15th Triennial World Congress, Barcelona, Spain

ANALYSIS OF SAFETY PROPERTIES IN THE SYNTHESIS OF DISCRETE-EVENT CONTROLLERS

A. Sanchez [1] **R. Gonzalez A. Michel**

Dept. of Elec. Eng. and Comp.
Centro de Investigacion y Estudios Avanzados (CINVESTAV)
Apdo. Postal 31-438, Guadalajara 45091, Jalisco, Mexico

Abstract:
An "ad-hoc" formal framework is proposed for the analysis of three types of safety specifications describing the conditional execution of finite sequences of controlled events. The notion of a specification set free of errors and redundancies is introduced as a *minimal set of consistent specifications* as well as procedures to establish it. The satisfiability verification of the specifications by the closed–loop behavior model is also discussed. The use and advantages of the framework are illustrated with the synthesis of a class of discrete-event controller, termed procedural controller, for the operation of an industrial batch chemical reactor. Conflicts on the specification set were easily identified and corrected, reducing the synthesis effort. Satisfiability verification of the specifications by the closed-loop behavior establishes to what extent the controller fulfills the specifications.

Keywords: discrete–event control, formal specification, verification, batch processing

1. INTRODUCTION

Formal synthesis of controllers for event-driven operations using automata-based methods is frequently carried out employing a discrete event model of the process and a set of closed-loop behavior specifications established by the designer. The synthesis goal is to find a discrete-event controller that restricts the closed-loop behavior to a trajectory set fulfilling structural invariants (e.g. controllability) while satisfying maximally the set of specifications. The plant model and the specification set is frequently assumed to be initially free of errors, which in few occasions is the case. Thus, the synthesis task becomes an iterative procedure in which plant or specification models are modified in each iteration, until obtaining a satisfactory result. In most cases, not to mention large systems, it is very difficult to identify errors or inaccuracies in models either with incremental or monolithic specifications. Besides, the synthesis procedure itself can be computationally very expensive and time consuming.

This paper proposes an "ad-hoc" formal framework to facilitate the synthesis of a class of discrete-event controllers for finite trajectories, termed *procedural controllers*, for three types of safety specifications describing the conditional execution of sequences of controlled events. A procedural controller is capable of forcing the execution of controllable transitions by preempting uncontrollable events. Alsop et al. (1996) showed the applicability of these discrete-event controllers in the batch processing industries. The framework helps the designer i) to capture these specifications, ii) to analyze the specification set and initiate the actual controller synthesis with a specification set free of errors and redundancies and iii) to determine what specifications are satisfiable by the obtained closed-loop behavior. Thus, once the controller has been synthesized, the designer can establish to what extent the behavior being specified is part of the closed-loop trajectory set.

The paper presents in section 2 the modeling of a discrete-event process using standard automata as well as the notion of *semitrajectory* as a state-assignment sequence. The definition of procedural controller is also presented together with an explanation of the type of processes in which

[1] Corresponding author. e-mail address: arturo@gdl.cinvestav.mx. Fax: +52 3134 5579

this class of controllers can be used. Section 3 introduces the definition of three types of safety properties realized as semitrajectories with specific semantics. The types considered are: 1) given an operation state, a set of control commands is executed sequentially, 2) given the occurrence of a process event, a set of control commands is executed sequentially, and 3) given an operation state, a set of control commands are disabled. The notions of a *minimal set of consistent specifications* and *satisfiability* by the closed-loop behavior are introduced in sections 4 and 5. The paper then describes, using an example, how to incorporate these ideas in the synthesis of a procedural controller for the filling operation of a batch chemical reactor. The specification set for this operation was generated and thoroughly checked manually by a designer familiar with the reactor operation. Using the chosen synthesis procedure, the obtained controller was manually checked to the designer satisfaction. In a second exercise, using the proposed framework, inconsistencies among the specifications were spotted and, more importantly, it was found that the closed–loop behavior did not satisfy the specification set. The paper closes discussing what are the benefits of using this framework.

2. THE MODELING FRAMEWORK

A discrete–event process of finite trajectories is modeled in the standard fashion by a finite state machine (FSM) $P = \{Q, V, \Sigma, \delta, q_0, Q_m\}$, where

- Q is the state set.
- $V = \{v_1, v_2, \ldots, v_n\}$ is the state–variable set with n, state variable number. A state-variable v_i takes values from a finite domain D_i plus the distinguished symbol ∞ meaning "any value".
- Σ is the transition set, divided into two disjoint sets: Σ_u (uncontrollable transitions) and Σ_c (controllable transitions).
- $\delta : \Sigma \times Q \to Q$, is the state–transition partial function,
- q_0 is the initial state.
- Q_m is the marked states set.

