Modeling of Complex Systems II: A minimalist and unified semantics for heterogeneous integrated systems

Marc Aiguier, Boris Golden, Daniel Krob

Modeling of Complex Systems II: A minimalist and unified semantics for heterogeneous integrated systems

Marc Aiguier

École Centrale Paris
Laboratoire de Mathématiques Appliqués aux Systèmes (MAS)
Grande Voie des Vignes - F-92295 Châtenay-Malabry
marc.aiguier@ecp.fr

Boris Golden[1], Daniel Krob

École Polytechnique et CNRS
Laboratoire d'Informatique de l'École Polytechnique (LIX)
91128 Palaiseau (France)
Phone: +33 (1) 69 33 40 73
Fax: +33 (1) 69 33 40 49
boris.golden@polytechnique.edu
dk@lix.polytechnique.fr

Abstract

The purpose of this paper is to contribute to a unified formal framework for complex systems modeling. To this aim, we define a unified semantics for systems including integration operators. We consider complex systems as functional blackboxes (with internal states), whose structure and behaviors can be constructed through a recursive integration of heterogeneous components. We first introduce formal definitions of time (allowing to deal uniformly with both continuous and discrete times) and data (allowing to handle heterogeneous data), and introduce a generic synchronization mechanism for dataflows. We then define a system as a mathematical object characterized by coupled functional and states behaviors. This definition is expressive enough to capture the functional behavior of any real system with sequential transitions. We finally provide formal operators for integrating systems and show that they are consistent with the classical definitions of those operators on transfer functions which model real systems.

Keywords: Complex systems, Systems modeling, Systems semantics, Systems Engineering, Systems integration, Timed Mealy machine, Hybrid time, Non-standard analysis

1. Introduction

The concept of *complex systems* has led to various definitions in numerous disciplines (biology, physics, engineering, mathematics, computer science, etc). One speaks for instance of dynamical, mechanical, Hamiltonian, hybrid, holonomic, embedded, concurrent or distributed systems (cf. [2, 4, 21, 24, 28, 32]). A minimalist fuzzy definition consistent with (almost) all

[1]Corresponding author

those of the literature is that a 'system' is "a set of interconnected parts forming an integrated whole", and the adjective 'complex' implies that a system has "properties that are not easily understandable from the properties of its parts". In the mathematical formalization of "complex systems", there are today two major approaches: the first one is centered on understanding how very simple, but numerous, elementary components can lead to complex overall behaviors (e.g. cellular automatas), the second one (that will also be ours) is centered on giving a precise semantics to the notion of system and to the integration of systems to build greater overall systems.

When mathematically apprehended, the concept of system (in the sense of this second approach) is classicaly defined with models coming from:

- control theory and physics, that deal with systems as partial functions (dynamical systems may also be rewritten in this way), called transfer functions, of the form:

  $$\forall t \in T,\ y(t) = F(x, q, t)$$

  where x, q and y are inputs, states and outputs dataflows, and where T stands for time (usually considered in these approaches as continuous (see [32, 1, 12]).

- theoretical computer sciences and software engineering, with systems that can be depicted by models equivalent to timed Turing machines with input and output, evolving on discrete times generally considered as a universal predefined sequence of steps (see for instance [19, 5, 16]).

However all these models do not easily allow to handle layered systems with multiple time scales. The introduction of a more evolved notion of time within Turing-like models involves many difficulties, mainly the proper definition of sequential transitions or the synchronization of different systems exchanging dataflows without synchronization of their time scales. Dealing with evolved definitions of times will generally imply to introduce infinity and infinitesimal (for instance with non-standard real numbers). There is therefore a great challenge (which we propose to address in this paper) on being able to unify in a same formal framework mathematical methods dealing with the design of both continuous and discrete systems.

The theory of hybrid systems was developed jointly in control theory (see [32, 34]) and in computer science (see [2, 3, 22]) to address this challenge. A serious issue with this theory is however that the underlying formalism has some troubling properties such as the Zeno effect which corresponds to the fact that an hybrid system can change of state an infinite number of times within a finite time (because of the convergence of series of durations) that one usually prefers to avoid in a robust modeling approach. Moreover, it does not allow to consider various time scales of heterogeneous granularity (which will be the central point of our approach). Other interesting and slightly different attempts in the same direction can also be found in Rabinovitch and Trakhtenbrot (see [27, 33]) who tried to reconstruct a finite automata theory on the basis of a real time framework, or in [35].

In the literature about (complex) systems, the real object and its model are often confused and both called "system". We will call a *real system* any object of the real world which transforms flows of data. We will call *system* the mathematical object introduced to model real systems. In this paper, we are interested in modeling the functional behavior of real systems, and their integration. Thus, we will model real systems as functional blackboxes (with an internal state), whose structure and behaviors can be described by the recursive integration of heterogeneous

smaller subsystems (thus considering complex systems as heterogeneous integrated systems). We will thus focus on two aspects of the complexity of systems:

- the *heterogeneity* of systems (modeled following continuous or discrete time, and exchanging data of different types, - informational, material or energetic).[2]
- the *integration* of systems, i.e. the mechanism to construct a system resulting from the composition of smaller systems, whose behaviors may be described at a more concrete level (i.e. a finer grain).

We will assume that the observational behavior of any real system can be modeled by a functional machine processing dataflows (for related work on dataflow networks, see [19, 11, 10]) in a way that can be encoded by timed transitions for changing states and outputs in instantaneous reaction to the inputs (comparable with timed Mealy machines [25] with possibly infinite states). We show that our formalization makes it possible to model the basic kinds of real systems (physical, software and human/organizational), which is especially important in Systems Engineering [6, 23, 30].

This paper is the second of a series on "Modeling of Complex Systems". Indeed, we generalize the approach of the first paper [7] (where a unified framework for continuous and discrete systems was defined by using non-standard infinitesimal and finite time steps) by dealing with time, data, and synchronization axiomatically, and by introducing integration operators. The purpose of this second paper is to give a unified and minimalist semantics for heterogeneous integrated systems and their integration. By "unified", we mean that we propose a unified model of real systems that can describe the functional behavior of heterogeneous systems and that is closed under integration. By "minimalist" we mean that our formalization intends to provide a small number of concepts and operators to model the behaviors and the integration of (complex) real systems. We believe that our work allows to give a relevant formal semantics for concepts and models typically used in Systems Engineering, where semi-formal modeling is well-spread. The paper is organized as follows:

- in Section 2 and 3, we introduce unified definitions of time (both continuous and discrete) and data (with various behaviors) to handle heterogeneous components and encompass classical approaches. We also define a generic synchronization for dataflows,
- in Section 4, we introduce a formal definition of systems as unified functional objects modeling heterogeneous real systems,
- in Section 5, we introduce minimalist operators for integrating systems (with closure of the definition of system) and prove that they are consistent with classical concepts of integration formalized on transfer functions.

