Towards a Functional Formalism for Modelling Complex Industrial Systems *

D. Krob † and S. Bliudze ‡
Laboratoire d'Informatique de l'Ecole Polytechnique (LIX)
CNRS & École Polytechnique

Abstract

This paper is devoted to the presentation of the main lines of an unified functional formalism for modelling complex industrial systems, that is to say systems that typically mix a big number of different software and physical devices. Our approach is based on a discrete non standard representation of time. It captures both the hierarchical architecture and the temporal and data multi-scale structures of a complex industrial system. We show in particular in this paper how our formalism allows to recover in an totally unified way various sorts of simple systems such as Turing Machines, elementary conservative physical systems and low level software/physical interfaces (sampler, modulator).

Keywords – Complex system; Hybrid system; Non standard analysis; Physical system; Software system; System modelling; Temporized system; Time scale; Turing machine.

This paper is dedicated to the memory of Imre Lakatos

1 Introduction

In the modern world, complex industrial systems are just everywhere even if they are so familiar for us that we usually forgot their underlying technological complexity. Transportation systems (such as airplanes, cars or trains), industrial equipments (such as microelectronic or telecommunication systems) and information systems are for instance typical examples of complex industrial systems that we are using or dealing with in the everyday life.

"Complex" refers here to the fact that the engineering of these industrial systems relies on incredibly complex technical and managerial processes. Such systems are indeed characterized by the intrinsic difficulty of their design, due both to an important technological heterogeneity and to the large number of sub-systems they involve. To face this huge complexity, engineers developed a number of methodological tools, popularized in the industry under the name of *system engineering* (see [22, 23] for general systems or [30, 34] for software systems), that fundamentally

* This paper was supported by the Ecole Polytechnique and Thales' chair "Engineering of complex systems".

† Corresponding author: Laboratoire d'Informatique de l'Ecole Polytechnique (LIX) – École Polytechnique – 91128 Palaiseau Cedex – France – Tel/Fax: 01 69 33 40 73/30 14 – e-mail: dk@lix.polytechnique.fr

‡ Laboratoire d'Informatique de l'Ecole Polytechnique (LIX) – École Polytechnique – 91128 Palaiseau Cedex – France – e-mail: bliudze@lix.polytechnique.fr

rely on the fact that complex industrial systems can be always recursively decomposed in a series of coupled sub-systems, up to arriving to totally elementary systems which can be completely handled. In such a framework, system engineering provides then methods for helping both the design, the architecture, the progressive integration and the final validation and qualification steps that structure the construction of an industrial complex system.

This methodological environment is however not a fully satisfactory answer to the problems that engineers must permanently solve in practice to handle this complexity. This empirical and operational engineering approach indeed hides the fact that there are basically no theoretical tools for dealing with systems at a global level. The key problem comes here in particular from the fact that the notion of an industrial system in its whole is not very well defined and rather subjective, even if it clearly corresponds to a strong industrial reality. Due to this lack of global formalism, lots of different approaches were developed – without any unified point of view – to take into account different (important, but partial) aspects of complex industrial systems. The direct practical consequence of this situation is the tremendous diversity and heterogeneity of the modelling and simuling tools that the system engineers are using in practice. These tools indeed depend first whether one is dealing with a pure software system (where UML – see [6] or [28] – is for instance still one of the core modelling technique), an embedded (real-time) system (where synchronous languages – see [4] for a comprehensive introduction and a rather complete bibliography – are the typical good design tools; see however also [19, 20] for different types of approaches to the same subject), a physical system (where Mathlab and Simulink – see [21] – are probably the key modelling and simuling standards) or an hybrid system (different kinds of existing models for this other important class of mixed systems can be found in [2, 13, 15, 25, 29, 37]). Independently of the type of system which is modelled, engineers also use different specification tools according to the part of the life cycle of a system they are dealing with: the initial informal global specification tools (such as Doors – see [35]) are for instance totally disconnected from the more formal tools that can support verification techniques (see [1, 5, 7, 31, 36] for details), but that are only used for specifying the most critical sub-systems, which are themselves altogether totally independent from the IVVQ [1] tools that can be found at the very end of the design and construction cycle of a complex industrial system.

The purpose of this paper is to try to reconnect all these (more or less disconnected) streams by going back to the very fundamentals, that is to say by looking for *an unified definition of an industrial system* from which all these different models could be deduced. Observe that such an approach is clearly in rupture with the usual one which is in fact more oriented to the local fixing of the connection problems existing between the different tools that are used for designing and managing an industrial system (by transforming them into interface design questions). We think however that the key problem is much deeper and comes directly from the fact that there does not really exist any mathematically consistent global point of view on industrial systems (even if some interesting approaches are to be noticed – see for instance [8, 33, 38]). This paper should therefore be seen as a – modest – attempt in this direction.

We indeed propose here a formal definition of a system which intends to capture both continuous and discrete systems which are the two major types of technical sub-systems that compose a given industrial system [2] (see Section 3). The key point on which our approach relies is a common (discrete) modelling of time based on the use of a non standard model of real numbers (see Section 2). Making this (very strong) change allows indeed to take into account in the same way both conservative physical systems and computer systems (see again Section 3 for several

[1] Integration, Verification, Validation, Qualification.

[2] We will in particular not try to integrate human systems – such as company organizations – in our modelling even if such systems are also fundamental for some type of applications (typically information systems).

examples). Moreover our systemic models are always causal (see Section 3.1): non causality appears indeed in our approach as the consequence either of abstraction (i.e. simplifying our model) or standardization (i.e. going back to the usual model of time). Observe also finally that large classes of classical systems such as synchronous, Hamiltonian or dynamical systems can also be recovered within our approach (see Section 4 for some insights on these questions).

2 Time

In order to model all industrial systems in an unified way, the first key problem – as already mentioned above – is to develop a common functional [3] framework that takes both into account continuous and discrete systems (typically physical and computer systems), that is to say systems whose time evolution behavior is represented either continuously or discretely. At this point, two directions can be chosen to construct an unified theory, depending whether one prefers to develop a continuous or a discrete point of view on time. In this paper, we decided to develop the discrete approach since we wanted to keep the usual intuitions on computer systems. The price to pay is then the change of model of real numbers on which time rely in order to capture also continuous systems in a same global framework. Note however that one could also do the converse and deal always with the usual continuous modelling of time which will then lead us to develop a distribution point of view (see [32]) on computer systems, using for instance typically Dirac combs for modelling the discrete entries of a given software system.

2.1 Non standard analysis

To develop a global (discrete) unified framework for dealing both with continuous and discrete industrial systems, we will therefore go back to the 18th century representation of real numbers (see [10, 17, 18]), that is to say to the so-called non-standard model of $\mathbb{R}$. In other words, we will consider in all the sequel that real numbers are given by the set ${}^*\mathbb{R}$ of *non-standard reals* [4] as initially formalized by Robinson [27] (for a comprehensive introduction to non standard reals see the paper of Lindström in [9]). This set (see Figure 1) is a real-closed field that contains all usual real numbers, but also the infinitesimal reals (i.e. the non zero non standard real numbers that have their absolute value strictly less than any $r \in \mathbb{R}_*^+$) – whose set will be denoted by $\mathbb{I}$ in the sequel – and their inverses which are the infinitely great reals (i.e. whose with an absolute value strictly greater than any usual real number $r \in \mathbb{R}$).

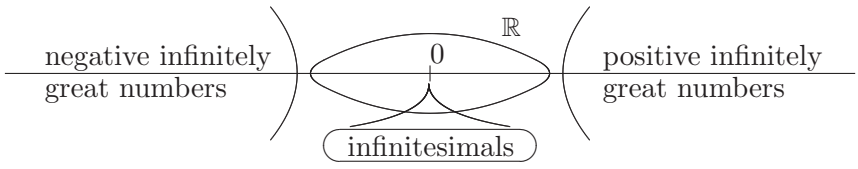


Figure 1: Graphical representation of non-standard real numbers.

