Accepted for *8th European Symposium on Computer Aided Process Engineering*, Brugge, Belgium, May 24-27, 1998.
Will appear in *Computers and Chemical Engineering*, Elsevier. Corrected version.

Using Model-Checking for Timed Automata to Parameterize Logic Control Programs

Stefan Kowalewski and Sebastian Engell

Process Control Group, Chemical Engineering Dept., University of Dortmund,
D-44221 Dortmund, Germany, Email: {stefan | engell}@ast.chemietechnik.uni-dortmund.de

Ralf Huuck, Yassine Lakhnech, Ben Lukoschus and Luis Urbina

Institute of Computer Science and Applied Mathematics, Christian-Albrechts-University,
Preußerweg 1-9, D-24105 Kiel, Germany, Email: {rhu | yl | bls | lu}@informatik.uni-kiel.de

Abstract
In this contribution we describe how the modeling and analysis framework of Timed Automata can be used to determine valid parameter ranges for timers in logic control programs. The procedure is illustrated by means of a simple process engineering example for which the complete Timed Automata model is presented. To analyse the model, the tool HyTech is used which provides routines to determine values of model parameters depending on reachability conditions.

Keywords: Programmable Logic Controllers, process safety, validation, Timed Automata, HyTech.

INTRODUCTION

When Programmable Logic Controllers (PLCs) are used in process control, often the problem arises that the correct timing of discrete control actions depends on the continuous dynamics of the controlled plant. Consider for example an exothermal reaction. There, the admissible response time between the indication of a temperature alarm and switching on the emergency cooling or injecting a reaction stopper is given by the reaction dynamics and the performance of the cooling or the stopper. Whereas in this example it is merely required that the control action has to take place as soon as possible, situations may arise in which two or more dynamical phenomena "race against each other" and the controller must neither react too late nor too soon (an example for such a problem is given below). In such cases timers have to be introduced in the PLC program to specify certain response times. It is clear that the choice of these times is crucial for the correct plant operation and possibly even for plant safety.

In this contribution we consider the problem of finding valid parameters for timers in PLC programs in situations as described above. For this purpose, we take advantage of a relatively new modeling and analysis method with corresponding tools from Computer Science which was developed originally for the verification of computer protocols or hardware designs with real-time requirements. The basis of the approach is the *Timed Automaton* paradigm (Alur and Dill, 1994). Timed Automata are an extension of the common finite automaton model and include real variables representing *clocks*. Clocks can be reset to zero by discrete transitions of the automaton and then increase with a fixed rate of one per time unit. The discrete transitions can be made dependent on clock standings and can be forced by means of so-called *invariants*. Invariants are predicates over clock variables which are assigned to a discrete state and have to be true while the system is in this state. Several Timed Automata can run in parallel, interacting via shared variables and synchronization of events. There are computer tools available for the analysis of Timed Automata models, for example KRONOS (Olivero and Yovine, 1993), UppAal (Bengtsson et al., 1995), and HyTech (Henzinger et al., 1996). While all of these tools offer basic operations like composition of subsystems and reachability analysis (Henzinger et al., 1994), only HyTech can perform parametric analysis, i.e. determine the range of clock variables for which a specific system property (e.g. reachability of certain discrete states) holds. Since this is exactly the kind of analysis which is needed when the valid parameter ranges for timers in control programs have to be determined, we chose HyTech for our application.

The paper is organized as follows. In the next section we briefly sketch the steps which are necessary to apply the Timed Automata model to analyse logic control programs for process systems. After that the example and the application of our approach to it are described in detail. We omit a formal introduction to Timed Automata (see for this purpose Alur and Dill, 1994) and illustrate the basic ideas by means of the example. Finally, we present the analysis results and discuss practical aspects of the approach.