2. Time

Most of the challenges raised by the unified definition of heterogeneous integrated systems are coming from time. Indeed, real systems are naturally modeled according to various time scales (modeling discrete or continuous time), and we must therefore be able to define:

[2] Data encompasses here all kinds of elements that can be exchanged between real objects. We distinguish three kinds of homogeneous systems : hardware/physical systems (transforming continuous physical parameters), software systems (transforming and managing discrete data), and human/organizational systems (organized through processes).

- a unified model of time encompassing continuous and discrete times to later introduce a unified definition of heterogeneous systems,
- the mixture of various time scales for integrating systems.

Unifying both discrete and continuous times can seem paradoxal (see [9] for an exhaustive survey on the subject). To reach this purpose, we propose to extend the approach developped in the first paper of the series [7] where discrete and continuous times have been unified homogeneously (by using techniques of non-standard analysis [26, 29, 15]). We propose a more generic approach and deal with time axiomatically, that is by expressing the minimal properties that both time references and time scales have to satisfy. That allows to consider in a same uniform framework many different times: usual ones such as $\mathbb{N}$ and $\mathbb{R}$, or more specific ones such as the non-standard real numbers ${}^*\mathbb{R}$, Harthong-Reeb's line [17, 18], or the VHDL time (see below).

2.1. Time references

A time reference is a universal time in which all systems will be defined.

Definition 2.1 (Time reference). *A* **time reference** *is an infinite set T together with an internal law $+^T : T \times T \to T$ and a pointed subset $(T^+, 0^T)$ satisfying the following conditions:*

- *upon T^+:*
  - $\forall a, b \in T^+,\ a +^T b \in T^+$ — *closure (Δ_1)*
  - $\forall a, b \in T^+,\ a +^T b = 0^T \Longrightarrow a = 0^T \wedge b = 0^T$ — *initiality (Δ_2)*
  - $\forall a \in T^+,\ 0^T +^T a = a$ — *neutral to left (Δ_3)*
- *upon T:*
  - $\forall a, b, c \in T,\ a +^T (b +^T c) = (a +^T b) +^T c$ — *associativity (Δ_4)*
  - $\forall a \in T,\ a +^T 0^T = a$ — *neutral to right (Δ_5)*
  - $\forall a, b, c \in T,\ a +^T b = a +^T c \Longrightarrow b = c$ — *cancelable to left (Δ_6)*
  - $\forall a, b \in T,\ \exists c \in T^+,\ (a +^T c = b) \vee (b +^T c = a)$ — *linearity (Δ_7)*

Elements of T are *moments* whilst elements of T^+ are *durations* (or distances between moments). Any duration can be considered as a moment, by considering a conventional origin.

Example 1. In the previous paper of this series [7], we chose for time reference the set of non-standard real numbers ${}^*\mathbb{R}$ defined as the quotient of real numbers $\mathbb{R}$ under the equivalence relation $\equiv \subseteq \mathbb{R}^{\mathbb{N}} \times \mathbb{R}^{\mathbb{N}}$ defined by:

$$(a_n)_{n\geq 0} \equiv (b_n)_{n\geq 0} \Longleftrightarrow m(\{n \in \mathbb{N} | a_n = b_n\}) = 1$$

where m is an additive measure that separates between each subset of $\mathbb{N}$ and its complement, one and only one of these two sets being always of measure 1, and such that finite subsets are always of measure 0. The obvious zero element of ${}^*\mathbb{R}$ is $(0)_{n\geq 0}$, ${}^*\mathbb{R}^+$ is its positive part taken here as durations, and the internal law + is defined as the usual addition on $\mathbb{R}^{\mathbb{N}}$, i.e.:

$$(a_n)_{n\geq 0} + (b_n)_{n\geq 0} = (a_n + b_n)_{n\geq 0}$$

${}^*\mathbb{R}$ satisfies all the conditions of Definition 2.1 and is a well-defined time reference. Observe also that ${}^*\mathbb{R}$ has as subset, the set of non-standard integers ${}^*\mathbb{Z}$ (and subsequently ${}^*\mathbb{N}$) where infinite numbers are all numbers having absolute value greater that any $n \in \mathbb{N}$. ◊

The properties given upon T and T^+ are constraints that catch the intuitive view that the time elapses linearly by adding successively durations between them.

Proposition 1 (Total order on a time reference). *We can define a total order $\preceq^T$ (later written $\preceq$ for convenience) on T as follows:*

$$a \preceq^T b \Leftrightarrow \exists c \in T^+,\ b = a +^T c$$

Proof. This is a classical result (in semigroups theory, cf [13]) using Δ_2, Δ_4, Δ_5, Δ_6 and Δ_7. □

Moreover, we can remark that Δ_1 insures that any element of T greater than an element of T^+ will be in T^+, and Δ_3 insures that 0^T is the minimum of T^+, so that the set of durations has natural properties according to $\preceq^T$ and can be understood as "positive" elements of T.

Some authors, e.g. [20], add commutativity and Archimedean properties in the definition of a time reference. Commutativity is intuitive and the Archimedean property excludes Zeno's paradox. However, they are not satisfied by the VHDL time used in some programming languages.

Example 2. The VHDL time [8] $\mathcal{V}$ is given by a couple of natural numbers (both sets of moments and durations are similar): the first number denotes the "real" time, the second number denotes the step number in the sequence of computations that must be performed at the same time – but still in a causal order. Such steps are called "δ-steps" in VHDL (and "micro-steps" in StateCharts). The idea is that when simulating a circuit, all independent processes must be simulated sequentially by the simulator. However, the real time (the time of the hardware) must not take these steps into account. Thus, two events e_1, e_2 at moments $(a, 1), (a, 2)$ respectively will be performed sequentially (e_1 before e_2) but at a same real time a. The VHDL addition is defined by the following rules:

$$(r' \neq 0) \Longrightarrow (r, d) + (r', d') = (r + r', d')$$
$$(r' = 0) \Longrightarrow (r, d) + (r', d') = (r, d + d')$$

where r, r', d and d' are natural numbers and + denotes the usual addition on natural numbers. Clearly, the internal law + above is not commutative, nor Archimedean: we may infinitely follow a δ-branch by successively adding δ-times.[3] ◊

2.2. Time scales

Time references give the basic expected properties of the set of all moments. Now, we want to define time scales, i.e. sets of moments of a time reference that will be used to define a system.