Note in particular that one can prove that the field ${}^*\mathbb{R}$ is elementarily equivalent to $\mathbb{R}$, which means that the first order logical properties of $\mathbb{R}$ and ${}^*\mathbb{R}$ (expressed in the logical theory of

[3] "Functional" refers to the fact that systems will only be considered here as input/output functions. We will in particular not try to model the architecture on which relies a system (which would bring us too far ...).

[4] A star on the left of a symbol as in ${}^*\mathbb{R}$ stands always for "non-standard", whereas on the right, it means either that zero is not included, as in $\mathbb{R}^*$ or ${}^*\mathbb{R}^*$ (which denote respectively the usual and the non standard sets of non zero real numbers) or that we speak of the set of words over a given alphabet, as in A^*.

ordered fields) are exactly the same (see [3] and [14] for more model theoretical fundamentals of non standard analysis). Observe also that, among all non-standard real numbers, one can of course consider the set ${}^*\mathbb{Z}$ of *non-standard integers* that, on top of standard integers, contains infinitely great ones, having absolute value greater than any $n \in \mathbb{N}$.

Note finally that two non-standard real numbers x and y are said to be *infinitely close* (which is denoted by $x \approx y$) if and only if $x - y$ is infinitesimal. This last notion allows, for instance, to recover the usual concept of continuity. Indeed, the standard image $f : \mathbb{R} \to \mathbb{R}$ of a non standard function ${}^*f : {}^*\mathbb{R} \to {}^*\mathbb{R}$ is continuous in a standard point $x_0 \in \mathbb{R}$ if and only if one has

$$\forall\ \delta \in {}^*\mathbb{R}, \quad \delta \approx 0 \ \Rightarrow\ {}^*f(x_0 + \delta) \approx {}^*f(x_0)\,.$$

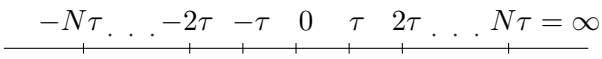
More details about non standard analysis can be found in [11] and [9].

2.2 Time scales

From now on, we will suppose that the time is modelled by ${}^*\mathbb{R}$ throughout all this paper. Let us therefore give the following first fundamental definition.

Definition 2.1 *Let $\tau \in {}^*\mathbb{R}_*^+$ be a strictly positive non-standard real number. The set $\mathbb{T}_\tau = {}^*\mathbb{Z}\,\tau$ will then be called the* time scale *of step $\tau > 0$. Any element $t \in \mathbb{T}_\tau$ will said to be a* moment *on this last time scale.*

A time scale can therefore be seen as a discrete series of clock ticks occuring at times $n \cdot \tau$, with $n \in {}^*\mathbb{Z}$ being a non-standard integer eventually infinitely great.



The following lemma shows then we can recover usual continuous time within this model by considering time scales with infinitesimal steps (i.e. with $\tau \approx 0$).

Lemma 2.2 *Let $\mathbb{T}_\tau$ be a time scale with an infinitesimal step τ and let $r \in \mathbb{R}$ be any standard real number. Then there always exists a moment $t \in \mathbb{T}_\tau$ which is infinitely close to r.*

Proof — Let $\tau \approx 0$ be an infinitesimal non standard real number and let r be a usual standard real number. The non standard version of Archimedes' axiom (see Chapter A.6 of [3] or use the transfer principle as explained in [11]) shows that there exists a non standard integer $N \in {}^*\mathbb{N}$ (here necessarily infinitely great) such that $N < r/\tau \leq N+1$. Hence we have $N\tau < r \leq (N+1)\tau$ and consequently $0 < r - N\tau \leq \tau \approx 0$. Thus putting $t = N\tau$, we have $t \approx r$. ∎

More generally, we can classify possible all time scales $\mathbb{T}_\tau$ into the following three groups according to the nature of their step τ:

- *continuous* time scales when their time step is infinitesimal, i.e. when one has $\tau \approx 0$,
- *discrete* time scales when their time step is a non infinitesimal bounded non standard real number, i.e. when one has $\tau \approx r$ for some strictly positive usual real number $r \in \mathbb{R}_*^+$,
- *infinite* time scales when their time step is an infinitely great non standard real number.

Note that the latter case is not of practical interest as there are essentially only three "standard" moments on an infinite time scale, that is to say $-\infty$, 0, and $+\infty$. Therefore we will concentrate in the sequel of this paper uniquely on the time scales of the first two types.

Let us finally give the following three definitions that we will use for dealing with systems.

Definition 2.3 *A time scale $\mathbb{T}_\tau$ is said to* refine *another time scale $\mathbb{T}_{\tau'}$ – which is denoted by $\mathbb{T}_{\tau'} \preceq \mathbb{T}_\tau$ – if and only if one of the two following equivalent properties holds:*

- $\mathbb{T}_{\tau'} \subset \mathbb{T}_\tau$
- $\exists\ N \in {}^*\mathbb{N},\ \tau' = N\tau$

Definition 2.4 *Given two time scales $\mathbb{T}_\tau$ and $\mathbb{T}_{\tau'}$, we shall call* synchronization points *all the moments that belong both to the two time scales, i.e. to $\mathbb{T}_\tau \cap \mathbb{T}_{\tau'}$.*

Observe that, if $\mathbb{T}_\tau$ refines $\mathbb{T}_{\tau'}$, any moment on $\mathbb{T}_{\tau'}$ is a synchronization point.

Definition 2.5 *A* temporal filter *is a set of time scales $\mathcal{F} = \{\mathbb{T}_\tau\}_{\tau \in T}$ such that $\preceq$ is a total order on $\mathcal{F}$. In other words, $\mathbb{T}_\tau$ and $\mathbb{T}_{\tau'}$ are always comparable by $\preceq$ for any $\tau, \tau' \in T$.*

3 Systems

3.1 Definition

In this section, we give a formal definition of the notion of system which tries to capture the realness of industrial complex systems. Most systems – be that industrial scale technological systems or networks of information processing machines (and probably also certain biological systems) – are too complex to be modelled or analyzed as a whole, but can be treated as the result of the integration of several components. These components tend to be simpler systems that can in their turn be decomposed in the same way. We arrive eventually at the level where all such components are *elementary systems*, i.e. sufficiently simple to be considered independently of their structure. The following key definition is an attempt to capture this as yet intuitive and informal process (on which however relies system engineering in the industry).

Definition 3.1 *A* system *S is recursively defined as the union of the following elements:*

- *an input/output mechanism that consists respectively in:*
  - *an* input channel *x which is capable of receiving – only at moments that belong to a given time scale $\mathbb{T}_{\tau_i}$ called the* input time scale *– data that belong to a given set I, called the* input domain *of S (and also denoted by $In(S)$),*
  - *an* output channel *y which is capable of emitting – only at moments that belong to a given time scale $\mathbb{T}_{\tau_o}$ called the* output time scale *– data that belong to a given set O, called the* output domain *of S (and also denoted by $Out(S)$),*
- *two internal storage mechanisms that consist respectively in:*
  - *an* internal memory *given by a tape indexed by ${}^*\mathbb{N}$ – with a window that can take any value within ${}^*\mathbb{N}$ – which may contain any (non standard) finite number* [5] *of values in a given set M (also denoted by $Mem(S)$), called the* memory domain,

[5] Recall that a set is said to be a (non standard) "finite" set if and only if it can be put in bijection with a set of the type $[0, N]$ where N stands for any (either usual or infinitely great) non standard positive integer in ${}^*\mathbb{N}$.