A transition τ is enabled in state q if $\delta(q, \tau)$ is defined.

Definition 1. (Assignment). An assignment over the state-variable set $s : V \to D$ is defined by the rule $v_i \rightarrowtail s(v_i)$ for $i = 1, \ldots, n$, where D is the union of the state-variable domains including the distinguished symbol ∞. The assignment s is represented by an n-tuple $(s(v_1), s(v_2), \ldots, s(v_n))$ for $i = 1, \ldots, n$.

For each state $q \in Q$, an assignment can be associated by a function $\beta : Q \to A$, defined by $\beta(q) = s_q$ where A is the set of all possible assignments. Notice that $\beta(q)$ is not injective, thus it is possible to have two states with the same assignment.

Definition 2. (Non-executable transition set). For an assignment s such that $\beta(q) = s$ for some $q \in Q$, the set of non-executable transitions of s is given by $net(s) = \{\tau \in \Sigma | \tau$ is an enabled transition in any q with assignment s which, by design, is not permitted to occur$\}$.

Definition 3. (Semitrajectory). A semitrajectory of length $m+1$ is a finite mixed sequence of assignments and transitions $\pi = s_0^{\tau_1} s_1^{\tau_2} \ldots^{\tau_m} s_m$. The relationship between assignments and transitions is established by $\gamma : A \times \Sigma \to A$ with the following rule: for $s_{i+1} = \gamma(s_i, \tau)$, there exists exactly one $j \in \{1, 2, \ldots, n\}$ such that $s_{i+1}(v_j) \neq s_i(v_j)$ and for all $k = 1, \ldots, n$ such that $j \neq k$, $s_{i+1}(v_k) = s_i(v_k)$.

Definition 4. (Covering (refinement)). The assignment s covers assignment s' (equivalently, s' refines s) if and only if there exist at least one $j \in \{1, 2, \ldots, n\}$ such that $s(v_j) = \infty$, $s'(v_j) \neq \infty$ and for all $k = 1, 2, \ldots, n$ such that $j \neq k$, $s(v_k) = s'(v_k)$.

Sanchez et al. (1999) introduced the control device used in this work, termed procedural controller. It is modeled as well as an FSM $C = \{X, \Sigma, \gamma, x_0, X_m\}$, where

- X is the state set
- Σ is the same transition set as in the process model
- $\gamma : \Sigma \times X \to X$, is the state–transition partial function.
- x_0 is the initial state
- X_m is the marked states set

In particular, for each $x \in X$, $\sigma \in \Sigma$ such that $\gamma(\sigma, x)$ is defined, one of the following is true:

(1) $\sigma \in \Sigma_u$ and for all $\sigma_c \in \Sigma_c$, $\gamma(\sigma_c, x)$ is undefined.
(2) $\sigma \in \Sigma_c$ and for all $\sigma' \neq \sigma \in \Sigma$, $\gamma(\sigma', x)$ is undefined.

That is, a procedural controller can either be in: 1) a state in which one of a set of uncontrollable transitions occurs or 2) a state in which the execution of the only controllable transition defined is enforced. It is assumed that a procedural controller acting synchronously with a process preempts the occurrence of any uncontrollable transition that can occur from the current process state. Even though this assumption of preemption may appear to be strong, it is largely a modeling issue,

which in particular is, in most cases, true for the type of process plants considered here (i.e. batch processing facilities with relatively slow dynamics and mostly of procedural nature). That is, if a controllable transition has slow dynamics, it can always be partitioned into two transitions: a fast controllable one, representing the execution decision and an uncontrollable transition, representing the slow system response to that decision. Sanchez et al. (1999) presented conditions of existence of a procedural controller and closed-loop invariant properties. The procedural controller FSM is defined in such a way that no decision branching points exist (i.e. there is no need for an external mechanism to decide which controllable transition to execute at a given state) and the closed–loop behavior equates to the controller behavior. Sanchez et al. (2001) argues that this facilitates the refinement and translation of the procedural controller into control code or hardwired logic.