Definition 2.2 (Time scale). *A* **time scale** *is any subset $\mathbb{T}$ of a time reference T such that:*

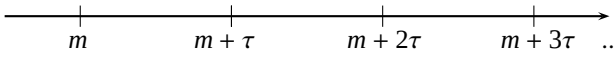
- *$\mathbb{T}$ has a minimum $m^{\mathbb{T}} \in \mathbb{T}$*
- *$\forall t \in T,\ \mathbb{T}_{t+} = \{t' \in \mathbb{T} \mid t \prec t'\}$ has a minimum called $succ^{\mathbb{T}}(t)$*
- *$\forall t \in T$, when $m^{\mathbb{T}} \prec t$, the set $\mathbb{T}_{t-} = \{t' \in \mathbb{T} \mid t' \prec t\}$ has a maximum called $pred^{\mathbb{T}}(t)$*

[3] This is not the intended use of VHDL time, however: VHDL computations should perform a finite number of δ-steps.

- *the principle of induction* [4] *is true on* $\mathbb{T}$.

The set of all time scales on T *is noted* $Ts(T)$.

A time scale is defined so that it will be possible to make recursive constructions on it, and to locate any moment of the time reference between two moments of a time scale. A time scale necessarily has an infinite number of moments. In fact, a time scale is expected to comply with the Peano axioms[5], excepted that the $succ^{\mathbb{T}}$ and $prec^{\mathbb{T}}$ are defined for moments of T and not only $\mathbb{T}$.[6] This is not equivalent: a simple counter-example on time reference $\mathbb{R}^+$ can show it is possible to have *prec* and *succ* properly defined for moments of the subset $\mathcal{T} = \{1 - \frac{1}{2^n} \text{for } n \in \mathbb{N}\} \cup \{1 + \frac{1}{2^n} \text{for } n \in \mathbb{N}\}$ whereas moment 1 has no *prec* or *succ* in $\mathcal{T}$. This fundamental property prevents Zeno's effect on any time scale. Most of time scales (discrete and continuous) used when modeling real systems can be defined as unified regular time scales of step τ and of minimum m:



Example 3. By using results of non-standard analysis, continuous time scales can then be considered in a discrete way. Following the approach developed in [7] to model continuous time by non-standard real numbers, a regular time scale can be ${}^*\mathbb{N}\tau$ where $\tau \in {}^*\mathbb{R}^+$ is the step, $0 \in {}^*\mathbb{N}\tau$ and $\forall t \in {}^*\mathbb{N}\tau$, $succ^{{}^*\mathbb{N}\tau}(t) = t + \tau$. This provides a discrete time scale for modeling classical discrete time (when the step is not infinitesimal) and continuous time (when the step is infinitesimal). ◊

Example 4. In the VHDL time $\mathcal{V}$, the internal law induces a lexicographic ordering on $\mathbb{N} \times \mathbb{N}$. Thus, let $\mathcal{W} \subset \mathcal{V}$ such that: $\forall a \in \mathbb{N},\ \exists N_a \in \mathbb{N},\ \forall (a, b) \in \mathcal{W}, b \leq N_a$ (i.e. there are only a finite number of steps at each moment of time in $\mathcal{W}$). Then $\mathcal{W}$ is a time scale in the VHDL time. ◊

Example 5. A time scale on the time reference $\mathbb{R}^+$ can be any subset A such that: $\forall t, t' \in \mathbb{R}^+,\ |A \cap [t; t + t']|$ is finite. ◊

One might also use the new language for representing finite, infinite and infinitesimal numbers introduced in [31] to deal with time.

We have shown that we can accomodate heterogeneous times with our definitions. We introduce a fundamental proposition allowing to unify different time scales, which is necessary for systems integration (when the systems involved do not share the same time scales). Overall, our definition of time will be suitable for heterogeneous integrated systems.

Proposition 2 (Union of time scales). *A finite union of time scales (on the same time reference T) is still a time scale.*

Proof. The proof for two time scales is enough. Let $\mathbb{T}_1, \mathbb{T}_2$ be two time scales on T. Let $\mathbb{T} = \mathbb{T}_1 \cup \mathbb{T}_2$. We want to prove that $\mathbb{T}$ is a time scale.
$\mathbb{T}$ is a subset of T. Note that $\mathbb{T}$ has a minimum $min(m^{\mathbb{T}_1}, m^{\mathbb{T}_2})$, and that the *succ* and *prec* functions can be obviously defined by: $\forall t \in T,\ succ^{\mathbb{T}}(t) = min(succ^{\mathbb{T}_1}(t), succ^{\mathbb{T}_2}(t))$ and when

[4] For $A \subset \mathbb{T}$, $(m^{\mathbb{T}} \in A\ \&\ \forall t \in A, succ^{\mathbb{T}}(t) \in A) \Rightarrow A = \mathbb{T}$.

[5] It can be easily checked that the above conditions imply Peano axioms.

[6] These specific properties are necessary to prove that time scales are closed under finite union.

$t > m^{\mathbb{T}}$, $prec^{\mathbb{T}} = max(prec^{\mathbb{T}_1}(t), prec^{\mathbb{T}_2}(t))$ [7]. So the only problem is to prove that the induction principle hold on $\mathbb{T}$. This can be proved by using a lemma: if $m^{\mathbb{T}} \in A \,\&\, \forall t \in A, succ^{\mathbb{T}}(t) \in A$ then $\forall t \in \mathbb{T}_i$, $succ^{\mathbb{T}_i}(t) \in \mathbb{T}_i$ for $i = 1, 2$ (this lemma is easily proved using the principle of induction in $\mathbb{T}_i$ on intervals of successive elements of $\mathbb{T}_i$ in $\mathbb{T}$). So that finally, $\mathbb{T} = \mathbb{T}_1 \cup \mathbb{T}_2$ satisfies the principle of induction. Thus, $\mathbb{T}$ is a time scale on T. □

3. Data

Another challenge to address to model complex systems is the heterogeneity of data (modeling any element that can be exchanged between real systems) and of their synchronization between different time scales. We introduce datasets that will be used for defining data transmitted by dataflows. The dataflows will be used to describe variables of systems (inputs, outputs and states), and we define the synchronization of dataflows between time scales.