  - *an* internal state set *which is just an arbitrary finite (in the usual standard meaning) set Q (that is also denoted by $State(S)$),*

- *an internal time behavior which is given by*
  - *an* internal time scale $\mathbb{T}_{\tau_s}$ *that refines both the input and the output time scales, i.e. which satisfies to $\mathbb{T}_{\tau_s} \preceq \mathbb{T}_{\tau_i}$ and $\mathbb{T}_{\tau_s} \preceq \mathbb{T}_{\tau_o}$,*
  - *an* internal state evolution function $q(t)$ *which maps each element $t \in \mathbb{T}_{\tau_s}$ onto some element $q(t) \in Q$ (called the value of the internal state at moment t),*
  - *an* internal memory evolution function $m(t)$ *which maps each element $t \in \mathbb{T}_{\tau_s}$ onto some element $m(t) \in M^f$ (called the value of the internal memory at moment t)* [6] *,*

- *three transition mechanisms that consist respectively in:*
  - *a function* read $: Q \times I \to Q \times M^f$ *that can read a given value on the input channel and write correspondingly a series of values – depending on the status of a state $q \in Q$ (updated after the operation) – onto the internal memory,*
  - *a* controller function $\delta : Q \times M \times {}^*\mathbb{N} \to Q \times M \times {}^*\mathbb{N}$ *that allows – as we will see – to replace an element of the internal memory by another one* [7] *,*
  - *a function* write $: Q \times M^f \to Q \times O$ *that can read a set of values on the internal memory and write correspondingly a value – that may depend on the status of a state $q \in Q$ (updated after the operation) – onto the output channel,*

- *a finite (in the usual standard meaning) set of systems $Sub(S) = \{S_1, \ldots, S_n\}$, which are called the* sub-systems *of S that are equipped with:*
  - *for each $k = 1, \ldots, n$, a function $\rho_k : Q \times Out(S_k) \to Q \times M^f$ that reads the output of S_k, writes it into the internal memory and changes possibly of internal state,*
  - *for each $k = 1, \ldots, n$, a function $G_k : Q \times M^f \times \bigotimes_{i=1}^{n} Out(S_i) \to Q \times In(S_k)$ that defines the interactions between the sub-systems,*

  and such that moreover the output and input time scales of all these sub-systems are always refined by the internal time scale of S [8] *.*

The previous definition is however purely statical and does in particular not give any insight on the time behavior of a given system. To progress in this last direction, we first introduce the notion of instantaneous description of a system.

Definition 3.2 *Let S be a system. An* instantaneous description *of S is then any quadruple of the type $d = (t, q, m, i) \in \mathbb{T}_{\tau_s} \times Q \times M^f \times {}^*\mathbb{N}$ such that one has*

- *$t \in \mathbb{T}_{\tau_s}$ is a moment of the internal time scale $\mathbb{T}_{\tau_s}$ of S,*
- *$q \in Q$ is the value $q(t)$ of the internal state evolution function of S at moment t,*
- *$m \in M^f$ is the value $m(t)$ of the internal memory evolution function of S at moment t,*

[6] We denote here by M^f the set consisting of all non standard finite (in the non standard meaning) sequences over a set M (i.e. partial functions ${}^*\mathbb{N} \to M$ with non standard finite support).

[7] Depending on the value of a given state of Q which can be also changed by the action of δ.

[8] Forming therefore altogether a temporal filter.

- $i \in {}^*\mathbb{N}$ *is the position of the window of the internal memory of* S *at moment* t [9] .

We are now in position to define the time behavior of a system which just appears as a series of instantaneous descriptions (completed by the time evolutions of the values of the input and output channels that we did not integrated in these descriptions) indexed by its internal time scale and submitted to some natural transition constraints.

Definition 3.3 *Let* S *be a given system. A* time behavior *associated with* S *is then any family* $d(t) = (t, m(t), q(t), i(t))$ *of instantaneous descriptions of* S *where* t *describes the internal time scale* $\mathbb{T}_{\tau_s}$ *of* S *and where one always pass from* $d(t)$ *to* $d(t+\tau_s)$ *by executing one of the following possible transition actions:*

1. *for any* $k = 1, \ldots, n$ *and at every synchronization point* $t + \tau_s$ *between all concerned time scales, update the input of* S_k *and the current internal state of* S *by setting*

$$(q(t+\tau_s), x_k(t+\tau_s)) = G_k\Big(q(t); m(t); y_1(t), \ldots, y_n(t)\Big)\,, \tag{1}$$

   where x_k *denotes the input channel of the sub-system* S_k *and where each* y_i *stands for the output channel of* S_i *correspondingly,*

2. *for any* $k = 1, \ldots, n$ *and at each synchronization point* t *between the internal time scale* $\mathbb{T}_{\tau_s}$ *of* S *and the output time scale of* S_k*, read the output of* S_k *and update both the internal memory of* S *– beginning at the current position of its associated window – and the current value of its internal state by setting*

$$(q(t+\tau_s), m(t+\tau_s)_{i\geq i(t)}) = \rho_k\Big(q(t); y_k(t)\Big)\,, \tag{2}$$

3. *at each synchronization point* t *with the input time scale, perform a* read *operation – depending on the value of the internal state of the system – and update both the internal memory of* S *beginning at the current position of its window and this internal state:*

$$(q(t+\tau_s), m(t+\tau_s)_{i\geq i(t)}) = \mathrm{read}\Big(q(t); x(t)\Big)\,, \tag{3}$$

4. *at each synchronization point* t *with the output time scale, perform a* write *operation by taking into account both the internal state of the system (that is updated after the operation) and the values of the internal memory that begin at the current position of its window:*

$$(q(t), y(t)) = \mathrm{write}\Big(\, q(t); m(t-\tau_s)_{i\geq i(t-\tau_s)}\Big)\,, \tag{4}$$

5. *at any other moment* t *on* $\mathbb{T}_{\tau_s}$*, update the internal state, the value of the current position of the internal memory of* S *and the current position of its associated window:*

$$(q(t+\tau_s), m(t+\tau_s)_{i(t)}, i(t+\tau_s)) = \delta\Big(q(t), m(t)_{i(t)}, i(t)\Big)\,. \tag{5}$$

Time behaviors may however not be unique. This remark leads us to the following definition (observe that we will mainly consider deterministic systems in the examples of this paper).

Definition 3.4 *A system* S *is said to be* deterministic *if and only* S *possesses exactly one single time behavior. If it is not the case, the system is said to be* non-deterministic.

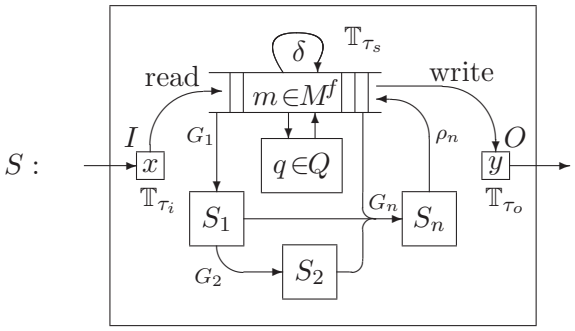


Figure 2: Graphical representation of a system S.

The previous definitions are illustrated by the diagram in Figure 2. For the sake of simplicity and clarity, we will occasionally vary this representation. In particular, we will sometimes omit domains of variables or their names, if the omitted parts are clear from the context.

Note 3.5 One can also consider systems with more than one tape, that is to say with several internal memory variables, which corresponds to saying that M is a direct product of different independent sets. In this case, there has to be a window as above per tape.

Note 3.6 The sub-system graph of S is the graph which is formed by taking $Sub(S)$ as vertices and with edges defined by $(\mathcal{G}_i)_{i=1,\dots,n}$, i.e. such that there is an edge going from S_i to S_j if and only if x_j depends on y_i according to Equation (1). This last graph can contain cycles and does not necessarily have to be connected. Cycles in the sub-system graph represent feedback loops common to the sub-systems that are involved in such cycles.

