On the precise and efficient representation of industrial controllers in Modelica

Alberto Leva[1]

[1]DEIB, Politecnico di Milano, Italy alberto.leva@polimi.it

Abstract

This paper highlights some still-open issues in the Modelica representation of industrial controllers, particularly when simulation efficiency must balance with fidelity to their real-world implementation as digital components of cyber-physical systems. The treatise focuses on PID controllers due to their predominant role, though the ideas readily extend to virtually any other control structure. The aim is to stimulate discussion around this important yet often overlooked topic, while also suggesting directions for future development.

Keywords: Modelica, industrial controllers, PID controllers, digital control, digital twins, cyber-physical systems.

1 Introduction

Over the last decades, the Modelica ecosystem has grown to embrace a wide range of domains where control systems play a fundamental role. In many of those domains – process systems being a notable example – control blocks are combined to form articulated structures (Skogestad 2023). These structures are then deployed onto heterogeneous hardware/software architectures, resulting in complex cyber-physical systems. The *realistic* simulation of such systems is still a non completely solved open problem, as conventional "textbook" controller models fall short in terms of completeness, flexibility, and – if fidelity in the digital part matters – of computational efficiency (Cimino et al. 2023).

In the following we discuss the problem just evidenced, referring to the PID controller due to its generality and diffusion — though the underlying ideas are evidently general. After the brief motivating example in Section 2, Section 3 deals with completeness, illustrating the I/O structure required to manage a control block in a realistic application setting; Section 4 describes the various forms that a linear (PID) control law can take, while Section 5 addresses nonlinear functionalities like saturation management and shows how differently they can be realised.

The compound of the above should give the reader an idea of how flexible a controller model should be in order to put the analyst in the position of representing what will be really implemented, precisely enough and with an acceptable effort. Section 6 spends just a very few words on the important problem of simulating a control block accounting for its digital realisation, i.e., of balancing fidelity with computational efficiency. Finally, Section **??** draws some conclusions and sketches out the desired effects of the discussion we aim to trigger.

Throughout the paper, key design decisions are highlighted. While some recommendations are offered, no definitive solutions are proposed, as the complexity and criticality of the overall problem make it difficult to predict the long-term consequences of present choices. The aim of this work, as just said, is in fact primarily to foster informed discussion within the community, so as to promote thoughtful and effective decisions going forward.

2 A motivating example

First of all, before entering the problem, the reader might ask whether we really need to bother. The answer could take more than one paper, but suffice a toy example. Consider a control loop with a process and a PI controller, respectively described by

$$P(s) = \frac{\mu}{(1+sT_1)(1+sT_2)}, \quad C(s) = K\left(1+\frac{1}{sT_i}\right) \quad (1)$$

with $\mu = 2$, $T_1 = 2$, $T_2 = 0.25$, $K = 2$, $T_i = 2$. Build for it five simulation models, all fully continuous-time, with different antiwindup strategies: actuation error feedback, also known as integral recomputation, with tracking time T_t (denoted by `aerr1` for $T_t = 2.5$ and `aerr2` for $T_t = 0.5$); conditional integration (denoted by `cint`); clamping (`clamp`), internal feedback (`ifb`).

In the absence of saturation all models behave exactly the same, as expected; we omit plots for brevity. We conversely show in Figure 1 what happens of the controlled variable (`PV_`$\star$) vs. the set point (`ref`) and of the control signal (`CS_`$\star$) in the presence of [0,1] saturation. It is evident that realism requires the simulated antiwindup mechanism to match the implemented one.

Antiwindup aside, several PID functionalities are managed by industry-standard controllers in heterogeneous ways, often involving additional inputs and outputs, both modulating and logic, with respect to the minimal interface of a controller. And to top, all of the above intertwines with the form chosen for the PID control law, as well as with its realisation as digital algorithm.