3.1. Datasets

Definition 3.1 (ϵ-alphabet). *A set D is an ϵ-**alphabet** if $\epsilon \in D$. For any set B, we can define an ϵ-alphabet by $\overline{B} = B \cup \{\epsilon\}$.*

The elements of an ϵ-alphabet are called *data* and ϵ is a universal blank symbol ϵ accounting for the absence of data (as the blank symbol in a Turing machine). An ϵ-alphabet can have an infinite number of data. A system dataset (also called dataset) is an ϵ-alphabet with the description of the behavior of the data (when read or written in a "virtual" buffer):

Definition 3.2 (System dataset). *A **system dataset** is a pair $\mathcal{D} = (D, \mathcal{B})$ such that:*

- *D is an ϵ-alphabet*
- *$\mathcal{B}$, called **data behavior**, is a pair (r, w) with $r : D \to D$ and $w : D \times D \to D$ such that[8]:*
  - $r(\epsilon) = \epsilon$ $\quad (R1)$
  - $r(r(d)) = r(d)$ $\quad (R2)$
  - $r(w(d, d')) = r(d')$ $\quad (R3)$
  - $w(r(d'), d) = d$ $\quad (W1)$
  - $w(w(d, d'), r(d')) = w(d, d')$ $\quad (W2)$

$\mathcal{B}$ will be useful to synchronize dataflows defined on different time scales (see Projection below). Data behaviors can be understood as the functions allowing to read and write data in a "virtual" 1-slot[9] buffer defining how this synchronization occurs at each moment of time:

- when a buffer is read, what is left (depending on the nature of data, it can partially vanish)
- when a new data is written (second parameter of w), knowing the current content of the buffer (first parameter of w), what is the new content of the buffer (depending on the nature of data and the new incoming data, it can be partially or totally modified).

[7] for convenience of writing, we assume that if $prec^{\mathbb{T}_i}$ is not well defined for its argument, its value is $m^{\mathbb{T}}$.

[8] These axioms give a relevant semantics and are necessary to define consistent projections of dataflows on time scales.

[9] 1-slot means that the buffer can contain only one data. This data will be used to compute the value of a dataflow at any moment of a time scale, to be able to synchronize a dataflow with any possible time scale.

In this context, the conditions on r and w can be understood as follows:

- (R1): reading an empty buffer (i.e. containing ϵ) results in an empty buffer
- (R2): reading the buffer once or many times results in the same content of the buffer
- (R3): reading a buffer in which a data has just been written results in the same content whatever the initial content of the buffer was before writing the data
- (W1): when the buffer has just been read, the new data erases the previous one
- (W2): when the buffer has just been written with a data, it will not be modified if it is again written with the result of the reading function on this same data[10]
- we also have by (R1) + (W1): $w(\epsilon, d) = d$ (W3). When an empty buffer is written with a new data, the buffer contains this new data.

There are two classical examples of data behaviors when modeling real systems:

Example 6. [Persistent data behavior] In this case, data cannot be consummed by a reading, and every writing erases the previous data (this data behavior was the only one used in [7]) :

$$r(d) = d \text{ and } w(d, d') = d'$$

$\Diamond$

Example 7. [Consumable data behavior] In this case, data is consummed by a reading, and every writing (excepted when it is ϵ) erases the previous data :

$$r(d) = \epsilon \text{ and } w(d, d') = \begin{cases} d & \text{if } d' = \epsilon \\ d' & \text{else} \end{cases}$$

$\Diamond$

We give a less classical example of data behavior that can be used to represent the ability to accumulate data received (what can be meaningful when data are written more frequently than read). It is important to notice that the buffer is still a 1-slot buffer and that all accumulated data will be consumed entirely by a single reading[11].

Example 8. [Accumulative data behavior] Let A be a non-empty set and $D = \mathcal{P}(A)$ be the set of subsets of A. We consider that $\epsilon = \emptyset$, so that D is an ϵ-alphabet[12]. In this case, data is consumed by a reading, and every writing is added (using internal law of D, here $\cup$) to the previous data :

$$r(d) = \epsilon \text{ and } w(d, d') = d \cup d'$$

$\Diamond$

The same real data can be modeled using different behaviors: for instance, an electric current might be measured by a number of electrons at each step of a time scale (consumable behavior, data expressed as a natural number), or by a continuous flow of electrons (persistent behavior, data expressed as a real number in amperes). Thus, a data behavior is not an intrinsic property of the real data it models, but a modeling choice.

[10] This rule will insure that a dataflow projected on a finer time scale is equivalent to the initial dataflow.

[11] Modeling another kind of reading shall be modeled by buffers in the system itself, this is not the purpose of these "virtual" buffers dedicated to synchronization of data between different time scales.

[12] We can extend this example to any unital magma of identity element ϵ.

3.2. Dataflows

In what follows, $\mathcal{D}$ will stand for a dataset of ϵ-alphabet D with behaviors (r_D, w_D). A dataflow is a flow defined at the moments of a time scale carrying data of a dataset. It will be used to define the evolution of states, inputs and outputs of a system.

Definition 3.3 (Dataflow). *A* **dataflow** *over ($\mathcal{D}$,$\mathbb{T}$) is a mapping $X : \mathbb{T} \to D$.*

Definition 3.4 (Sets of dataflows). *The set of all dataflows over ($\mathcal{D}$,$\mathbb{T}$) is noted $\mathcal{D}^{\mathbb{T}}$. The set of all dataflows over $\mathcal{D}$ with any possible time scale on time reference T is noted $\mathcal{D}^T = \bigcup_{\mathbb{T} \in Ts(T)} \mathcal{D}^{\mathbb{T}}$.*

The projection of a dataflow on a time scale makes it possible to synchronize data exchanges between two different time scales, with the rule that a data arriving at t will be read at the first next moment on the time scale of projection (the computation of this synchronization only requires a 1-slot virtual buffer and data behaviors). It will be essential when composing together systems using different time scales to define the properties of the exchange of data.

Definition 3.5 (Projection of a dataflow on a time scale). *Let X be a dataflow on ($\mathcal{D}$, $\mathbb{T}_X$) and $\mathbb{T}_P$ be a time scale. Let $\mathbb{T} = \mathbb{T}_X \cup \mathbb{T}_P$. Let $\mathbb{T}'_P = succ^{\mathbb{T}}(\mathbb{T}_P)$[13]. We define recursively the buffer function $b : \mathbb{T} \to D$ by* [14]*:*

- *(P1) if $t \in \mathbb{T}_X \setminus \mathbb{T}'_P$, $b(t) = w(b(prec^{\mathbb{T}}(t)), X(t))$*
- *(P2) if $t \in \mathbb{T}'_P \setminus \mathbb{T}_X$, $b(t) = r(b(prec^{\mathbb{T}}(t)))$*
- *(P3) if $t \in \mathbb{T}_X \cap \mathbb{T}'_P$, $b(t) = X(t)$*
- *(P4) if $t \in \mathbb{T}_P \setminus (\mathbb{T}_X \cup \mathbb{T}'_P)$, $b(t) = b(prec^{\mathbb{T}}(t))$*

The ***projection*** *$X_{\mathbb{T}_P}$* ***of*** *X* ***on*** *$\mathbb{T}_P$ is then the dataflow on ($\mathcal{D}$, $\mathbb{T}_P$) defined by setting $X_{\mathbb{T}_P}(t) = b(t)$ for every $t \in \mathbb{T}_P$.*

Note: (P1) occurs when a new data is received, and when the data on the buffer has not been read at the previous step so does not need to be processed with the reading function. (P2) occurs when no new data is received, and when the data on the buffer has been read at the previous step and so needs to be processed in the buffer with the reading function. (P3) occurs when a new data is received, and when the data on the buffer has been read at the previous step, so that the content of the buffer is the new data, by condition (W1). Finally, (P4) occurs when no new data is received, and when the data on the buffer has not yet been read, so that nothing changes.