Note 3.7 Note that we suppose that each time scale $\mathbb{T}$ of an input or an output channel of a sub-system of a given system S is always refined by the internal time scale $\mathbb{T}_{\tau_s}$ of this system, i.e. that one has $\mathbb{T}_{\tau_s} \prec \mathbb{T}$, even if such a time scale is *free for interaction* which means that the corresponding channel is never implied in one of the relations (1).

In order to classify the level of complexity of a system, we now introduce the notion of *order* of a system. A system shall said to be of *order* N if it is constructed using only sub-systems of order $N-1$ and less. Systems of *zero-th order* are called *elementary* systems.

Definition 3.8 *We define the* order *of a system S by setting*

$$ord(S) = \begin{cases} 0 & \text{if } Sub(S) = \emptyset\,, \\ 1 + \max\{ord(S') \,|\, S' \in Sub(S)\} & \text{otherwise}\,. \end{cases}$$

We are now in position to introduce the notion of well defined system (which will in fact be the only kind of systems that we shall consider in the sequel).

Definition 3.9 *A system S is said to be* well defined *if there exists a positive standard integer $N \in \mathbb{N}$ such that $N = ord(S)$.*

[9] Whose value is given by $m(t)$.

Note that each time behavior of a system implicitly defines an input/output relation of the following type (where we took here all the notations of the previous definitions):

$$y(t_0+\tau_o) = \mathcal{F}\Big(x(u), m(t), q(t) \,\Big|\, u,t \in [t_0, t_0+\tau_o[\Big)\,, \tag{6}$$

where t_0 describes the output time scale $\mathbb{T}_{\tau_0}$ of the system and where $u,t \in [t_0, t_0+\tau_o[$ signifies that each of these variables is taken between t_0 and $t_0+\tau_o$ (without reaching this last upper limit) respectively on the time scale $\mathbb{T}_{\tau_i}$ and $\mathbb{T}_{\tau_s}$. For more simplicity, we will rewrite equivalently the input/output relation (6) in the following simplified functional form

$$y = \mathcal{F}(x; q, m)\,. \tag{7}$$

It is easy to see that the function $\mathcal{F}$ is uniquely associated with a given system S if and only if S is a deterministic system. Observe also that $\mathcal{F}$ must obviously (by construction) always be a causal function, i.e. each value $y(t)$ depends only on the values $x(t')$ with $t' < t$.

Note 3.10 In several classical models of systems – such as dynamical systems (see [12]) or synchronous systems (see [4]) – non causality is however a real problem that can eventually occur. Such a situation is indeed always the direct consequence of the collapse between system feedbacks and the hypothesis – which is implicit in usual continuous modelling (and corresponds to standardization with respect to our approach), but totally explicit in synchronous modelling – that no time is required to realize the internal treatments of a given system.

Two main and complementary approaches to system design, namely specification and engineering, can be distinguished. While the latter is interested in full details in the effective way a system can be constructed, the former only deals with formal requirements on the input/output relation of a system. In other words, the system specification approach considers a system as a "black box" with a precise functional behavior, but without trying to know how such a behavior is obtained. This approach is fundamental in the industry: a computer manufacturer will for instance be able to give the specification of a given wireless interface to different suppliers that may realize different wireless computer interfaces from their structural point of view, as soon as the input/output relations specified by the manufacturer are exactly the same. Such situations are modelled by the following definition for equivalent systems.

Definition 3.11 *Two systems S_1 and S_2 are said to be* equivalent *if and only if one both has $In(S_1) = In(S_2)$, $Out(S_1) = Out(S_2)$ and the following conditions, i.e.*

$$\forall\ t \in \mathbb{T}_{\tau_i},\ x_1(t) = x_2(t)\ \Rightarrow\ \forall\ t \in \mathbb{T}_{\tau_o},\ y_1(t) = y_2(t)\,,$$

where (x_1, y_1) and (x_2, y_2) stand respectively for the input and output channels of S_1 and S_2.

Note 3.12 The previous definition has mainly a meaning when S_1 and S_2 are deterministic.

3.2 Elementary systems

We will now study more in details some important classes of elementary systems, i.e. of systems of order 0. We will in particular show that our framework allows already to capture at this level different classical interesting classes of systems of very different nature. Note finally that we will concentrate in this subsection on elementary systems of the following three types (using here a general terminology which is not specially reserved to elementary systems):

1. *software systems* model phenomena observed mostly in information technologies: they are characterized by the fact that their three defining time scales are all discrete,
2. *physical systems* are characterized by the fact that their three defining time scales are all continuous: they are generally used to model real-life physical systems,
3. *hybrid systems* mix finally – by definition – both discrete and continuous time scales.

3.2.1 Elementary software systems

Elementary software systems have only discrete time scales. Their input and output spaces will be called alphabets (which refers to the notion of set of letters or symbols). We assume that an elementary software system is equipped with a tape and a corresponding window that indicates the position of its head (this definition can be generalized to take into account several tapes).

As depicted on Figure 3, elementary software systems are receiving – on an input channel – data within some input alphabet I at a rate given by the input time scale $\mathbb{T}_{\tau_i}$. They are also emitting – on an output channel – data that belong to some output alphabet O at a rate given by the output time scale $\mathbb{T}_{\tau_o}$. Moreover any elementary software system has also the right to perform – at a rate given by its internal time scale $\mathbb{T}_{\tau_s}$ – a number of internal actions controlled by the value of an internal state $q \in Q$, that is to say:

- read an input data x, transform it into a word (depending only on x) on the tape alphabet and write it finally on the tape beginning at the current position of its window,
- change the value of the element of the tape that is obtained by looking on the current position of its window (that can be updated after the operation),
- write an output data y by taking a word w on the tape beginning at the current position of its window and transforming it (depending only on w) into y.

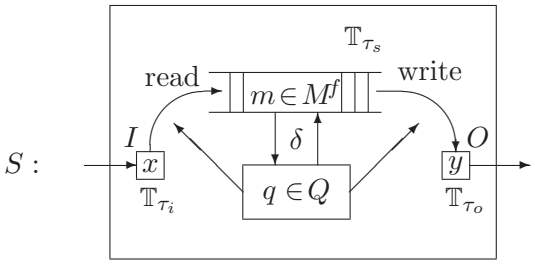


Figure 3: Graphical representation of an elementary software system S.

As we can see, elementary software systems are therefore just a slight generalization of usual Turing machines which is obtained by adding to this classical model a permanent input/output temporal behavior. The reader can indeed easily check that Turing machines (or equivalently recursive functions if one prefers to stay within a functional approach) correspond to the degenerated case of our model where one considers elementary software systems that can only perform a unique read action (or whose input channel will only receive a single input data during all possible moments of time). Note also that as an immediate consequence of this simple observation, we got the undecidability of the existence of a system's output !

Example 3.13 (One element buffer) Let us now present an example of elementary deterministic software system that we will use in the sequel (as a sub-system of an higher order system). Our example consists in a buffer capable of storing only – at each moment of time – one single message out of a message set A. We assume that this buffer has two input channels. On the first input channel, the buffer can only receive either a message $m \in A$ or a distinguished empty message ε. On the second one, it can receive either a write request, that we will denote by '$\uparrow$', or again the distinguished empty message ε. The buffer stores each message it receives on the first channel in a fixed memory cell which is overwritten each time a new non-empty message arrives on the same channel. When the buffer receives a write request, it sends the currently stored message on its output channel. A representation of such a buffer is shown in Figure 4.

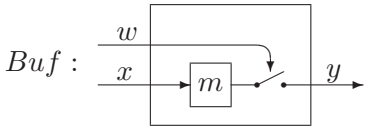


Figure 4: Graphical representation of a one-element buffer.