3. SPECIFICATION OF SAFETY PROPERTIES

A safety specification informally states that "something bad" does not happen during execution (e.g. mutual exclusion, deadlock freedom, first-come-first-serve). When dealing with batch processing plants, we have found that most of behavior described by operating procedures used as specifications for designing discrete-event controllers can be described in terms of *forbidden states* and the following three situations:

(1) Given an operation state, a set of control commands is executed sequentially.
(2) Given the occurrence of a process event, a set of control commands is executed sequentially.
(3) Given an operation state, a set of control commands are forbidden to occur.

The formalization of these three types of safety properties as semitrajectories are introduced in the following paragraphs.

Definition 5. (Type 1 Safety Specification). Semitrajectory $\pi = s_0^{\tau_1} s_1^{\tau_2} \ldots^{\tau_m} s_m$ of length $m+1$ with $m > 0$ models a type 1 specification. s_0 represents the *initial assignment* of π.

Definition 6. (Type 2 Safety Specification). Semitrajectory $\pi = s_0^{\tau_1} s_1^{\tau_2} \ldots^{\tau_m} s_m$ of length $m+1$ with $m > 0$ models a type 2 specification if and only if s_0 covers s_1.

$\gamma(s_0, \tau)$ is used as the initial assignment s_1 from where the rest of π is executed. This semitrajectory is a compact way of representing the triggering by a transition of a sequence. This type of semitrajectory will be written as $\pi = s_0^{\wedge\tau_1} s_1^{\tau_2} s_2^{\tau_3} \ldots s_m$, where expression $\wedge\tau_1$ indicates the emphasis over the execution of the event

Definition 7. (Type 3 Semitrajectory). A semitrajectory of type 3 is a two-assignment semitrajectories of the form $\pi = s_0^{\tau} s_1$, where s_0 and s_1 are fixed assignments whereas τ can be any transition such that $\gamma(s_0, \tau) = s_1$ and τ does not belong to set $net(s_0)$. s_0 is the initial assignment. This type of semitrajectory will be written as $\pi = s_1^{\vee\tau} s_2$

4. THE MINIMAL SET OF CONSISTENT SEMITRAJECTORIES

Definition 8. (Consistent semitrajectories). Semi–trajectories π_1 and π_2 are consistent if and only if one of the following is true:

(1) Initial assignments of π_1 and π_2 are not equal and do not cover each other.
(2) If condition 1 is not true and semitrajectories are not of type 3, then they must coincide after the initial assignments.
(3) If condition 1 is not true and only one semitrajectory declares the execution of transition τ from its initial assignment, then the other semitrajectory must not forbid the execution of such transition.

Definition 9. (Duplicate semitrajectories). Semi—trajectories π_1 and π_2 are duplicated if and only if their initial assignments are

(1) either equal or one assignment covers the other and
(2) either coincide after the initial assignment or, for specifications of type 3, at least one transition of $net(s_0)$of π_1 belongs to $net(s_0)$ of π_2.

Definition 10. A set of consistent semitrajectories is a set in which all semitrajectories are mutually consistent.

Definition 11. A minimal set of consistent semitrajectories is a set of consistent semitrajectories without duplications.

A minimal set of consistent semitrajectories can be obtained by direct application of definitions 8 and 9. Computational procedures were implemented to carry out the required matching calculations.

5. SEMITRAJECTORY SATISFIABILITY IN THE CLOSED–LOOP MODEL

A semitrajectory $\pi = s_0^{\tau_1} s_1^{\tau_2} \ldots^{\tau_r} s_r$ of length $r+1$ with $r > 0$ is accepted by an FSM $M_\pi = \{X, V, \Sigma, \rho, x_0, X_m\}$ where

- X, state set.
- V is the state variable set of the process model with assignments given by $\beta(x_i) = s_i$ for $i = 1, \ldots, r+1$.
- Σ is the transition set of the process model.
- $\rho : X \times \Sigma \to X$ such that if $\gamma(s_i, \tau)$ is defined in π, then $\rho(x_i, \tau)$ is defined in M_π.
- The initial state x_0 contains the initial assignment of π.
- $X_m = \{x_r\}$ is the set of marked states.

A semitrajectory π accepted by an FSM M_π is satisfiable by a closed-loop FSM $M_{CL} = \{C, V, \Sigma, \eta, c_0, C_m\}$ if and only if

- Each state-assignment $\beta(x_i)$ of state x_i in M_π is equal to or covers at least one state-assignment $\beta(c)$ of M_{CL}.
- If $\rho(x_i, \tau)$ is defined, then $\eta(c, \tau)$ is defined.