We define equivalent dataflows as dataflows that cannot be distinguished by any projection.

Definition 3.6 (Equivalent dataflows). *The dataflows X and Y are* ***equivalent*** *(noted $X \sim Y$) if, and only if:*

$$\text{for any time scale } \mathbb{T} \text{ on } T,\ X_{\mathbb{T}} = Y_{\mathbb{T}}$$

[13] Corresponding to moments such that a data has been read at the previous moment and shall be marked as "read".

[14] By convention, $b(prec^{\mathbb{T}}(m^{\mathbb{T}})) = \epsilon$, which makes it simpler to define the rules of projection without making a special case when $t = m^{\mathbb{T}}$.

Definition 3.7 (Equivalent dataflows as far as). *The dataflows X and Y are **equivalent as far as** $t_0 \in T$ (noted $X \sim_{t_0} Y$) if, and only if:*

$$\textit{for any time scale } \mathbb{T} \textit{ on } T, \textit{ for all } t \preceq t_0 \textit{ in } \mathbb{T}, \;\; X_{\mathbb{T}}(t) = Y_{\mathbb{T}}(t)$$

We now introduce two propositions insuring the relevancy of the projection of dataflows.

Proposition 3 (Equivalence of the projection on a finer time scale). *Let X be a dataflow on $(\mathcal{D}, \mathbb{T}_X)$ and let $\mathbb{T}_P$ be a time scale such that $\mathbb{T}_X \subseteq \mathbb{T}_P$. Then, we have:*

$$X \sim X_{\mathbb{T}_P}$$

Proof. The proof uses the properties of the data behaviors and the principle of induction on time scales to show that: $\forall \mathbb{T} \in Ts(T),\ X_{\mathbb{T}} = (X_{\mathbb{T}_P})_{\mathbb{T}}$, and thus $X \sim X_{\mathbb{T}_P}$ (by definition of $\sim$). Let $\mathbb{T}$ be a time scale. We note $P = X_{\mathbb{T}_P}$ and want to prove that $X_{\mathbb{T}} = P_{\mathbb{T}}$.

Let b_X be the buffer (defined for moments of $\mathbb{T}_X \cup \mathbb{T}$) used for defining the projection of X on $\mathbb{T}$ and b_P be the buffer (defined for moments of $\mathbb{T}_P \cup \mathbb{T}$) used for defining the projection of P on $\mathbb{T}$. We will prove by induction on $\mathbb{T}$ that: $\forall t \in \mathbb{T},\ b_X(t) = b_P(t)$.

(a) First, we want to prove that the equality is true for moments before $m^{\mathbb{T}_X}$. Let $t_0 \in \mathbb{T}$ with $t_0 \prec m^{\mathbb{T}_X}$ (we will suppose without loss of generality[15] that $m^{\mathbb{T}} \prec m^{\mathbb{T}_X}$).

- $X_{\mathbb{T}}(m^{\mathbb{T}}) = \epsilon$ by (P0) + (P4) to initialize the buffer.
- for P:
  - if $m^{\mathbb{T}} \prec m^{\mathbb{T}_P}$, then $P_{\mathbb{T}}(m^{\mathbb{T}}) = \epsilon$ by (P0) and (P4)
  - else: we have $\forall t \in \mathbb{T}_P \mid t \prec m^{\mathbb{T}_X}$, $P(t) = X_{\mathbb{T}_P}(t) = \epsilon$ by (P0) and (P4) since $m^{\mathbb{T}} \prec m^{\mathbb{T}_X}$. As $w(\epsilon, \epsilon) = \epsilon$ by (W3) and $r(\epsilon) = \epsilon$ by (R1), the buffer till $m^{\mathbb{T}}$ in the projection of P on $\mathbb{T}$ is alway equal to ϵ and we have $P_{\mathbb{T}}(m^{\mathbb{T}}) = \epsilon$.
- finally, $X_{\mathbb{T}}(m^{\mathbb{T}}) = P_{\mathbb{T}}(m^{\mathbb{T}})$.
- we can extend the proof by induction to any $t_0 \in \mathbb{T}$ with $t_0 \prec m^{\mathbb{T}_X}$ since $\forall t \in \mathbb{T}_P$ such that $t \prec m^{\mathbb{T}_X}$, $P(t) = \epsilon$.

(b) For the case where $t = m^{\mathbb{T}_X}$, we have $b_X(t) = b_P(t) = X(m^{\mathbb{T}_X})$, which will allow to initiate the induction.

(c) Then, we want to prove that the induction hypothesis can be used on $\mathbb{T}$. Let $t_0 \in \mathbb{T}$ with $t_0 \geq m^{\mathbb{T}_X}$ such that $b_X(t_0) = b_P(t_0)$ and $t_1 = succ^{\mathbb{T}}(t_0)$. We want to prove that $b_X(t_1) = b_P(t_1)$. Let $A =]t_0; t_1] \cap \mathbb{T}_P$ [16] and $B =]t_0; t_1] \cap \mathbb{T}_X$ (we have $B \subseteq A$ since $\mathbb{T}_X \subseteq \mathbb{T}_P$). Let $t_x = prec^{\mathbb{T}_X}(succ^{\mathbb{T}_X}(t_0))$[17]. Three cases are to be considered:

- (1.1) if $B = \emptyset$ and $A = \emptyset$
  - for $b_X(t_1)$: by (P2) we have $b_X(t_1) = r(b_X(t_0))$

[15] We may add if necessary a smaller initial moment to $\mathbb{T}$.

[16] This notation is intended to capture the elements of $\mathbb{T}_P$ greater than t_0 and less than or equal to t_1.

[17] t_x is the latest moment of T_X before or at t_0. It exists since $t_0 \geq m^{\mathbb{T}_X}$.

- for $b_P(t_1)$: by (P2) we have $b_P(t_1) = r(b_P(t_0)) = r(b_X(t_0))$
- and so $b_X(t_1) = b_P(t_1)$.