Summing up, if industrial realism is required, there is no single "reference" controller model implementation —

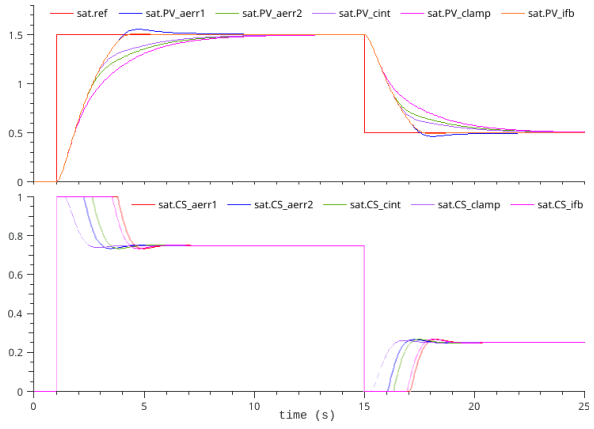


Figure 1. Motivating example: set point and controlled variable (top), control signal (bottom).

but at the same time, attempting to replicate the whole zoo of available products would not be practical. Indeed, a systematic design approach needs devising.

3 Interface and functionalities

Figure 2 illustrates a typical set of I/O signals for an industrial (PID) controller, adopting the Modelica colour coding, together with a brief description of their meaning. These signals refer to functionalities that are crucial for using PIDs in control structures; a few examples follow.

First, without a Bias input one cannot realise feedforward compensation, decoupling, or any structure where the output of a controller must receive an additive correction, as failure to account for that correction in the computation of the controller state may cause highly undesired behaviours. A notable case is backward decoupling, illustrated in Figure 3 by comparing the "textbook" model (top) and one in which the additive correction is duly introduced *upstream* saturation management (bottom).

Second, one cannot manage saturation in cascade controls without increment and decrement locks, as the saturation values for the inner controller come from the actuator, while those for the outer controller are substantially unknown. Assuming all gains positive for simplicity, one has to respectively connect the HI and LO saturation signals from the inner controller to the increment and decrement locks F+ and F- of the outer controller, so as to prevent it from pushing the inner one further into saturation if this occurs.

Third, and again about cascade controls, if the inner controller is set to tracking mode, the outer one must be forced into tracking as well. More in general, whenever a modulating control block is set to operate in any mode but automatic, all the blocks whose loops are consequently open cannot be in automatic mode (particularly if they are not asymptotically stable, as is the case when integral action is present). Enforcing this constraint requires logics to conveniently manipulate the TS and TR signals of the involved blocks. Figure 4, where the notation should be self-explanatory, provides an example of how the second and third point can be handled; variants are possible but discussing them is not our point here.

Fourth, TS and TR are necessary also to manage the switchover among two or more controllers: to avoid undesired bumps in the control signal applied to the rest of the system, all controllers must receive that signal as TR, the active one must be in automatic mode (TS=false) and all the others in tracking mode (TS=true). We omit a block diagram for this case given its simplicity.

More examples could be given – a longer discussion can be found in (Leva 2024) – but the point for this paper is that handling the above matter requires critical model design choices. Should we provide several PID blocks (from "basic" to "complete")? Or just one with flags (named for example hasTracking, has Locks and so forth) to enable/disable the required functionalities, plus the necessary conditional connectors? Or any combination thereof?

4 The LTI control law

The variety of forms or structures that the LTI (linear, time-invariant) control PID control law can take is large indeed (O'Dwyer 2009) but some "families" can be identified; with no exhaustiveness claim we could mention one degree of freedom (1-dof) forms, described as transfer functions $C(s)$ from the error $e := SP - PV$ to the control output CS, i.e.,

$$C(s) = K\left(1 + \frac{1}{sT_i}\right)\frac{1+sT_d}{1+sT_d/N}; \tag{2}$$

non-interacting (aka parallel), i.e.,

$$C(s) = K\left(1 + \frac{1}{sT_i} + \frac{sT_d}{1+sT_d/N}\right); \tag{3}$$

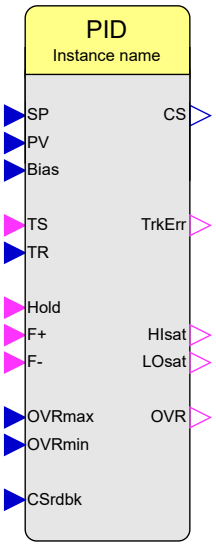
two degree of freedom (2-dof) forms characterised by different $SP \to CS$ and $PV \to CS$ signal paths, most notably the ISA one

$$CS(s) = K\Bigg(bSP(s) - PV(s) + \frac{1}{sT_i}(SP(s) - PV(s)) + \frac{sT_d}{1+sT_d/N}(cSP(s) - PV(s))\Bigg) \tag{4}$$

with the notable "output derivation" case ($c = 0$), as well as other analogous ones realised by pre-filtering the set point signal.