METHODOLOGY

Our approach consists of three steps. The first one is to build a Timed Automata model of the plant. This step requires that minimal or maximal durations of the relevant dynamic processes are available (e.g. it is known that it takes maximally 10 seconds from the injection of a reaction stopper until no more reaction heat is produced). If only the continuous dynamics are given in terms of differential equations, this

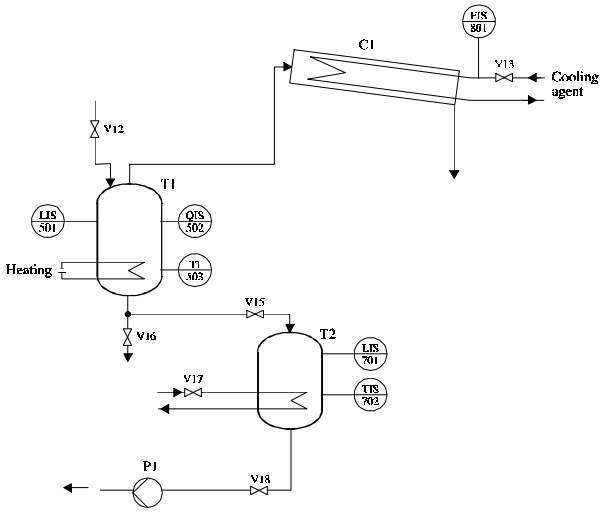
information can be determined by simulation or the Timed Automata model can be derived automatically by approximation techniques (Stursberg et al., 1997). It is important that the plant model represents the "uncontrolled" behavior. This means it has to capture not only the nominal process but also all the disturbances, equipment faults and operator failures which shall be handled by the PLC program under consideration. For the analysis, it is necessary that the undesired consequences of these disturbances etc. are defined in terms of "bad states". HyTech can then compute the parameter ranges for which these states are not reachable.

The second step is to translate the PLC program into a Timed Automaton. This is done by creating a discrete transition system for the logic part and introducing a clock variable for each timer. The connection to the Timed Automata representing the plant is realized by the above mentioned means of shared variables and synchronization of events. Finally, the third step consist of the composition of the plant and the controller model and the determination of the valid ranges for the controller clock variables. It is performed automatically by HyTech.

APPLICATION EXAMPLE

To illustrate the underlying modeling framework and the validation procedure, the approach will now be applied to a small example process which has been used for illustrative purposes before (Kowalewski and Preußig, 1996).

Fig. 1. Flowchart of the example process.



The example process

In the process the following production sequence takes place (see the flowchart in Fig. 1): A solution is filled into tank *T1* and then evaporated until a desired concentration is reached. During evaporation, the condenser *C1* is in operation and captures the steam coming from *T1*. When the desired concentration is reached, the evaporation step is finished, the heating will be switched off and the material is drained from *T1* into *T2* as soon as *T2* becomes empty. A postprocessing step then takes place in *T2*, before the material can be pumped out of *T2* to a subsequent part of the plant. We direct our interest to the problem of an appropriate control reaction on a cooling breakdown in the condenser. This failure can lead to a dangerously high pressure in the condenser tube, if the evaporating is continued. So, the appropriate reaction would be to switch off the heating in *T1*. But then the problem arises that the material in *T1* will become solid after a certain time when it cools down and is not drained into *T2* (for example because *T2* is not empty). This is not desired. Consequently, the controller has to wait a certain time after a cooling breakdown before it can switch off the heating. We have exactly the situation described above: The controller must neither react too early (then the material cannot be drained before it becomes solid) nor too late (then the pressure in the condenser will become too high).

Fig. 2. Control program for the example (excerpt).