- (1.2) if $B = \emptyset$ and $A \neq \emptyset$, two subscases shall be considered
  - (1.2.1) if $t_x = t_0$, then
    - for $b_X(t_1)$: by (P2) we have $b_X(t_1) = r(b_X(t_0))$. According to the situation: by (P3) $b_X(t_0) = X(t_0) = w(\epsilon, X(t_0))$ or by (P1) $b_X(t_0) = w(\ldots, X(t_0))$. Anyway, $b_X(t_0) = w(\ldots, X(t_0))$, and by (R3) we have $b_X(t_1) = r(X(t_0))$.
    - for $b_P(t_1)$: $\forall t \in A,\ P(t) = r(X(t_0))$, using (P2) and (R2) in the definition of P as the projection of X on $\mathbb{T}_P$. So, by (P1) we have $b_P(m^A) = P(m^A) = r(X(t_0))$. As $w(r(d'), d) = d$ by (W1), we have applying (P1) for all $t \in A$, $b_P(t) = r(X(t_0))$. If $t_1 \notin A$, we apply (P4) and get: $b_P(t_1) = r(X(t_0))$, and the result is the same if $t \in A$.
    - and so $b_X(t_1) = b_P(t_1)$.
  - (1.2.2) if $t_x \prec t_0$, then
    - for $b_X(t_1)$: by (P2) we have $b_X(t_1) = r(b_X(t_0))$. But $b_X(t_0)$ can be, by (P2) and (P4), expressed recursively as $b_X(t_0) = r(b_X(t_X))$. Whatever the situation, as in (1.2.1) we can write $b_X(t_x) = w(\ldots, X(t_x))$ and so by (R3) we have $b_X(t_0) = r(b_X(t_X)) = r(X(t_x))$.
    - for $b_P(t_1)$: the proof is exactly the same as in (1.2.1).
    - and so $b_X(t_1) = b_P(t_1)$.

- (1.3) if $B \neq \emptyset$ then $A \neq \emptyset$. We show as in (1.2) that $b_X(m^B) = b_P(m^B)$, and we conclude recursively that $b_X(t_1) = b_P(t_1)$ using (W2).

(d) Finally, by induction, we have $\forall t \in \mathbb{T},\ b_X(t) = b_P(t)$ (begining the induction at $t = m^{\mathbb{T}_X}$, the anterior t being handled by the first case (a)).

Hence, $X_{\mathbb{T}} = P_{\mathbb{T}} = (X_{\mathbb{T}_P})_{\mathbb{T}}$ for any time scale $\mathbb{T}$. Thus $X \sim X_{\mathbb{T}_P}$ (by definition of $\sim$) which proves our result. □

Proposition 4 (Equivalence of projections on nested time scales). *Let X be a dataflow and let $\mathbb{T}_A \subseteq \mathbb{T}_B$ be two nested time scales. Then, we have:*

$$(X_{\mathbb{T}_B})_{\mathbb{T}_A} = X_{\mathbb{T}_A}$$

Proof. This technical proof is very similar to the previous one. □

4. Systems

We introduce the definition of a system as a timed Mealy machine, and show that it can be represented by a transfer function.

4.1. Formal definition of a system

We define a system as a mathematical object (figuring a functional black box with an internal state[18]), characterized by coupled functional and states behaviors (defining step by step transitions for changing state and output in instantaneous reaction to the input received).

Definition 4.1 (System). *A **system** is a 7-tuple* $\int = (\mathbb{T}_s, Input, Output, S, q_0, \mathcal{F}, Q)$ *where*

- $\mathbb{T}_s$ *is a time scale called the time scale of the system,*
- $Input = (In, \mathcal{I})$ *and* $Output = (Out, O)$ *are datasets, called input and output datasets,*
- S *is a non-empty* ϵ*-alphabet* [19]*, called the* ϵ*-alphabet of states,*
- q_0 *is an element of* S*, called initial state,*
- $\mathcal{F} : In \times S \times \mathbb{T}_s \to Out$ *is a function called functional behavior,*
- $Q : In \times S \times \mathbb{T}_s \to S$ *is a function called states behavior.*

$(Input, Output)$ *is called the signature of* $\int$*.*

Our definition of system can be understood as a timed Mealy machine, i.e. a Mealy machine [25] where we have introduced time[20], and where the set of states is not supposed to be finite (what is a fundamental difference from the point of view of computability).

$\mathbb{T}_s$ represents the moments of "life" of the system, i.e. the moments where state and output can change in the behavior of the system, and where input is read (the system will necessarily read its input at each moment of its time scale, the virtual buffer being only used to synchronize the received dataflow with the time scale of the system[21]). The state of a system is all information "inside" the system, allowing to define its instantaneous behavior according to inputs and time.

$\mathcal{F}$ and Q compute respectively the output and the current state of a system, from its last defined state, its current input (the input can therefore have an instantaneous influence on output and state) and the moment of time considered. Introducing time in the transition functions is necessary so that the system has information about time to make transitions only at moments on its time scale. Defining the system just as a sequential behavior on its time scale (which is only *one* possible time scale in the time reference) without knowledge of time would make it difficult to compose meaningfully this system with another system having a sequential behavior on another time scale, so that the composition can still be expressed as a system[22].

The introduction of time defines, from the point of view of computability, a recursive hierarchy of systems following a recursive hierarchy of time scales on a given time reference. We will develop this point in future work.

We then define the dynamic execution of a system allowing to transform (step by step) an input dataflow into an output dataflow, while defining a state dataflow.

[18] The properties of this internal state are decisive to study computability (but out of the scope of this paper).

[19] Defining S as an ϵ-alphabet (therefore containing ϵ) and not just as a set will make it possible to define a dataflow of states, what will later be convenient.

[20] The introduction of time is a fundamental difference as it makes it possible to use hybrid times (and corresponding heterogeneous systems) and to define synchronizations between different systems.

[21] Buffers of the system can be defined inside the system itself.

[22] Defining the product of 2 systems with different time scales as a system requires to define them on a shared time scale, what perturbates the initial time scale of each system and makes it impossible to define a step by step behavior of the resulting system without knowledge of time (or without introducing tricky states to "count" the number of moments).