A source of confusion here comes from a certain tendency of the product literature to give different names to the same concept, but this is quite easy to cure with a proper documentation — and here too, a design equilibrium needs finding. More serious problems can arise from the different meanings of the PID parameters, as the reader can immediately guess from the small sample of possible structures in Figure 5.

DOI
10.3384/ecp218603



SP	Set point.
PV	Process (controlled) variable.
CS	Control signal.
Bias	Additive correction to CS (state consistency enforced).
TS	Track Switch: when true makes CS follow TR (state consistency enforced).
TR	Track Reference: followed by CS when TS is true.
TRkErr	Tracking Error: true if TS is true but for any reason (e.g., range) CS is not following TR.
HIsat	True when CS hits the high saturation value.
LOsat	True when CS hits the low saturation value.
Hold	When true causes CS to remain constant (state consistency enforced).
F+	Freeze plus: when true prevents CS from increasing (state consistency enforced).
F-	Freeze minus: when true prevents CS from decreasing (state consistency enforced).
OVRmax	Override max: takes over CS if greater than the computed value (state consistency enforced).
OVRmin	Override max: takes over CS if smaller than the computed value (state consistency enforced).
OVR	True if CS is overridden by OVRmax or OVRmin.
CSrdbk	Actuator readback for some antiwindup strategies (not discussed herein).

Figure 2. Typical I/O connections in an industrial (PID) controller.

In this case the problem will be finding a set of structures as small as possible but capable of accommodating for all the relevant LTI law variants. One structure cannot be enough because some have limitations and some do not (e.g., a series form like the topmost in Figure 5 cannot have complex zeroes), in some forms N modifies the zeroes with respect to the ideal PID form while in some others it does not, and there are other "system-theoretical" reasons to account for.

Furthermore, though in this paper there is not the space for addressing the subject with the required depth, the said set will need to encompass representations of biasing and direct/reverse action, as well as of velocity (as opposed to positional) forms; and moving toward non-LTI facts as we are going to do in the next section, of overriding and control signal rate limits.

5 Nonlinearities

The first item in this list is clearly saturation management or antiwindup, and here the author takes the occasion to suggest removing the adjective "integral" as for windup to occur a controller just needs to have a large enough gain and a slow enough mode with respect to the loop cutoff frequency. The point, in any case, is that here arbitrary choices are inevitable: denoting by $x_c(t)$ be the controller state and writing its output equation as

$$u(t) = c_c x_c(t) + d_c e(t); \tag{5}$$

when $u(t)$ is stuck into saturation, *any* x_c fulfilling (5) is I/O consistent and therefore fits — and notice that the same applies every time $u(t)$ is anyhow "altered" with respect to the value coming from the LTI law (whatever its form), that is with tracking, increment/decrement locks, holding, and so on.

There are other nonlinear functionalities in the list, and they interact with one another, and they could even *conflict* with one another: a few examples follow.

- Shall TR be allowed to violate saturation limits?
- Shall overrides prevail over tracking or *vice versa*?
- Shall increment/decrement locks be subject to tracking/holding or not?

Here too, different products make different choices, sometimes not immediate to figure out if not by thoroughly searching through their documentation. Further design problems are thus at the horizon, as we need to allow the analyst to take knowledgeable decisions because the behaviour of control structures is heavily impacted by facts like those just mentioned.