Let us now describe how to model this simple buffering mechanism by an elementary deterministic software system, denoted by Buf. The input, internal and output time scales of such a system are all discrete with respective time steps $\tau_i = \tau$ and $\tau_s = \tau_o = \tau/2$. In other words, we require the buffer to operate internally on a rate which is twice as fast as its input rate. The input domain of Buf is clearly modelled by $(A \cup \{\varepsilon\}) \times \{\varepsilon, \uparrow\}$ in order to take into account the two entry channels, when the output domain is just equal to $A \cup \{\varepsilon\}$. The memory domain will be equal to A. Finally the internal state set of Buf is defined as $\{r, \varepsilon, \uparrow\}$ (the first state models the reading of the input channel, when the two last ones correspond to the two possible writing decisions on the output channel). Therefore we have:

$$In(Buf) = (A \cup \{\varepsilon\}) \times \{\varepsilon, \uparrow\}, Mem(Buf) = A, Out(Buf) = A \cup \{\varepsilon\}, State(Buf) = \{r, \varepsilon, \uparrow\},$$

The control mechanisms of Buf are now given by the following transition functions:

$$\delta = -, \quad \texttt{read}(r; (x, w)) = \begin{cases} (w; x) & \text{if } x \neq \varepsilon, \\ (w; -) & \text{otherwise}, \end{cases} \qquad \texttt{write}(q; y) = \begin{cases} (r; y) & \text{if } q = \uparrow, \\ (r; \varepsilon) & \text{if } q = \varepsilon, \end{cases} \tag{8}$$

where $-$ means not defined or no action (depending on the situation). Observe that the choice of δ just reflects the fact that the input message is stored in a single cell of the internal memory on which no action can be made. The unique possible time behavior of the buffer consists then just in alternating a `read` and a `write` action at each moment of its internal time scale. ■

3.2.2 Elementary physical systems

Before presenting the notion of elementary physical system, let us first introduce the general framework on which rely this concept. We will indeed suppose that each instance p of a given physical parameter φ (such as mass, distance, kinetic energy, potential energy, torsion energy, temperature, kinetic momentum, etc.) with whom we will deal, can always be both

1. *measured* using a measure function m_φ, which means that one can associate to each such physical quantity p of type φ its measure $m_\varphi(p) \in {}^*\mathbb{R}$,
2. *decomposed into a (non standard) finite sum of infinitesimal quantities*, i.e. a sum of physical quantities p of type φ that have an infinitely small measure $m_\varphi(p) \approx 0$.

The set of all physical quantities of a given type φ is then called the physical domain associated with φ and denoted by $\mathbb{P}_\varphi$. In the same way, the physical infinitesimal domain of type φ – which is denoted by $\mathbb{I}_\varphi$ – consists of all infinitesimal physical quantities of type φ.

Elementary physical systems can now be described exactly in the same way than elementary software systems, i.e. by a mechanism similar to the one given by Figure 3, the only (but fundamental) difference being here that such systems manipulate physical quantities using continuous time scales. An elementary physical system is indeed characterized by the fact that it has

1. continuous input, internal and output time scales,
2. input and output domains that are both equal to the same finite (in the usual meaning) product of physical domains, i.e. both equal to $\otimes_{i=1}^{n} \mathbb{P}_i$ for some finite (standard) positive integer $n \in \mathbb{N}$, where each $\mathbb{P}_i$ stands for a physical domain of a given type,
3. an internal domain which is necessarily equal to $\otimes_{i=1}^{n} \mathbb{I}_i$ where each $\mathbb{I}_i$ stands for the infinitesimal physical domain associated with the physical domain $\mathbb{P}_i$ which is involved both in the corresponding input and the output domain.

For the sake of simplicity, we can of course consider – without any extension of the representation power of our model – that an elementary physical system has a finite (in the usual sense) number of tapes, each of them devoted to some particular infinitesimal physical domain.

Such elementary systems are intended to model real physical systems, considered as transformers of infinitesimal physical quantities. The behavior of an elementary physical system S – in our framework – has indeed to be physically interpreted as follows:

- S receives at each moment of its input time scale (hence infinitely often) a vector x that consists of different physical quantities of given types, i.e. a vector $x \in \otimes_{i=1}^{n} \mathbb{P}_i$ where each $\mathbb{P}_i$ stands for some physical domain; it transforms then each component $x_i \in \mathbb{P}_i$ of x into a (non standard) finite sequence $(x_i^j)_{j=1\ldots N}$ – written on a specific tape of the internal memory of S – of infinitesimal physical quantities within $\mathbb{I}_i$ (i.e. of the same type than $\mathbb{P}_i$) whose sum has the same measure than x_i, i.e. such that

$$\sum_{j=1}^{N} m_i(x_i^j) = m_i(x_i)\,, \tag{9}$$

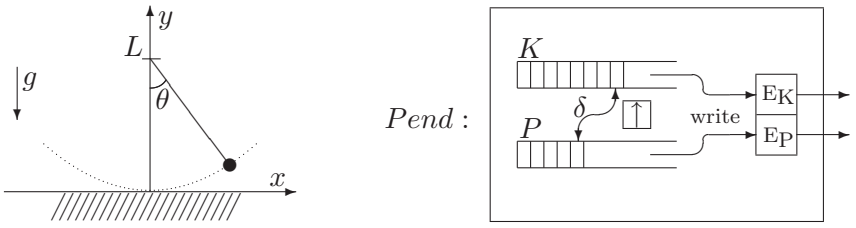
  where m_i stands for the measure function associated with the physical domain $\mathbb{P}_i$,

- S can transform infinitely often, at the rate given by its internal time scale, any infinitesimal physical quantity of a given type written on one of its tapes into another infinitesimal physical quantity of another given type (that can also be stored on another tape),
- S emits at each moment of its output time scale (hence again infinitely often) a vector $y \in \otimes_{i=1}^{n} \mathbb{P}_i$ whose components are obtained by "gluing" altogether sequences of infinitesimal physical quantities (of compatible types) coming from the internal memory of S, by using the reverse process of the initial writing mechanism as described above.

Observe that we can only model in such a way *non dissipative* physical systems since the internal controller of an elementary physical system – modelled by the function δ – is only able to change an elementary physical quantity of known type into another elementary physical quantity of known type ("elementary" being modelled by "infinitesimal" in our approach). Conversely one can also prove that large classical classes of conservative physical systems – such as Hamiltonian

systems (cf. [24]) – can be recovered (as higher order systems) inside our model. We will however not prove here this last result which is quite technical, but rather illustrate on a simple example how to analyze a classical mechanical system as an elementary deterministic physical system.

Example 3.14 (Simple pendulum) Let us now consider a simple pendulum as shown in Figure 5-a. The pendulum consists of a point mass m attached to the point $(0, L)$ by a rigid string of negligible mass and of length L (when hanging freely the pendulum touches the ground).



(a) (b)

Figure 5: Simple pendulum: mechanical (a) and systemic (b) representations

Its motion can be classically described by applying the fundamental principle of dynamics which leads immediately to the following differential equation

$$m\,L\,\dot{\varphi} = -m\,g\,\sin\theta\ , \tag{10}$$

where θ and $\varphi = \dot{\theta}$ stand respectively for the angle formed by the string and the y axis and for the corresponding angular speed. In its turn, Equation (10) is clearly equivalent to the following energy preservation equation (obtained by integrating this last equation)

$$\frac{1}{2}\,m(L\varphi)^2 + mgL\cos\theta = C\,, \tag{11}$$

where the first and the second summand in the left hand side represents respectively the kinetic and the potential energy of the pendulum and where C stands for the initial potential energy of the pendulum, when it is in the point farthest from the y axis with zero angular speed.