Definition 4.2 (Execution of a system). *Let $\int$ be a system. Let $X \in In^T$ be an input dataflow* [23] *for $\int$ and $\tilde{X} = X_{\mathbb{T}_s}$. The **execution of $\int$ on the input dataflow** X is the 3-tuple (X, Q, Y) where*

- *$Q \in S^{\mathbb{T}_s}$ is recursively defined by* [24]*:*
  - $Q(m^{\mathbb{T}_s}) = Q(\tilde{X}(m^{\mathbb{T}_s}), q_0, m^{\mathbb{T}_s})$
  - $\forall t \in \mathbb{T}_s,\ Q(succ^{\mathbb{T}_s}(t)) = Q(\tilde{X}(succ^{\mathbb{T}_s}(t)), Q(t), succ^{\mathbb{T}_s}(t))$ [25]
- *$Y \in Out^{\mathbb{T}_s}$ is defined by:*
  - $Y(m^{\mathbb{T}_s}) = \mathcal{F}(\tilde{X}(m^{\mathbb{T}_s}), q_0, m^{\mathbb{T}_s})$
  - $\forall t \in \mathbb{T}_s,\ Y(succ^{\mathbb{T}_s}(t)) = \mathcal{F}(\tilde{X}(succ^{\mathbb{T}_s}(t)), Q(t), succ^{\mathbb{T}_s}(t))$

X, Q and Y are respectively input, state and output dataflows.

4.2. Transfer functions

Functional behaviors (between inputs and outputs) of systems are given by "causal" functions transforming dataflows, i.e. functions whose behavior is deterministic and only depending on data received in the past (not in the future).

Definition 4.3 (Transfer function). *Let Input and Output be two datasets and let $\mathbb{T}_s$ be a time scale. A function $F : Input^T \rightarrow Output^{\mathbb{T}_s}$ is a (causal) **transfer function** on time scale $\mathbb{T}_s$ of signature $(Input, Output)$ if, and only if:*

$$\forall X, Y \in Input^T,\ \forall t \in T,\ (X_{\mathbb{T}_s} \sim_t Y_{\mathbb{T}_s}) \Rightarrow (F(X) \sim_t F(Y))$$

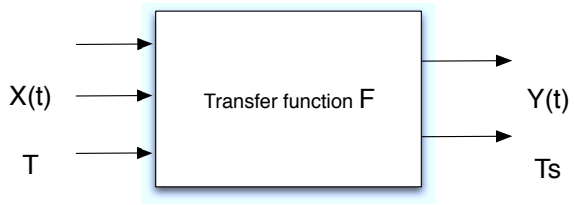


Figure 1: A transfer function

A transfer function is a classical and "universal" representation (see [32, 12]) of any functional behavior (i.e. an object that receives and sends data within time). We will show that every system induces a transfer function, and later that integration operators defined on systems corresponds to integration operators on transfer functions.

Equivalent transfer functions are transfer functions which cannot be distinguished.

[23] The system cannot restrict the possible dataflows on *Input* it will receive, what is a safer modeling principle.

[24] The ϵ-alphabet of states S is associated with a persistent behavior, since the state of a system at any moment of the time reference can be obtained by considering its last defined state.

[25] Defining the current state as the state after the state transition allows to model meaningful real states behaviors with an instantaneous influence of the input on the state.

Definition 4.4 (Equivalence of transfer functions). *Let F_1 and F_2 be two transfer functions sharing the same signature. F_1 and F_2 are **equivalent** (noted $F_1 \sim F_2$) if, and only if:*

$$\forall X \in Input^T,\ F_1(X) \sim F_2(X)$$

A unique transfer function can be associated with any system. It describes the functional behavior of this system.

Theorem 1 (Transfer function of a system). *Let $\int$ be a system. There exists a unique [26] transfer function $F_\int$, called **the transfer function of** $\int$ and such that: for all input dataflow X of $\int$, $F_\int(X)$ is the output dataflow in the execution of $\int$.*

Proof. Let $F_\int : In^T \to Out^{\mathbb{T}_s}$ be the function defined by setting for every $X \in In^T$, $F_\int(X)$ as the unique output dataflow $Y \in Out^{\mathbb{T}_s}$ corresponding to the input dataflow X in the execution of $\int$. Let $X, Y \in In^T$ be two input dataflows for the system $\int$. If $X \sim_{t_0} Y$ for some $t_0 \in T$, then by definition we have $X_{\mathbb{T}_s}(t) = Y_{\mathbb{T}_s}(t)$ for every $t \preceq t_0$ in $\mathbb{T}_s$. The execution of a system for an input dataflow only depends on the projection of this dataflow on the time scale of the system. By definition of the execution of a system, the output of this system at t only depends on inputs received until t (included), and so $F_\int(X) \sim_{t_0} F_\int(Y)$. Thus, $F_\int$ is a transfer function. □

A system can then be represented as in Figure 2 (where the white squares on the left account for the "virtual" buffers projecting the input dataflow on the time scale $\mathbb{T}_s$ of the system).

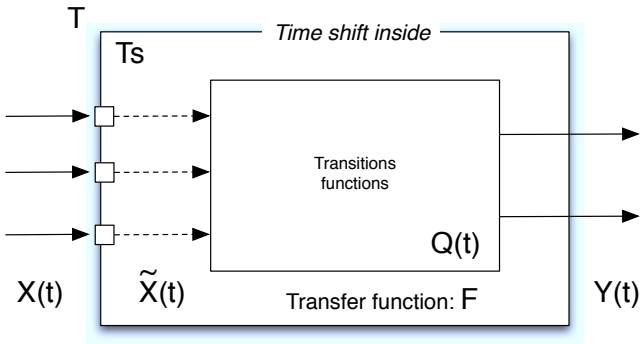


Figure 2: A system

4.3. Examples of systems

Example 9. [Nondeterministic systems] We have defined systems a deterministic objects. It will however be useful to simulate nondeterministic behaviors within our model. Nondeterministic behaviors can be modeled in a system as follows: one of the input $\mathcal{E}$ can be used as an oracle (or a dataflow of *events*), i.e. an input giving information to the system to make its transitions (functional and states).

[26] Unique on time scale $\mathbb{T}_s$, and up to equivalence on all time scales.

It can simulate classical nondeterminism of Mealy machines (where functional and states behaviors have their value in nonempty subsets of the target datasets of their deterministic version) by indicating at each step which element to chose within this subset (so that the nondeterministic behavior is simulated by a dataflow of events within a deterministic system). This flow can also be understood as the formalization of the imperfection/underspecification of any deterministic model[27]. It is therefore possible to take into account this imperfection by considering that transitions can be influenced by specific events carried by the oracle.

This kind of "events dataflow" typically corresponds to the events used in States diagrams in modeling languages like SysML (where an event as "the water tank is full" will in fact correspond to expressing in a deterministic way a nondeterministic event that cannot be computed from the current input or state of the system). ◊

We now give examples of modeling of the three kinds of real systems (physical, software and human) in our framework.

Example 10. [Software system] A software system can be modeled as a Turing machine with input and output, whose transitions are made following a time scale. In our model, the state of a system contains the memory of the Turing machine, its logical state, and its RW-head's position.