The author tends to prefer a solution in which the controller blocks do not contain any logic if not that entailed by the definition of I/O signals and their role as per Figure 2, while the coordination of the above functionalities (including the way possible conflicts are solved) is realised externally by means of logic blocks. Such a design approach is in the favour of flexibility and clarity, but on the other hand requires a certain level of user consciousness as well as modelling language knowledge: another point for the community to discuss.

6 Digital realisation

When modelling and simulating a digitally controlled process – or said in more recent terms, when creating and using a digital twin for a cyber-physical system (Lee 2008) – a precise representation of control algorithms is sometimes of relatively low importance but some other times extremely critical (Åström and Murray 2021; Cimino et al. 2024).

To effectively support a model-based design toolchain, especially for complex systems as discussed for exam-

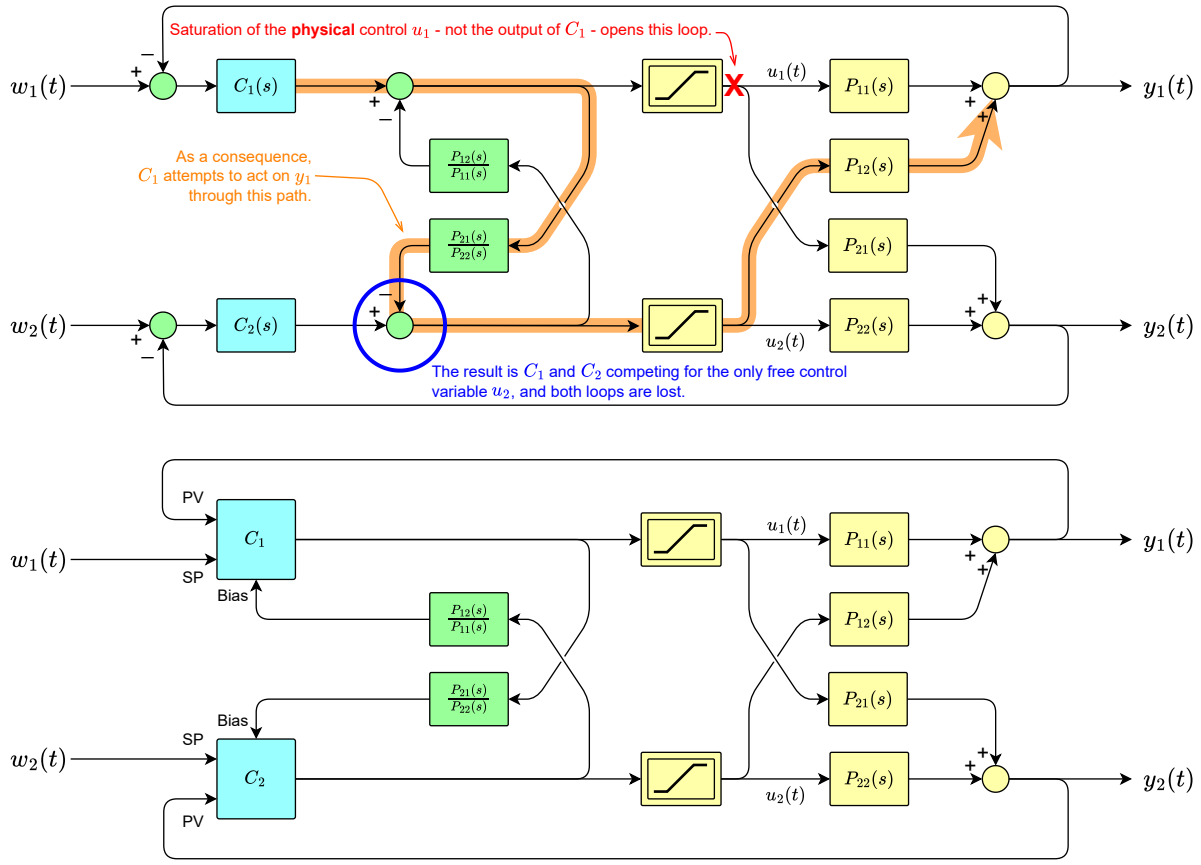


Figure 3. Decoupling control and the need for a Bias input.

ple in (Grieves and Vickers 2017), controller models must therefore be capable of operating sometimes in a time-driven manner, allowing for example for the use of variable-step solvers, and sometimes in an event-based manner, most typically (but not only) to represent the multi-clocked nature of industrial control architectures.