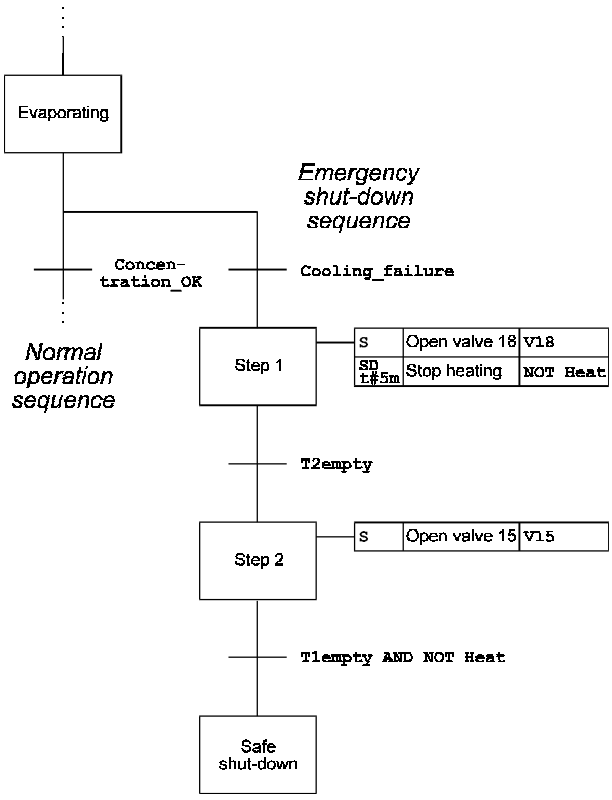


Figure 2 shows a part of the control program for the described process, presented as a so-called Sequential Function Chart (IEC, 1993). After a cooling failure, Tank 2 is drained immediately (*open valve 18*), whereas the heating in Tank 1 is switched off after 5 minutes (`D t#5m` stands for *delayed for 5 minutes*). As soon as Tank 2 becomes empty (*T2empty*), Tank1 is drained (*open valve 18*). Finally, when Tank 1 is empty and the heating is switched off, the controller moves to a step indicating a safe shutdown.

As described above, the critical design parameter for this control program is the waiting time between a cooling failure and switching off the heating. In the following, we will not only investigate whether the choice of 5 minutes is correct but also determine the whole range of the waiting time for which neither the material solidifies nor the pressure becomes too high.

The model

An appropriate model of this example consists of four Timed Automata representing the two tanks (Fig. 3 and 4, resp.), the condenser including the cooling (Fig. 5), and the controller (Fig. 6). The four automata run in parallel, meaning that they can do transitions independently from another wherever this is not restricted by transition conditions on shared variables or by synchronization. Table 1 presents the clocks, the synchronization labels, and the shared variables used in these four models.

Fig. 3. Timed Automata model for tank 1.

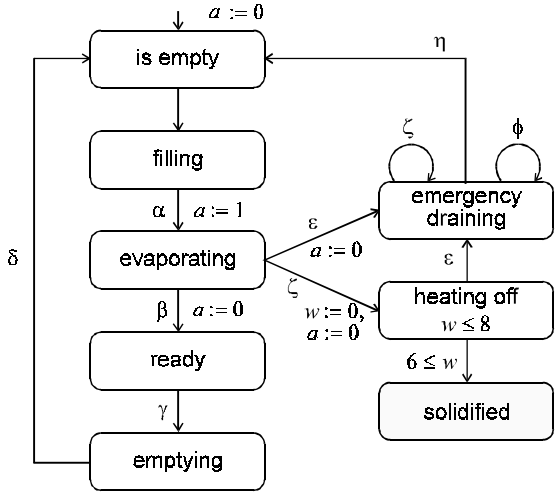


Fig. 4. Timed Automata model for the condenser.

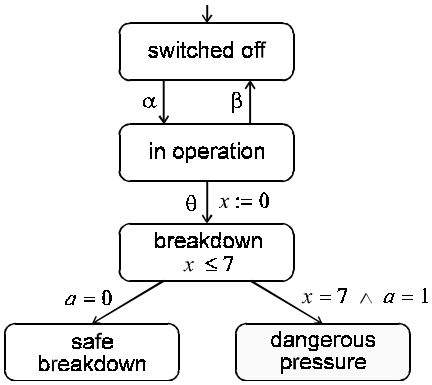


Fig. 5. Timed Automata model for tank 2.

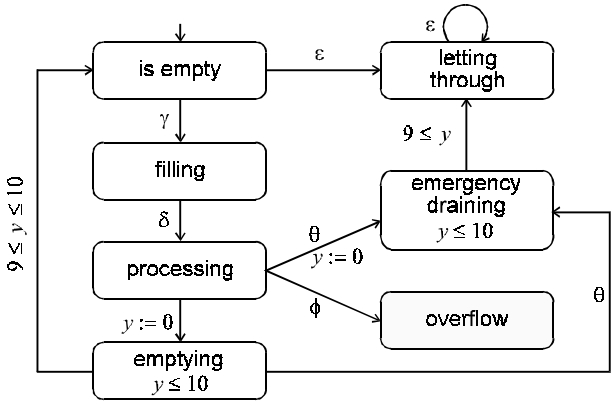


Fig. 6. Timed Automata model for the controller.