That is, the FSM M_π fits at least once into the closed-loop FSM M_{CL}. Thus, semitrajectory satisfiability by the closed-loop model can be established by using a standard searching strategy. The current computer procedure implementing the verification of specification satisfiability outputs whether a given specification is satisfied by the closed-loop model. Otherwise, it is indicated what condition was not fulfilled.

6. USING SEMITRAJECTORIES IN PROCEDURAL CONTROLLER SYNTHESIS

When synthesizing discrete-event controllers using explicit calculations it is common practice to model both the discrete-event process and the specification set as FSMs in such manner that their synchronous product is used as the basis for calculating a maximal state-transition structure (e.g. the supremal controllable language in supervisory control theory proposed by Ramadge and Wonham (1987) or the maximal controller superstructure in the case of procedural controllers proposed by Sanchez et al. (1999)). From this structure, a control device is obtained, which is minimally restrictive with respect to the specifications and fulfills closed-loop invariants. As mentioned in section 1, in our experience, when synthesizing a controller it is common to spot errors and inaccuracies in both the process and specification models. Thus, the synthesis procedure becomes an iterative task in which process and specification models are modified in each iteration until obtaining a satisfactory result. In particular, when synthesizing controllers for large systems it is common to declare the desired closed–loop behavior using a set of FSM specifications. By using the minimal set of consistent specifications, the synthesis effort can be reduced substantially because this set is free of internal inconsistent behavior and does not contain repeated specifications. Thus, if the resultant closed-loop behavior does not satisfy some of the specifications is due only to controllability restrictions. With this information at hand, the designer can decide what to modify (either the process or specification models) and how in order to improve the controller design.

A constructive definition of a synchronization operation is proposed here to be used as part of the synthesis procedure. The compact representation and size of semitrajectory FSMs are exploited to reduce the computation effort employed to carry out this operation.

Let M_p a discrete–event process model and M_π the FSM accepting a assignment sequence induced by a given specification. The composition $M = M_p \parallel M_\pi$ is obtained by the following procedure:

(1) Duplicate M_p
(2) For all $q_p \in Q_p$ such that $\beta(q_p)$ equals or is covered by $\beta(x_{\pi_0})$, do
  (a) For specifications of types 1 and 2:
    (i) If $\delta_\pi(x_{\pi_0}, \tau)$ is defined then make $\delta_p(q_p, \tau) = q_{n_1}$ and $\delta_p(q_p, \tau')$ undefined for all $\tau' \neq \tau$ where q_{n_1} is a new state of M_p with $\beta(q_{n_1}) = \beta(\delta_p(q_p, \tau))$
    (ii) For $i = 1$ to $r-1$, if $\delta_{pi}(x_{\pi_i}, \tau)$ is defined, then do $\delta_p(q_{n_i}, \tau) = q_{n_{i+1}}$ such that $q_{n_{i+1}}$ is a new state in M_p with $\beta(q_{n_{i+1}} = \gamma(\beta(q_{n_i}), \tau)$.
  (b) For semitrajectories of type 3:
    (i) If $\delta_p(q_p, \tau)$ is defined for some $\tau \in net(s_{q_{\pi_0}})$, then change it to undefined.

The complexity of the operation is $O(ab)$, where a and b are the number of states of the process and semitrajectory models, respectively.

7. EXAMPLE

The use of the proposed framework is illustrated with the synthesis of the control logic for the operation of a batch chemical reactor (shown in figure 1) currently installed in a special lubricants factory. The reactor is controlled by a PLC executing the high level operating procedures for normal, abnormal and emergency operations. Here the synthesis of the control logic for normal, abnormal and recovery operations during the filling phase is used as an example. Table 1 includes the process and software components involved in

the operation, together with FSMs modeling their discrete-event behavior. States are identified by numbers as indicated in parenthesis in the respective columns. Uncontrollable transitions are marked with an *. The rest of the transitions are commands that the PLC can use to control the process. All initial and marked states are labeled with (0). The objective of the phase is to feed into the reactor a measured amount of a component from a warehouse location. The filling starts when the operator inputs the volume set point into the PLC and toggles the "start" button (B1) in the operation console. The PLC then sends the commands to open valves FV1 and FV2 and verifies that the valves opened successfully. If this is the case, it issues the command to start pump P1 and initializes the volume totalizer. Once the set amount has been fed to the reactor, the PLC sends commands to reset the totalizer, to switch off the pump and to close valves. During the filling, the operator can stop/restart the operation by toggling button B1. The PLC must be able to stop and resume safely the filling. If any of the valves shuts during the filling, pump P1 must be turned off and the controller must try to open the offending valve. If the procedure does not succeed or the operator presses the emergency button, the controller blocks any possible action of the operator, then issues the command to stop the pump and shuts any valve still open. Once the emergency button is pressed, the operation is aborted.