The Modelica language offers multiple means to address such needs, well known to the reader and not central to discuss in this paper. No matter what mix of the said means is adopted, however, the problem remains of structuring a controller model base in such a way that achieving and maintaining consistency between time- and event-driven (in a more industrial jargon, continuous- and discrete-time) models be as natural as possible.

To illustrate the ideas above, we report in Listing 1 a possible minimum set of base classes for Continuous-time (CT) and discrete-time (DT) PI models, and then in Listings 2 through 5 some elementary examples, referring to a 1-dof PI controller with antiwindup realised in the various ways mentioned in the motivating example in Section 2. Incidentally, these are the models used to produce the simulations reported in Figure 1; for the convenience of the reader willing to replicate he experiments and further investigate the matter, all the neces-

Listing 1. Base classes for continuous-time (CT) and discrete-time (DT) PI models.

```
partial model Base_CTPI
  parameter Real K      "gain";
  parameter Real Ti     "integral time";
  parameter Real CSmax "upper saturation value";
  parameter Real CSmin "lower saturation value";

  Modelica.Blocks.Interfaces.RealInput  SP
          "set point";
  Modelica.Blocks.Interfaces.RealInput  PV
          "process (controlled) variable";
  Modelica.Blocks.Interfaces.RealOutput CS
          "control signal";
end Base_CTPI;

partial model Base_DTPI extends Base_CTPI
  parameter Real Ts     "sampling time";
  // computed by algorithms in derived classes
  discrete Real cs;

equation
  cs = CS; // zero-order holder

end Base_DTPI;
```


DOI
10.3384/ecp218603

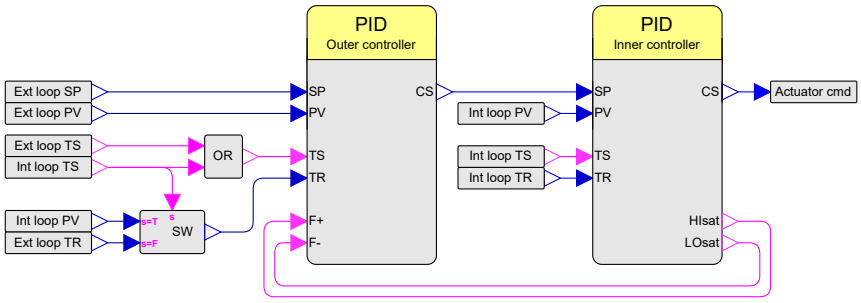


Figure 4. Cascade control and the need for locks and tracking logic (control scheme only, process not shown).

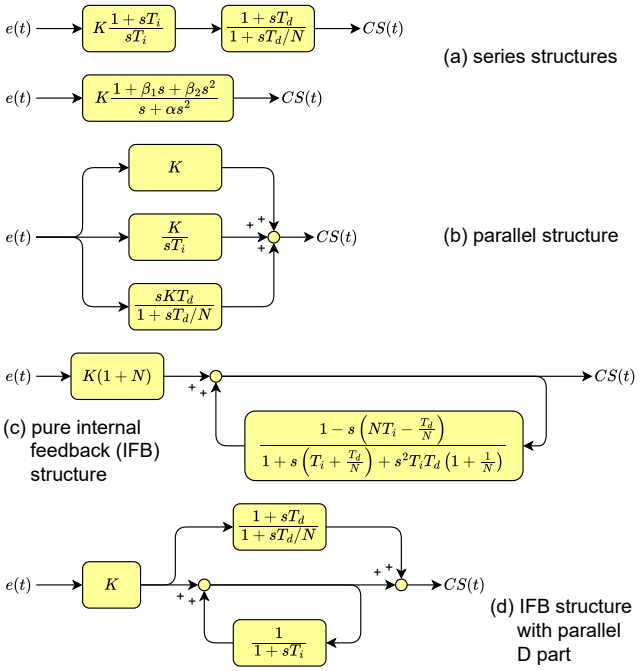


Figure 5. Some structures for the 1-dof LTI PID control law.