An elementary deterministic physical system $Pend$ modelling such a pendulum is shown in Figure 5$-b$. This system takes no input and provides on the output channel a pair of physical quantities that consists respectively in the pendulum's current kinetic and potential energy. Its internal and output time scales are supposed to be the same continuous time scale (with dt as common infinitesimal time step). We define finally the output domain, the memory domain and the internal state set of the system to be respectively equal to

$$Out(Pend) = \mathbb{E}_K \times \mathbb{E}_P\ ,\ Mem(Pend) = \{0, de_K\} \times \{0, de_P\}\ ,\ State(Pend) = \{x, s\} \times \{\uparrow, \downarrow\}\,,$$

where $\mathbb{E}_K$ and $\mathbb{E}_P$ denote respectively the two physical domains which are associated with kinetic and potential energy, where de_K and de_P stand respectively for two infinitesimal quanta of kinetic and potential energy – with a common measure $m_K(de_K) = m_P(de_P) = de \in \mathbb{I}$ – and where the x, s and the arrow states should respectively be interpreted as the two possible internal actions of the pendulum (exchanging energy on its two tapes – see below – or sending physical quantities to the output channel) and as the two possible directions of the pendulum's motion. This system has therefore two tapes K and P, each of them containing a (non standard)

finite number of copies of the corresponding infinitesimal quantum of energy (the other parts of the two tapes being equal to 0). Moreover the memory of our system evolves in such a way that it always contains the same global number (necessarily infinitely great) $N \in {}^*\mathbb{N}$ of energy quanta (which satisfy to the energy conservation condition $N\,de = C$). The system's behavior consists then essentially in taking at each moment of time one quantum of energy from one tape, depending on the direction the pendulum is moving, and putting it on the other one, until the working tape is empty. The corresponding controller function δ is given below.

$$\delta((x,\uparrow),(k,p),(i_K,i_P)) = \begin{cases} \Big((s,\uparrow),(0,de_P),(i_K-1,i_P+1)\Big) & \text{if } (k,p)=(de_K,0) \text{ and } i_K>1, \\ \Big((s,\downarrow),(0,de_P),(0,i_P+1)\Big) & \text{if } (k,p)=(de_K,0) \text{ and } i_K=1, \end{cases}$$

$$\delta((x,\downarrow),(k,p),(i_K,i_P)) = \begin{cases} \Big((s,\downarrow),(de_K,0),(i_K+1,i_P-1)\Big) & \text{if } (k,p)=(0,de_P) \text{ and } i_P>1, \\ \Big((s,\uparrow),(de_K,0),(i_K+1,0)\Big) & \text{if } (k,p)=(0,de_P) \text{ and } i_P=1, \end{cases}$$

where i_K and i_P are the cursors of tapes K and P. The `write` function is then defined as the constructer of the two global physical quantities (i.e. kinetic and potential energy) that can be obtained by summing all infinitesimal quanta that respectively exist on tapes K and P [10]. This last function works only in state $(s,\updownarrow)$ and ends in state $(x,\updownarrow)$ (where $\updownarrow$ stands for any type of arrow). The unique possible time behavior of our (deterministic) system consists hence just in alternating permanently a tape exchange operation with a write operation.

It is then immediate to see that the elementary physical system that we just defined, always satisfies – by construction – to the energy conservation equation

$$E_K + E_P = C \ , \tag{12}$$

where E_K and E_C stand for the measures of the kinetic and the potential energy of the pendulum, that is exactly equivalent to Equation (11). Note however that our approach does not connect the physical quantities that we manipulated (here kinetic and potential energy) with the high level parameters φ and θ that were used in writing down Equation (11).

To fill this gap, we must interpret the pendulum as a new deterministic system *Newpend* of higher order that contains the previous elementary physical system *Pend* as a sub-system. This new system has another sub-system which makes alternatively the two only operations:

- it reads the outputs of *Pend* and transforms them – by applying the two associated measure functions – into (non standard) real values k and p that are stored in its internal memory,
- it takes the two last non standard real values k and p and writes them on its output channel by applying the following transformation

$$\texttt{write}\,(k,p) = \left(\frac{1}{L}\sqrt{\frac{2\,k}{m}}\,,\, \arccos\frac{p}{mgL} \right) ,$$

  which can be expressed within our model by making use of adapted sub-systems, due to the fact that we are only dealing here with analytic transformations (see Section 4).

If one identifies the output channels of this last sub-system to the output channels of *Newpend*, it is then obvious to see that *Newpend* produces on its own output channels the pair (φ,θ) in such a way that the energy conservation Equation (11) is always fulfilled. ■

[10] We can easily assume that the cardinality of these two families of infinitesimal quanta is respectively given by i_K and i_P. Under this hypothesis, the `write` function can be more precisely defined – independently of the value of the current internal state of the system – by setting $\texttt{write}(K,P) = (i_K\,de_K, i_P\,de_P)$.

3.2.3 Elementary hybrid systems

Elementary hybrid systems are systems of order 0 which can transform continuous behaviors into discrete ones (or vice-versa). They can therefore naturally be used for modelling interfaces between software and physical systems. The two following examples illustrate how to interpret within our approach two classical interfaces of this kind – here a sampler and a modulator – that are totally fundamental in practice.

Example 3.15 (Sampler) A sampler is a mechanism that takes a continuous time input function and produces a discrete sequence of samples of its values. It can be modelled by an elementary deterministic hybrid system H_τ which is parameterized by the time step $\tau > 0$ of its discrete internal time scale. The output time scale of this system is also discrete with a double time step $2\,\tau$. On the other hand, the input time scale of H_τ is continuous with an infinitesimal time step $dt = \tau/N$ (where $N \in {}^*\mathbb{N}$ is a given infinitely great non standard integer). The input, output and memory domains of H_τ are all equal to ${}^*\mathbb{R}$, when its internal state set has just two states r and w that should be interpreted as the two possible internal actions of the sampler (reading on the input channel, writing on the output channel), that is to say:

$$In(Sampler) = Out(Sampler) = Mem(Sampler) = {}^*\mathbb{R}\,,\ State(Sampler) = \{r, w\}\ .$$

The control mechanisms of H_τ are now given by the following transition functions:

$$\texttt{read}\,(r; x) = (w; x)\,, \quad \texttt{write}\,(w; y) = (r; y)\,, \quad \delta = -\,, \tag{13}$$

where $-$ means that no action should be done (on the tape). The unique temporal behavior of this deterministic hybrid system is now obvious since H_τ can just make a succession of internal read and write actions. At each moment of its internal time scale which is not a synchronization point with the time scale of the output channel, i.e. at every $(2\,k + 1)\,\tau$ with $k \in \mathbb{N}$, the system uses the `read` function to memorize the value on the input channel inside a fixed cell of its internal memory (such a moment is always a synchronization point with the time scale of the input channel). On the other hand, the systems outputs – with the `write` function – the value currently stored in this cell at each synchronization point with the output time scale, i.e. at every $2\,k\,\tau$ with $k \in \mathbb{N}$, thus producing a discrete sequence out of a continuous one. ■

Example 3.16 (Modulator) The action of a modulator is more or less reciprocal to that of a sampler and consists in converting a discrete sequence of real numbers into a continuous function by making use – at a discrete rate – of a pulse shape $p_\tau(t)$, which will be considered here – for the sake of simplicity (see below for more details on this hypothesis) – as a given function with interval $[0, \tau]$ as support. A modulator can then be modelled by an elementary deterministic hybrid system Mod_{p_τ} which is parameterized by this last continuous function. The input, output and memory domains of such a system are equal to ${}^*\mathbb{R}$, when its internal state set has exactly four states r, b, w and a that correspond to the four allowed internal actions of the system (reading on the input channel, making a blank action – i.e. nothing – , writing on the output channel, adding the value 1 in the internal memory), that is to say:

$$In(Mod) = Out(Mod) = Mem(Mod) = {}^*\mathbb{R}\,,\ State(Mod) = \{r, b, w, a\}\,.$$