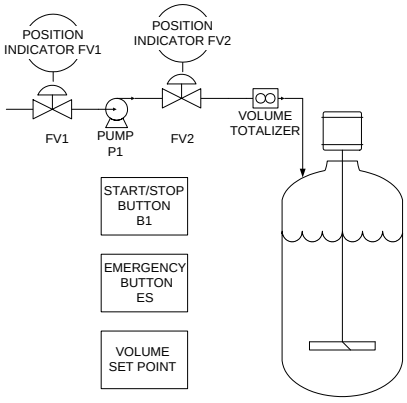

Fig. 1. Diagram of batch reactor

The designer captured the operation described above using 12 semitrajectories which were exhaustively checked by hand. The semitrajectories are shown in table 2. Notice that all specifications start with an assignment (i.e an operational state). Semitrajectories 1 and 7 which are specifications of type 2. The rest are of type 1. They are classified in three categories: normal operation, emergency operation and recovery procedures. For each specification, a natural language statement is given, followed by the associated semitrajectory.

Compnt	Transitions			
	Lbl	Description	from st.	to st.
B1 button	11*	switchOn	off (0)	on (1)
	12	switchOff	on (1)	off (0)
emerg. stop button	21*	switchOn	off (0) on (1)	on (1) off (0)
vol. set pt.	31	FeedSP	NotInic (0)	Inic (1)
	32	NoSP	Inic (1)	NotInic (0)
vol. tot flag	41*	volOK	clear (0)	OK (1)
	42*	clear	OK (1)	clear (0)
FV2 valve	51	openFV2	closed (0)	open (1)
	52	closeFV2	open (1)	closed (0)
FV2 pos. ind.	61*	FV2opens	closed (0)	open (1)
	62*	FV2closes	open (1)	closed (0)
FV1 valve	71	openFV1	closed (0)	open (1)
	72	closeFV1	open (1)	closed (0)
FV1 pos. ind	81*	FV1opens	closed (0)	open (1)
	82*	FV1closes	open (1)	closed (0)
pump status	91	startPmp	off (0)	strtng (1)
	92*	pmpStarts	strtng (1)	on (3)
	93	stopPmp	on (3)	stppng (2)
	94*	pmpStops	stppng (2)	off (0)

Table 1. Elementary components of reactor and their associated FSM models (* = uncontrollable transition).

In order to present these semitrajectories in a concise manner, only the first state assignment is shown. For subsequent states, it is indicated only what state variable number changed and to what value after the execution of the indicated transition. The process model was built as discussed in Sanchez (1996) and the synthesis of the procedural controller was then carried out using the available tools. The sizes of the process model and resultant procedural controller was of 1024 and 124 respectively. Satisfaction of each semitrajectory by the controller was then manually verified to the designers satisfaction. Before translating the resulting FSM into programming code, a second verification round was carried out using the framework proposed here. It was found that the specification set was inconsistent and not minimal. Duplication was spotted in semitrajectory 2 and 11. Both specification command to start the pump when the process is ready for the filling. The duplicate behavior was attributed in this case to an oversight of the designer. More importantly, it was found that semitrajectories 8 and 9 were inconsistent. Their initial assignments did not fulfil definition 4. That is, state-variable 3 was covered in specification 8 while it was refined in specification 9 and state-variables 5 and 7 were covered in specification 9 while they were refined in specification 8. This caused that there were 8 state assignments being shared by both variable sets which lead to the inconsistent behavior. In our experience, this is a common situation during the design of a controller. Exhaustive or systematic analysis (e.g. HAZOP, FMCA) that could spot this type of problems are not frequently carried out, unless regulations demand it.