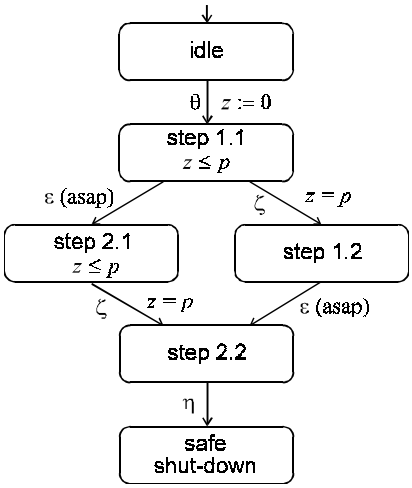


Table 1. Variables and parameters of the models.

Clocks:

w	measures time between heating stop and solidification
x	measures time between cooling failure and reaching dangerous pressure
y	measures time for draining tank 2
z	measures waiting time between cooling failure and switching off the heating

Shared variables:

a	$\in \{0, 1\}$, a=1 $\Leftrightarrow$ Tank 1 is evaporating.

Synchronization labels:

α	Evaporation starts.
β	Evaporation stops.
γ	Filling tank 2 starts.
δ	Filling tank 2 stops.
ε	Emergency draining of tank 1 starts.
ζ	Heating is switched off.
θ	Cooling failure.
ϕ	Overflow happens.
η	Emergency draining of tank 1 stops.

Parameter:

p	Value for waiting time

In the following we shortly explain the model of the Figs. 3 to 6. For each system, the behavior is expressed by (self-explanatory) discrete states and transitions between them. The dependency of transitions on timing constraints is expressed by the clocks. For example, in tank 1, the model describes that the the material will solidify within 6 to 8 time units after the heating is switched off. This is achieved by three expressions: First, the clock w is reset when the transition to the state *heating off* is taken ($w := 0$). Second, the invariant $w \leq 8$ is assigned to *heating off*, expressing that this state has to be left when $w > 8$ becomes true. And finally, the transition from *heating off* to *solidified* is "guarded" by the condition $6 \leq w$ which means it can only be taken when this condition is true.

As mentioned before, the interaction of the four systems is realized by two concepts, shared variables and synchronization labels. For example, the shared variable a is used to express that the transition from *breakdown* to *dangerous pressure* in the condenser automaton can only be taken when tank 1 is still evaporating. So, a is set to 1 when the transition from *filling* to *evaporating* is taken and reset to 0 when *evaporating* is left. The condition $a = 1$ can then be used in the guard of the transition to *dangerous pressure* in the condenser.

Synchronization labels force simultaneous execution of transitions. When two transitions in different automata have the same synchronization label, they can only be taken both at the same time and only if both transitions are possible. If the latter is not the case, one automaton has to wait until the other one is ready for the common transition. For example, the label γ is used to express that filling tank 2 is only possible when tank 1 is emptied and that both are started at the same time.

If transitions are synchronized by a common label and none of them is forced to happen by a corresponding invariant, the simultaneous occurence of all these transitions can take place at any time. Often, it has to be described that one of these transitions will be executed immediately after its enabling and that it will force the other ones to occur. To express this situation, the flag "asap" (for *as soon as possible*) can be assigned to the "forcing" transition. In the evaporator, starting the draining of tank 1 by the ε-labeled controller transitions in Fig. 6 is such an example.

Analysis results

The input for the analysis consist of two parts, first, the model and, second, an analysis objective. In our case, the objective is formulated by a set of forbidden states. In particular, we define the states *solidified* in tank 1, *dangerous pressure* in the condenser and *overflow* in tank 2 as undesired (symbolized by the grey shading in Figs. 3 - 5). Feeding this into HyTech's parametric analysis leads to the result that these three states are reachable when $0 \leq p \leq 4$ or $7 \leq p$ is true for the parameter p. Consequently, the safe value range for the waiting time is $4 < p < 7$. The computation took approximately 2 seconds on a Pentium P166 PC.