Listing 2. CT and DT PI models with actuation error anti-windup.

```
model CTPI_AERR extends Base_CTPI;
  parameter Real Tt;
  Real uns, ui(start=0);

equation
  der(ui) = K/Ti*(SP-PV)-(uns-CS)/Tt;
  uns     = ui+K*(SP-PV);
  CS      = max(CSmin, min(CSmax, uns));

end CTPI_AERR;

model DTPI_AERR extends Base_DTPI;
  parameter Real Tt;
  discrete Real csp, csi, csi1(start=0);

algorithm
  when sample(0, Ts) then
    csp  := K*(SP-PV);
    csi  := csi1+K*Ts/Ti*(SP-PV);
    cs   := max(CSmin,min(CSmax,csp+csi));
    csi1 := csi-(csp+csi-cs)*Ts/Tt;
  end when;

end DTPI_AERR;
```

sary Modelica code is available under a 3-clause BSD licence at `https://github.com/looms-polimi/Modelica_PID_MWEs`.

Coming back to the our main subject, te challenge is to move from extremely simple blocks like those just shown, where most of the problems mentioned above are not addressed, to others capable of representing all the needed functionalities of which we gave a sample in the previous sections. To undertake this challenge we need a methodological and a technological framework, connected to one another as strictly and rigorously as possible. On the methodological side a choice could be switching (nonlinear) dynamic systems, written as

$$\begin{cases} CS(t) &= g_{\sigma(t)}\big(x_c(t), SP(t), PV(t), TS(t)\dots\big) \\ \dot{x}_c(t) &= f_{\sigma(t)}\big(x_c(t), SP(t), PV(t), TS(t)\dots\big) \\ \sigma(t^+) &= s\big(\sigma(t), x_c(t), SP(t), PV(t), TS(t)\dots\big) \end{cases} \tag{6}$$

in the continuous time (aka time-driven) context and as

$$\begin{cases} CS(t_k) &= g^*_{\sigma(t_k)}\big(x_c(t_k), SP(t_k), PV(t_k), TS(t_k)\dots\big) \\ CS(t_{k+1}) &= f^*_{\sigma(t_k)}\big(x_c(t_k), SP(t_k), PV(t_k), TS(t_k)\dots\big) \\ \sigma(t_{k+1}) &= s^*\big(\sigma(t_k), x_c(t_k), SP(t_k), PV(t_k), TS(t_k)\dots\big) \\ t_{k+1} &= t_k + q \end{cases} \tag{7}$$

in the discrete time (aka *periodic* event-driven) one.

In both (6) and (7) the σ subscript indicates that the state and the output equation can take a finite number of forms, for example to match the operating modes of a block in each of which the unique state vector x_c is managed in a different manner. In (6) t is the continuous time and the $^+$ superscript denotes values after a σ switching; in (7) q is a time quantum such that switching events can only occur at its multiples. The reader interested in more detail on these aspects can refer to (Miskowicz 2018) and the works quoted therein.

As for the technological side, a starting point could be the IEC 61499 standard (Lyu and Brennan 2020; Wies-

Listing 3. CT and DT PI models with conditional integration antiwindup.

```
model CTPI_CINT extends Base_CTPI;
  Real ui(start=0);

protected
  // numerical workaround, might be critical
  parameter Real CSeps = (CSmax-CSmin)*1e-6;

equation
  der(ui) = if CS>CSmin+CSeps and CS<CSmax-CSeps
            then K/Ti*(SP-PV) else 0;
  CS      = max(CSmin, min(CSmax,K*(SP-PV)+ui));

end CTPI_CINT;

model DTPI_CIN extends Base_DTPI;
  discrete Real csp,csi0(start=0),csi,csi1;

algorithm
  when sample(0, Ts) then
    csp  := K*(SP-PV);
    csi0 := csi1+K*Ts/Ti*(SP-PV);
    csi  := if csp+csi0>CSmin and csp+csi0<CSmax
            then csi0 else csi1;
    cs   := max(CSmin, min(CSmax,csp+csi));
    csi1 := csi;
  end when;

end DTPI_CINT;
```