The input time scale of Mod_{p_τ} is then a discrete time scale of time step τ, when its internal and output time scales are both continuous with respective time steps $dt_s = \tau/(2\,N)$ and $dt_o = \tau/N$ where N stands for a fixed infinitely great positive integer within ${}^*\mathbb{N}$. All the cells of the internal

memory of Mod_{p_τ} are initialized with 0 values and the control is initially put on the 1-cell. The control mechanisms of this system are then given by the following transition functions:

$$\begin{aligned} \texttt{read}\,(r\,;y) &= (b\,;y)\,, \\ \texttt{write}\,(w\,;(x,y)) &= (a\,;y\cdot p_\tau(x\,dt_o))\,, \\ \delta(b;y;1) &= (w;y;0) \\ \delta(a;x;0) &= \begin{cases} (w;x+1;0) & \text{if } x\neq N-1\ , \\ (r;0;1) & \text{if } x=N-1\ . \end{cases} \end{aligned} \tag{14}$$

Note that this system makes only use of the two first cells of its internal memory: the 0-cell is used to store internal computations (here the sequence of all non standard integers from 0 to $N-1$) when the 1-cell stores the input values. The unique temporal behavior of Mod_{p_τ} consists then to read a value of its input channel (at each possible synchronization point) and store it in the 1-cell of its internal memory, to make a blank operation (i.e. doing nothing during one single internal clock tip) and then to apply alternatively a `write` operation and a δ controlled transition (up to reading a new entry value and re-entering in the same processing cycle).

Note also that though the pulse shape p_τ in the above example gives us a degree of freedom in the way we can modulate the continuous output, the most realistic choice is unfortunately just to take $p_\tau(t)=1$ for any $t\in[0,\tau]$. The examples that correspond to practical situations are indeed obtained if the pulse shape has support bigger than $[0,\tau]$, which requires to add several consecutive input values. This technique would however lead to a more complicated modelling, which explains why we restrained ourselves to the simpler system presented above. ■

3.3 An example of higher order system

To show the potentiality of our approach in practice, we will now model within our framework a simplified version of a radio transmission system (which appears to be an higher order system in the meaning of Definition 3.8).

Example 3.17 (Simplified radio transmission) We will now show how to model a communication transmitter taking messages from a buffer Buf – as described in Example 3.13 (whose notations will be taken here) – and transmitting them over a radio channel. To achieve this aim, we must first introduce an encoder component Enc which reads messages from the buffer, converts them into binary form and encodes the resulting sequence of bits – by blocks of N_b bits – into complex symbols [11]. Whenever the encoder has less than N_b bits to work on, it sends a write request to the buffer. The systemic organization of this component is given in Figure 6.

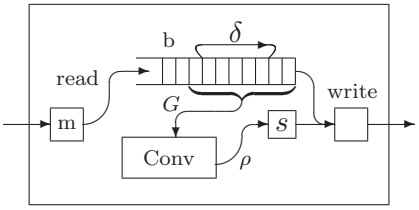


Figure 6: Description of the encoder component.

[11] The simplest example of such a process is the Binary Phase Shift Keying (BPSK) modulation protocol that converts every bit $b\in\mathcal{B}=\{0,1\}$ into the real number $(-1)^b$ (for more details, see for instance [26] or [16]).

The corresponding (non deterministic) system can indeed be defined as follows. We first suppose that the internal time scale of *Enc* has a discrete time step which is N_b times smaller than the rest of the global radio transmission system to which it will belong. The input domain of the encoder is just A, the same set of messages than the buffer to which it is connected. The output domain is taken to be $\mathbb{C} \times \{\varepsilon, \uparrow\}$ in order to model the fact that two types of output information can be sent on two different channels (i.e. a complex encoding or a write request). The memory domain is defined as $\{0,1\}^* \times \mathbb{C}$ since one must be able to store (on two different tapes) both sequences of bits and complex numbers. Finally the internal state set is reduced to two elements q_0 and q_1 (see the explanations below). These elements can be summarized by setting:

$$In(Enc) = A\,,\ Out(Enc) = \mathbb{C} \times \{\varepsilon, \uparrow\}\,,\ Mem(Enc) = \{0,1\}^* \times \mathbb{C}\,,\ State(Enc) = \{q_0,\, q_1\}\ .$$

We also suppose that *Enc* possesses a unique sub-system *Conv* – i.e. that $Sub(Enc) = \{Conv\}$ – which can convert blocks of N_b bits into complex symbols according to a given table and which has the same input time scale than *Enc* (the second state of *Enc* is in particular used to interconnect the reading of a bit on the tape of *Enc* with its sending to *Conv*). The control mechanisms of *Enc* can now be given by means of the following transition functions:

$$\begin{aligned}
\texttt{read}\,(q_0; m) &= (q_0; b \cdot \overline{m})\,, \\
G(q_0; (b, s)) &= (q_1; b_1)\,, \\
\delta(q_1; (b, s); 0) &= (q_0; (b \ll 1, s); 0)\,, \\
\rho(q_0; (b, s); y) &= (q_0; (b, y))\,, \\
\texttt{write}\,(q_0; (b, s)) &= \begin{cases} (q_0; (s, \uparrow)) & \text{if } |b| < N_b\ , \\ (q_0; (s, \varepsilon)) & \text{otherwise}\ , \end{cases}
\end{aligned} \tag{15}$$

where $m \in A$ and $\overline{m} \in \{0,1\}^*$ stand respectively for the message at the input of the encoder and for its binary form and where $(b, s) \in \{0,1\}^* \times \mathbb{C}$ is the current value of the internal memory of the system, with b denoting the sequence of bits that is currently stored on the main tape of the encoder and s being the complex symbol produced by the *Conv* sub-system. We also denoted above the concatenation product by $\cdot$, the first bit of the sequence b by b_1 and the shift of b by one bit by $b \ll 1$. Note finally that we did not represented here – for the sake of clarity – the precise behavior of δ on the main tape of *Enc* (that we modelled just as a single cell that can contain sequences of bits on which concatenation products can be directly applied).

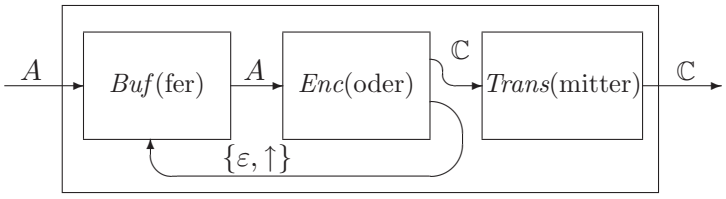


Figure 7: Graphical description of a simplified radio transmission chain.

The transmitter in its turn receives on the input a sequence of complex symbols $(s_k)_{k \geq 0}$ at a discrete rate and generates on the output a radio signal of the form

$$u(t) = \sum_{k=0}^{\infty} s_k\, p(t - t_k)\,,$$

where $p(t)$ is a pulse shape function and t_k is the moment when the k-th complex symbol is sent. This new operation can be realized by a modulator similar to that of Example 3.16. Composing now, as shown in Figure 7, the three components we introduced, one obtains the simplified radio transmitter system that we wanted to model.