DISCUSSION

We have applied the Timed Automaton framework and the analysis tool HyTech to the problem of finding safe, or generally speaking, valid parameter values for timers in PLC programs controlling processing systems. It has been shown that the approach is eligible to solve this problem by applying it to a simple process engineering process.

However, two shortcomings have to be stated. First, the modeling effort is huge since the model has to capture not only the nominal behavior but also failures and disturbances. Second, whereas the computational effort for the presented example is negligible, this situation changes rapidly when the models become larger. In fact in general it grows exponentially with the number of components, states, clocks and shared variables.

For these two reasons, the approach seems to be justified only in such cases where the correct operation of the logic controller is critical (e.g. safety related functions) and where the worst cases are not determinable directly. If it is clear what the worst cases are, simulation of continuous models with switching seems to be a more viable option.

ACKNOWLEDGMENTS

This work is supported by the *German Research Council* (*DFG*) in the special program *Continuous-Discrete Dynamics of Technical Systems* (*KONDISK*) under the grants Ro 1122/2 and En 152/19. We thank *Howard Wong-Toi* for his valuable comments to the evaporator model. Helpful discussions with *Jörg Preußig* and *Olaf Stursberg* are also acknowledged.

REFERENCES

Alur, R., and Dill, D. L., 1994, A Theory of Timed Automata. *Theor. Comp. Science* **126**, 183-235.

Bengtsson, J., Larsen, K., Larsson, F., Pettersson, P., and Yi, W., 1995, UppAal User Guide. `http://www.docs.uu.se/docs/rtmv/uppaal/`

Henzinger,T.H., Ho, P.H., and Wong-Toi, H., 1996, A User guide to HyTech. `http://www-cad.EECS.Berkeley.EDU/~tah/HyTech/`

Henzinger, T.H., Nicollin, X., Sifakis, J., and Yovine, S., 1994, Symbolic Model Checking for Real-Time Systems. *Inform. and Computation* **111**(2), 193-244.

International Electrotechnical Commission, 1993, International Standard 1131 *Programmable Logic Controllers, Part 3: Programming Languages*, Geneva.

Kowalewski, S. and Preußig, J., 1996, Verification of sequential controllers with timing functions for chemical processes. Preprints *IFAC 13th World Congress*, San Francisco, USA, Vol. J, 419-424.

Olivero, A., and Yovine, S., 1993, Kronos: A Tool for Verifying Real-time Systems - User's Guide and Reference Manual. `http://www.imag.fr/VERIMAG/TEMPORISE/kronos/`

Stursberg, O., Kowalewski, S. and Engell, S., 1997, Generating Timed Discrete Models of Continuous Systems. Proc. *2nd MATHMOD*, Vienna, 203-210.

