a process was derived. This formulation, combined with that for the closed loop language generated by two parallel controllers, yields a general formal definition of the closed loop behavior generated by two parallel controllers each with pre–checks. A further extension in the form of a regulator language handles the special case of shared controllable items. Further extension to multiple parallel controllers is immediate. The example used here is illustrative only. Larger applications will be published elsewhere.

## Nomenclature

$C_x$ and $C_y$ – Controllers
$L(C_x/M)$ – Closed loop language generated by controller $C_x$ and process $M$
$L_m(C_x/M)$ – Closed loop language generated by controller $C_x$ and marked process
$L(C_x \uparrow C_y / M)$ – Closed loop language generated by parallel controllers $C_x$ and $C_y$
$M$ – Process model FSM
$Q$ – Set of states
$Q_m$ – Set of marked states
$q_0$ – Initial state
$q_0^{'}$ – Nominal initial state
$S_x$ – The specification for controller $C_x$
$S_{\Sigma_u}^{X_w}$ – The self-loop operation at wait state with uncontrollable transition $\Sigma_u$
$S_{\sigma_x}^{Q_x}$ – The self-loop operation at state $Q_x$ with transition from $\sigma_x$.
$V^{n_v}$ – The set of states each represented by $n_v$ state variables

**Greek Symbols**

$\gamma$ – State variable transition partial function
$\delta$ – State transition partial function
$\xi$ – Controller state transition partial function
$\Sigma$ – Set of transitions or alphabet of process events
$\Sigma^*$ – The set of all process trajectories composed of transitions in $\Sigma$
$\sigma_x$ – Process event

## Reference


Alsop, N. J. 1996, Formal Techniques for the Procedural Control of Industrial Processes, PhD thesis, University of London

ISA, 1995, Batch Control Part 1: Models and Terminology.

Moon, I., G.J. Powers, J.R. Burch and E.M. Clarke, 1992, Automatic verification of sequential control systems using temporal logic. AIChE, **38**(1), 67.

Nimmo, I., 1993, Start up plant safely, *Chemical Engineering progress* **89**(12), 66.

Sanchez A and S. Macchietto, 1995. Design of procedural controllers for chemical process, Computers and Chemical Engineering, 19S, 381.

Sanchez, A., G. Rotstein, N. Alsop and S. Macchietto, 1999, Synthesis and implementation of procedural controllers for event-driven operations, AIChE J, **45** (8), 1753.

Wonham, W.M., 1996, *Notes on Control of Discrete–Event Systems,* Systems Control Group, Dept. of Electrical & Computer Engineering, University of Toronto. ECE 1636F/1637S.