Observe that in the above example, the nature of the first two components (that is to say Buf and Enc) is completely discrete, whereas that of the third component ($Trans$) is hybrid (discrete input with continuous output). This should be interpreted within the classification that we introduced in Section 3.2, as Buffer and Encoder are purely logical, whereas Transmitter serves as an interface between software and physical environments. ■

4 Conclusion

We tried to show in this paper that it is possible to construct a general unified theory of systems which may give a common framework to deal both with continuous and discrete systems (that are the two core kinds of systems used in engineering modelling). Note that our theory allows to take into account large classes of classical systems. For instance, (controlled) causal dynamical systems (see [12]) can be recovered within our framework as the standardization of suited physical-like deterministic systems, if one supposes that the observability function and the vector fields implied in the definition of such systems (see formula (III.2) of Section III.2 of [12]) are rationally analytic (which means that their Taylor development has only coefficients that are rational functions of their generic integer parameters) [12] . In the same way, synchronous systems – which are the discrete equivalents of causal dynamical systems – can also be modelled in our framework by software deterministic systems with the same input and output time scales.

However our approach does not reduce to re-interpret classical classes of systems. It also leads to the introduction of new classes of systems (such as the elementary software and physical systems that were discussed in Sections 3.2.1 and 3.2.2) and to lots of new questions (can one develop a Λ-calculus formalism for systems ? what are the "good" system sub-families ? can one construct a complexity theory for systems ? what are really the differences existing between deterministic and non-deterministic systems ? etc.) that should now be studied more in details in order to understand more deeply the notion of system.

Acknowledgements

The two authors would sincerely like to thank both Matthieu Martel and Marc Pouzet for the numerous discussions we had together during the writing of this paper, leading to a number of important suggestions and modifications of point of view.

[12] To recover this family of systems, one should transform the differential equation (III.2) of [12] into a finite difference equation that involves an infinitesimal time step. To see that this last equation expresses the functional behavior of a system, the key problem is then just to prove that any $A(q)$ can be computed by a system when A is a rationally analytic vector field. This last property can then be reduced to prove that (non standard) finite sums and products of non standard real numbers can be realized by a system, which can be done by using physical like systems (cf. Section 3.2.2). The only technical difficulty lies in fact in proving that one can design such a system for realizing the product of two non standard real numbers which can be done by implementing an adapted Euclidian algorithm (two non standard real numbers u and v being respectively infinitely closed to known multiples K and L of a given infinitesimal $1/N \in \mathbb{I}$ with $N \in {}^*\mathbb{N}$, one can find the non standard integer M such that $u\,v \approx M/N$ by computing the quotient of the non standard Euclidian division of $K\,L$ by N).

References

[1] Abrial J.R., *The B-book – Assigning programs to meanings*, Cambridge University Press, 1996.

[2] Alur R., Courcoubetis C., Halbwachs N., Henzinger T.A., Ho P.H., Nicollin X., Olivero A., Sifakis J., Yovine S., *The algorithmic analysis of hybrid systems*, Theor. Comp. Sci., 1995; **138** (1): 3–34.

[3] Barwise J., *Handbook of Mathematical Logic*, Studies in logic and the foundations of mathematics, **90**, North Holland, 1977

[4] Benveniste A., Caspi P., Edwards S.A., Halbwachs N., Le Guernic P., de Simone R., *The Synchronous Languages Twelve years later*, Proc. of the IEEE, Special issue on Embedded Systems, 2003; **91** (1): 64–83.

[5] Bérard B., Bidoit M., Laroussinie F., Petit A., Schnoebelen P., *Vérification de logiciels – Techniques et outils du model-checking*, Vuibert Informatique, 1999.

[6] Booch G., Rumbaugh J., Jacobson I.,, *The Unified Modeling Language User Guide*, Addision Wesley, 1999.

[7] Börger E., Stärk R., *Abstract state machines – A method for high-level system design and analysis*, Springer, 2003.

[8] Cha D.P., Rosenberg J.J., Dym C.L., *Fundamentals of Modeling and Analyzing Engineering Systems*, Cambridge University Press, 2000.

[9] Cutland N., *Nonstandard analysis and its applications*, London Mathematical Society Student Texts, **10**, Cambridge University Press, 1988

[10] D'Alembert, J. Le Rond dit, *Article "Différentiel"*, in Diderot D. et D'Alembert J. Le Rond dit, Eds.: Encyclopédie ou Dictionnaire raisonné des sciences, des arts et des métiers, Briasson, David, Le Breton et Durand, Paris, 1754, Tome **4**, pp. 985–989.

[11] Diener F. Reeb G., *Analyse non standard*, Hermann, 1989

[12] Fliess M., *Fonctionnelles causales non linaires et indtermines non commutatives*, Bull. Soc. Math. France, 1981; **109**: 3–40.

[13] Henzinger T.A., *The theory of hybrid automata*, in Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science, LICS'96, IEEE Society Press, 1996, pp. 278–292.

[14] Jensen C.U., Lenzing H., *Model Theoretical Algebra*, Gordon and Breach, 1989.

[15] Kowalewski S., Stursberg O., Fritz M., Graf H., Hoffmann I., Preussig J., Remelhe M., Simon S., Treseler H., *A case study in tool-aided analysis of discretely controlled continuous systems: the two tanks problem*, In Proceedings of the 5th International Workshop on Hybrid Systems, 1997, pp. 165–177.

[16] Krob D., Vassilieva E.A., *Performance analysis of modulation with diversity – A combinatorial approach II: Bijective methods*, Disc. App. Maths., 2005; **145**, (3): 403-421.

[17] Lakatos I., *Preuves et réfutations*, Hermann, 1984.

[18] Lakatos I., *Proofs and Refutations*, The British Journal for the Philosophy of Science, 1963-1964; Vol. XIV: 1–25, 120–139, 221–45, 296–342.

[19] Lavagno L., Martin G., Selic B., *UML for real – Design of embedded real-time systems*, Kluwer, 2003.

[20] Marwedel P., *Embedded systems design*, Kluwer, 2003.

[21] Mathworks, *Mathlab and Simulink*; `http://www.mathworks.com`.

[22] Meinadier J.P., *Ingénierie et intégration de systèmes*, Hermès, 1998.

[23] Meinadier J.P., *Le métier d'intégration de systèmes*, Hermès-Lavoisier, 2002.

[24] Meyer K.R., Hall G.R., *Introduction to Hamiltonian Dynamical Systems and the N-Body Problem*, Applied Mathematical Sciences, **90**, Springer Verlag, 1992.

[25] Mosterman P.J., *An overview of hybrid simulation phenomena and their support by simulation packages*, in Hybrid Systems: Computation and Control, Lectures Notes in Computer Science, Springer Verlag, 1999, pp. 165–177.

[26] J. Proakis, *Digital Communications*, 3rd Edition, McGraw-Hill, 1995.

[27] Robinson A., *Non Standard Analysis*, North-Holland, 1966

[28] Rumbaugh J., Jacobson I., Booch G., *The Unified Modeling Language Reference Manual*, Addision Wesley, 1999.

[29] Samad T., Balas G., Eds., *Software-Enabled Control*, John Wiley, 2003.

[30] Satzinger J.W., Jackson R.B., Burd S., Simond M., Villeneuve M., *Analyse et conception de systmes d'information*, Les éditions Reynald Goulet, 2003.

[31] Schneider K., *Verification of reactive systems – Formal methods and algorithms*, Springer, 2004.

[32] Schwartz L., *Théorie des distributions*, Hermann, 1966.

[33] Severance F.L., *System modeling and simulation – An introduction*, John Wiley, 2001.

[34] Sommerville I., *Software Engineering*, Addison Wesley, 6th Edition, 2001.

[35] Telelogic, *Doors*; `http://www.telelogic.com/index.cfm`.

[36] Wordsworth J.B., *Software engineering with B*, Addison-Wesley, 1996.

[37] Zaytoon J., Ed., *Systmes dynamiques hybrides*, Herms, 2001.

[38] Zeigler B.P., Praehofer H., Gon Kim T., *Theory of modeling and simulation – Integrating discrete event and continuous complex dynamic systems*, Academic Press, 2000.