APPENDIX: HyTech input file (not included in the published paper)

```
-- Evaporator example for ESCAPE 8
-- ----------------------------------------
-- Stefan Kowalewski, 5.11.97, revised 14.1.98

var
    solid_clock,        -- w
    press_clock,        -- x
    empty_clock,        -- y
    control_timer       -- z
        : clock;
    tank1_evaporating   -- a
        : discrete;
    waiting_time
        : parameter;    -- p

-- ------------------------------------------------------------------------------------------

automaton tank1
synclabs : evap_start,       -- alpha
           evap_stop,        -- beta
           fill2_start,      -- gamma
           fill2_stop,       -- delta
           emer_drain_start, -- epsilon
           heat_stop,        -- zeta
           overflow_happens, -- phi
           emer_drain_stop;  -- eta

initially is_empty & True;

loc is_empty: while True wait {}
        when True goto filling;

loc filling: while True wait {}
        when True do {tank1_evaporating'=1} sync evap_start goto evaporating;

loc evaporating: while True wait {}
        when True do {tank1_evaporating'=0} sync evap_stop goto ready;
        when True do {tank1_evaporating'=0} sync emer_drain_start goto emergency_draining;
            -- It is assumed that if emergency draining of tank 1 is initiated before the
            -- heating is switched off, the danger of pressure increase in the condenser
            -- will no longer exist.
        when True do {solid_clock'= 0, tank1_evaporating'=0} sync heat_stop goto heating_off;

loc ready: while True wait {}
        when True sync fill2_start goto emptying;

loc emptying: while True wait {}
        when True sync fill2_stop goto is_empty;

loc emergency_draining: while True wait {}
        when True sync overflow_happens goto emergency_draining;
        when True sync heat_stop goto emergency_draining;
        when True sync emer_drain_stop goto is_empty;

loc heating_off: while solid_clock<=8 wait {}
        when True sync emer_drain_start goto emergency_draining;
        when solid_clock>=6 goto solidified;

loc solidified: while True wait {}

end

-- ------------------------------------------------------------------------------------------

automaton cooling
synclabs : evap_start,       -- alpha
           evap_stop,        -- beta
           cooling_failure;  -- theta

initially switched_off;

loc switched_off: while True wait {}
        when True sync evap_start goto in_operation;

loc in_operation: while True wait {}
        when True sync evap_stop goto switched_off;
        when True do {press_clock'= 0} sync cooling_failure goto breakdown;

loc breakdown: while press_clock<=7 wait {}
        when press_clock=7 & tank1_evaporating=1 goto dangerous_pressure;
        when tank1_evaporating=0 goto safe_breakdown;

loc dangerous_pressure: while True wait {}

loc safe_breakdown: while True wait {}

end

-- ------------------------------------------------------------------------------------------
```

```
automaton tank2
synclabs : fill2_start,      -- gamma
           fill2_stop,       -- delta
           cooling_failure,  -- theta
           overflow_happens, -- phi
           emer_drain_start; -- epsilon

initially is_empty;

loc is_empty: while True wait {}
        when True sync fill2_start goto filling;
        when True sync emer_drain_start goto letting_through;

loc filling: while True wait {}
        when True sync fill2_stop goto processing;

loc processing: while True wait {}
        when True do {empty_clock'= 0} goto emptying;
        when True do {empty_clock'= 0} sync cooling_failure goto emergency_draining;
        when True sync overflow_happens goto overflow;

loc emptying: while empty_clock<=10 wait {}
        when empty_clock>=9 goto is_empty;
        when True sync cooling_failure goto emergency_draining;

loc letting_through: while True wait {}
        when True sync emer_drain_start goto letting_through;

loc emergency_draining: while empty_clock<=10 wait {}
        when empty_clock>=9 goto letting_through;

loc overflow: while True wait {}

end

-- ------------------------------------------------------------------------------------------

automaton controller
synclabs : cooling_failure,  -- theta
           emer_drain_start, -- epsilon
           heat_stop,        -- zeta
           emer_drain_stop;  -- eta

initially idle;

loc idle: while True wait {}
        when True do {control_timer'= 0} sync cooling_failure goto step_one_one;

loc step_one_one: while control_timer<=waiting_time wait {}
        when asap sync emer_drain_start goto step_two_one;
        when control_timer=waiting_time sync heat_stop goto step_one_two;

loc step_two_one: while control_timer<=waiting_time wait {}
        when control_timer=waiting_time sync heat_stop goto step_two_two;

loc step_one_two: while True wait {}
        when asap sync emer_drain_start goto step_two_two;

loc step_two_two: while True wait {}
        when True sync emer_drain_stop goto safe_shut_down;

loc safe_shut_down: while True wait {}

end

-- ------------------------------------------------------------------------------------------

var init_reg, final_reg, reached: region;

init_reg := loc[tank1] = is_empty &
            loc[cooling] = switched_off &
            loc[tank2] = is_empty &
            loc[controller] = idle &
            tank1_evaporating=0;

final_reg := loc[tank1] = solidified |
             loc[cooling] = dangerous_pressure |
             loc[tank2] = overflow;

reached := reach forward from init_reg endreach;

-- print trace to final_reg using reached;

print omit all locations
         hide non_parameters in
              reached & final_reg
         endhide;
```