Listing 4. CT and DT PI models with clamping antiwindup.

```
model CTPI_CLAMP extends Base_CTPI;
  Real xde(start=0), de, du;

protected
  parameter Real tau = Ti/1000;

equation
  xde+tau*der(xde) = SP-PV;
  de               = (SP-PV-xde)/tau;
  du               = K*de+K/Ti*(SP-PV);
  der(CS)          = if CS < CSmin and du < 0
                     or CS > CSmax and du > 0
                     then 0 else du;

end CTPI_CLAMP;

model DTPI_CLAMP extends Base_DTPI;
  discrete Real csp, csi(start=0), csi1;

algorithm
  when sample(0, Ts) then
    csp  := K*(SP-PV);
    csi  := csi1+K*Ts/Ti*(SP-PV);
    cs   := max(CSmin,min(CSmax,csp+csi));
    csi1 := cs-csp;
  end when;

end DTPI_CLAMP;
```

Listing 5. CT and DT PI models with internal feedback antiwindup.

```
model CTPI_IFB extends Base_CTPI;
  Real x(start=0);

equation
  CS = x+Ti*der(x);
  CS = max(CSmin,min(CSmax,K*(SP-PV)+x));

end CTPI_IFB;

model DTPI_IFB extends Base_DTPI;
  discrete Real x(start=0), x1;

algorithm
  when sample(0,Ts) then
    x  := (Ti*x1+Ts*cs)/(Ti+Ts);
    cs := max(CSmin,min(CSmax,K*(SP-PV)+x));
    x1 := x;
  end when;

end DTPI_IFB;
```

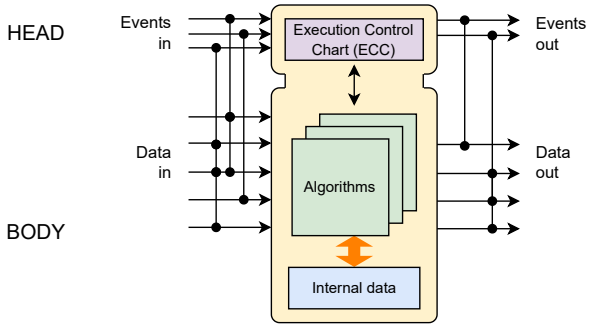


Figure 6. The concept of Function Block as per the IEC 61499 standard.

mayr, Mehlhop, and Zoitl 2023) and most notably its definition of Function Block (FB) exemplified in Figure 6. An FB is a modular, reusable component representing a functional unit in a distributed automation system. It offers data I/O to receive and produce information, and event I/O to execute and trigger actions.

Also, an FB is made of two main components, namely the Event-Condition-Action (ECA) automaton – or Execution Control Chart (ECC) – and the dynamic system. The ECA automaton governs the FB evolution based on events and conditions, triggering actions in response to specific events; the dynamic system is responsible for handling internal data, processing algorithms, and managing system state transitions over time.

The relationships between the two frameworks – methodological and technological – seem to the author quite evident and potentially capable of grounding the sought design and development process, yet they are not totally unambiguous, and in their application to the addressed domain decisions need taking. For example, continuous-time state/output equations naturally correspond to conditional DAEs, and discrete-time state/output

DOI
10.3384/ecp218603

equations to (branching) FB algorithms. Also, a sequential logic conceptually modelled as automaton naturally corresponds to an FB ECC in which one could talk about "mode σ" where σ takes values like AUTO, TRACKING, HIsat and so forth.

But coming back to our design-centric attitude, how could we *efficiently* implement this? By algorithm(s) with `when` clauses? With which event management policy? And most important, could we use the *same* automaton for time- and event-driven representations? Might we need (for the time-driven context) to represent as DAEs functionalities that were natively created as algorithms? Might we also need additional state variables for that purpose? Could this evidence numerical, solver-related issues/choices/precautions? As can be seen, we have a lot to discuss also on this front.