Normal Operation
1. If button B1 is toggled, then PLC issues commands to open FV2 and FV1 $(0,0,0,0,0,0,0,0,0) \wedge^{11} (1:1)^{51}(5:1)^{71}(7:1)$
2. Once valves are open, then PLC issues command to start pump $(1,0,0,0,1,1,1,1,0)^{91}(9:1)$
3. If pump is successfully started, then the volume set point is feed to the totalizer $(1,0,0,0,1,1,1,1,3)^{31}(3:1)$
4. Once volume amount is reached, B120 is toggled, set point flag returns to not initialized and the PLC issues commands to close valves and stop pump. $(1,0,1,1,1,1,1,1,3)^{12}(1:0)^{32}(3:0)^{52}(5:0)^{72}(7:0)^{93}(9:2)$
Emergency Operation
5. If FV2 shuts while filling up, then the PLC issues the command to stop pump $(1,0,\infty,0,1,0,\infty,\infty,3)^{93}(9:2)$
6. If FV1 shuts while filling up, then the PLC issues the command to stop pump $(1,0,\infty,0,\infty,\infty,1,0,3)^{93}(9:2)$
7. If emergency stop is activated, then button B1 is freed and operation cannot start again $(1,1,\infty,\infty,\infty,\infty,\infty,\infty,\infty) \wedge^{12} (1:0)$
8. Once the emergency stop has been activated and the filling was under way, then the PLC must issue commands to close valves FV2 and FV1 $(0,1,\infty,0,1,\infty,1,\infty,0)^{52}(5:0)^{62}(6:0)$
9. Once the pump is off, clear the volume set point $(0,1,1,0,\infty,\infty,\infty,\infty,0)^{32}(3:0)$
10. If emergency stop is activated and pump is on, then the PLC must issue a command to stop pump $(0,1,\infty,0,\infty,\infty,\infty,\infty,3)^{93}(9:20)$
Recovery procedures
11. After restarting normal operation, valves were opened successfully and pump is off, then the PLC must issue a command to start pump $(1,0,\infty,0,1,1,1,1,0)^{91}(9:1)$
12. After restarting normal operation, if valves were closed, then the PLC issues the commands to open them $(1,0,1,0,0,0,0,0,0)^{51}(5:1)^{61}(6:1)$

Table 2. Semitrajectories for filling phase

By verifying satisfiability of the specification set, it was found that specifications 9 and 12 were not satisfied by the closed-loop model . In the case of specification 9, this was due to the inconsistency detected previously. For specification 12, it was found that it was not possible to reach in a controllable manner a state with the assignment declared as initial in the specification. In a second exercise, specifications 11 and 12 were eliminated from the specification set. Specifications 9 was modified as shown in table 3 to avoid shared state assignments with specification 8. Using the modified specification set (minimal and consistent), the newly synthesized controller was the same as the previous controller. All specifications in the minimal set were satisfied by the closed–loop model.

9. Once the pump is off, clear the volume set point $(0,1,1,0,0,\infty,0,\infty,0)^{32}(3:0)$

Table 3. Modified semitrajectory for specification 9

8. CONCLUSIONS

Obtaining the same results in both exercises indicates that, although the controller guarantees a safe (i.e. controllable) closed-loop behavior, it was not known what the controller was actually controlling. From a practical point of view, this is not acceptable for any real process. Although it is not essential for the controller synthesis to debug the specification set to a minimal and consistent one, it was very useful for the better understanding of the controller role.

References

N. Alsop, L. Camillocci, A. Sanchez, and S. Macchietto. Synthesis of procedural controllers - Application to a batch plant. *Computers and Chemical Engineering*, 20(Suppl.):S1481–S1486, 1996.

P. J. Ramadge and W. M. Wonham. Modular feedback logic for discrete event systems. *SIAM Journal of Control and Optimization*, 25(5): 1202–1218, 1987.

A. Sanchez. *Formal Specification and Synthesis of Procedural Controllers for Process Systems.* Lecture Notes on Control and Information Sciences, v. 212, Springer–Verlag, 1996.

A. Sanchez, G. Rotstein, N. Alsop, and S. Macchietto. Synthesis and implementation of procedural controllers for event–driven operations. *AIChE Journal*, 45(8):1753–1775, 1999.

A. Sanchez, G. E. Rotstein, N. Alsop, J. P. Bromberg, C. Gollain, S. Sorensen, S. Macchietto, and C. Jakeman. Improving the development of event-driven control systems in the batch processing industry. A case study. *ISA Trans.*, 2001. In press.

Acknowledgements. Partial financial support from CONACYT (grant 31108U) and the use of the application example from Interlub S.A. are kindly acknowledged.