7 Conclusions and future work

Despite the numerous existing controller simulation models, also and particularly within the Modelica ecosystem, this paper has highlighted that substantial conceptual design choices remain to be made in the field. While many controller models – especially PID ones – are widely used, there is in fact a clear need for more flexibility to better reflect the complexities of real industrial systems, and most important, to allow an analyst to easily ensure that the simulated controls will match the particular implementation to consider. It is also important to stress that these choices are not merely technical; on the contrary, they involve a methodological shift that has yet to be fully realised.

In fact, at least to the best of the author's knowledge, no existing (Modelica) controller model has been entirely developed along the approach proposed here, and premising that the approach itself is still in its infancy and most likely contains several imperfections, nonetheless the observed *scenario* suggests that several critical design aspects are still unresolved. The challenge lies in synthesising the scattered and often fragmented knowledge available, alongside the need to adhere to industrial standards (the IEC ones being a paradigmatic but certainly not the unique example). Moreover, the development of such controller models requires careful and accurate planning, especially given the wide range of knowledge and expertise required. These conceptual and methodological gaps should be seen not just as challenges but also as opportunities for future research and innovation in the field of digital twinning for cyber-physical systems.

As an aside, though the matter could not find space in this paper, we have to notice that in addition to their relevance for industrial applications, flexible controller models also hold significant potential for educational purposes. These models could be particularly valuable in undergraduate and graduate programs, as well as in the context of continuing education, where they can bridge the gap between theoretical understanding and practical, real-world applications.

For the future, the hope is that this paper can stimulate a discussion on the problems here evidenced and (briefly) discussed, which he believes to be very relevant for the effective simulation of industrial (and particularly process) controls. Indeed, while the examples presented are intended – as said – to spark discussion, they could also mark a concrete first step toward a shared open-source initiative aimed at developing a structured, extensible library of industrial controllers, coordinating time- and event-based descriptions (here too, as exemplified above) to conjugate precision and efficiency. Yet, defining a coherent architecture and sustainable development – and particulary, maintenance – practices demands collective input, which is why the author warmly invites the community to engage in shaping both the technical direction and governance needed to realise such an initiative.

References

Åström, Karl Johan and Richard M Murray (2021). *Feedback systems: an introduction for scientists and engineers*. Princeton, NJ, USA: Princeton University Press.

Cimino, Chiara et al. (2023). "Efficient control representation in Digital Twins: An imperative challenge for declarative languages". In: *IEEE Transactions on Industrial Informatics* 19.11, pp. 11080–11090.

Cimino, Chiara et al. (2024). "Scalable and efficient digital twins for model-based design of cyber-physical systems". In: *International Journal of Computer Integrated Manufacturing* 37.10-11, pp. 1232–1251.

Grieves, Michael and John Vickers (2017). "Digital twin: mitigating unpredictable, undesirable emergent behavior in complex systems". In: *Transdisciplinary Perspectives on Complex Systems*, pp. 85–113.

Lee, Edward A (2008). "Cyber physical systems: design challenges". In: *Proc. 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing*. Orlando, FL, USA, pp. 363–369.

Leva, Alberto (2024). "Teaching PID to future professionals as a contribution to fill a historical gap". In: *IFAC-PapersOnLine* 58.7, pp. 79–84.

Lyu, Guolin and Robert William Brennan (2020). "Towards IEC 61499-based distributed intelligent automation: a literature review". In: *IEEE Transactions on Industrial Informatics* 17.4, pp. 2295–2306.

Miskowicz, Marek (2018). *Event-based control and signal processing*. Boca Raton, FL, USA: CRC press.

O'Dwyer, Aidan (2009). *Handbook of PI and PID controller tuning rules*. Singapore: World Scientific.

Skogestad, Sigurd (2023). "Advanced control using decomposition and simple elements". In: *Annual Reviews in Control* 56, 100903:1–100903:44.

Wiesmayr, Bianca, Sven Mehlhop, and Alois Zoitl (2023). "Close enough? Criteria for sufficient simulations of IEC 61499 models". In: *Proc. 19th IEEE International Conference on Automation Science and Engineering*. Auckland, New Zealand, pp. 1–7.