We consider a classical Turing machine with input and output. Let Q^{Tur} be the finite, nonempty set of logical states of the Turing machine, q_{Tur0} be the initial logical state, Σ be the ϵ-alphabet of internal tape symbols, In and Out be the sets of input and output, and $\delta : Q^{Tur} \times \Sigma \times In \to Q^{Tur} \times \Sigma \times Out \times \{-1, 0, 1\}$ the transition function (separated into 4 projections δ_1, δ_2, δ_3, δ_4 respectively on Q_{Tur}, Σ, Out and $\{-1, 0, 1\}$).

We define the system $\int = (\mathbb{T}_s, Input, Output, S, q_0, \mathcal{F}, Q)$ simulating this Turing machine by:

- $\mathbb{T}_s$ is any time scale isomorphic to $\mathbb{N}$ (on any time reference)
- $Input = (In, \mathbb{I})$ where $\mathbb{I}$ is a behavior on In
- $Output = (Out, \mathbb{O})$ where $\mathbb{O}$ is a behavior on Out
- $S = \Sigma^{\mathbb{Z}} \times \overline{Q^{Tur}} \times \overline{\mathbb{Z}}$ [28]
- $q_0 = (\epsilon, q_{Tur0}, 0)$
- $\mathcal{F}(x, (tape, q_{Tur}, i), t) = \delta_3(k)$ where $k = q_{Tur}, tape[i], x \in Q^{Tur} \times \Sigma \times In$
- $Q(x, (tape, q_{Tur}, i), t) = (tape[i \leftarrow \delta_2(k)], \delta_1(k), i + \delta_4(k))$ [29]

This system will compute exactly the same outputs as the initial timed Turing machine with input and output. Thus, our model contains Turing-like models of software systems. ◊

[27] Note that this imperfection/underspectification can be on purpose, to define a simpler system. This point will be further discussed when introducing "abstraction" in Section 5. In practice, the possibility to model deterministic behaviors of a real system is often restricted by the limited grain of description of the real system state observed.

[28] In $\Sigma^{\mathbb{Z}}$, ϵ is the sequence $(\epsilon)_{\mathbb{Z}}$.

[29] Where $tape[i \leftarrow x]$ means replacing in the sequence $tape$ the i^{th} symbol with x.

Example 11. [Human system] A basic example of a human system may be an individual, John, in the context of its work. In our modeling, John has two states ("normal" or "tired"). He can receive requests (by phone) from its colleagues (he must answer them by "Yes" or "No") and can also receive energy (when eating for example, what makes him normal if he was tired). Lastly, John can become tired after receiving too many requests from its colleagues. John is a very helpful guy always ready to help people, but when he is tired, he only helps urgent requests.

In the scope of our story, John can be modeled as the following nondeterministic system[30]:

- we choose $\mathbb{T}_s = \mathbb{N}$ (each unit of time being a second)[31]
- $Input = (In, \mathcal{I})$ with $In = \{Urgent\_request, Request, \epsilon\} \times \{Energy, \epsilon\} \times \{Too\_many, \epsilon\}$ and $\mathcal{I}$ being the consumable behavior associated to In. Too_many is an event to model nondeterministic behaviors of the system at this abstraction level
- $Output = (Out, O)$ with $Out = \{Yes, No, \epsilon\}$ and O being the consumable behavior
- $S = \{Tired, Normal, \epsilon\}$ and $q_0 = Normal$
- $\mathcal{F}((x_1, x_2, e), q, t) \begin{cases} Yes & \text{if } x_1 = Request \ \& \ q = Normal \text{ or } x_1 = Urgent\_request \\ No & \text{if } x_1 = Request \ \& \ q = Tired \\ \epsilon & \text{else (i.e. } x_1 = \epsilon) \end{cases}$
- $Q((x_1, x_2, e), q, t) \begin{cases} Tired & \text{if } e = Too\_many \text{ or } q = Tired \ \& \ x_2 \neq Energy \\ Normal & \text{else} \end{cases}$

$\Diamond$

Example 12. [Physical system] It has been proved that any Hamiltonian system can be modeled within the framework introduced in [7]. As our definition of system generalizes the work of this first paper[32], we will recall a simplified example of a Water Tank given in [7], which is a well-known example of the hybrid systems and control theory literature.

We work in the time reference ${}^*\mathbb{R}$ of nonstandard real numbers. Let us fix first some regular continuous time scale $\mathbb{T}$ with infinitesimal time step τ. We consider a water tank where water arrives at a variable rate $w_i(t) \geq 0$ (with $t \in \mathbb{T}$) through one single pipe. The water leaves through another (output) pipe at rate $w_o(t)$ (with $t \in \mathbb{T}$) controlled by a valve whose position is given by $v(t) \in [0, 1]$ (with $t \in \mathbb{T}$), 0 and 1 modelling respectively here the fact that the valve is closed or open. The water tank can be modeled as a system, taking on input the current values of the incoming water flow $w_i(t)$ and the position v(t) of the valve and sending on its output the corresponding output water flow $w_o(t)$ and water level $l(t)$ according to the following equations:

$$w_o(0) = C\,V_0, \quad w_o(t+\tau) = C\,v(t) \ \text{ for every } t \in \mathbb{T}^*,$$
$$l(0) = L_0, \quad l(t+\tau) = l(t) + (w_i(t) - w_o(t))\,\tau \ \text{ for every } t \in \mathbb{T}^* \ .$$

[30] The nondeterminism allows to express in this high-level modeling the fact that John will become tired when having received "too many" requests (what cannot be expressed precisely at this abstraction level).

[31] The choice of the time scale will be especially important when composing this system with other systems having their own time scales. The hidden assumption here is that John cannot receive more than one phone request each second.

[32] It is out of the scope of this paper to prove it, however the proof is not difficult as one can notice that non-standard time scales defined in [7] are still time scales in our new model, and that transitions defined in [7] can be rewritten as transitions in our model.

The input and output spaces of the system are thus $In_T = \overline{[0, C]} \times \overline{[0, 1]}$ and $Out_T = \overline{[0, C]} \times \overline{[L_1, L_2]}$. This illustrates the modeling of a simple physical system in our framework. Modeling of more complex physical systems can be found in [7]. ♢

5. Integration operators

We propose three elementary operators allowing to model systems integration, i.e. to build greater systems from a set of elementary systems by recursive application of composition operators and abstraction operator.

5.1. Systems composition

Composition consists in aggregating systems together in an overall greater system where some inputs and outputs of the various systems have been interconnected. Composition requires to have a definition of the synchronization of dataflows between the different time scales of the systems considered. We assume that the transmission of data between systems is instantaneous.

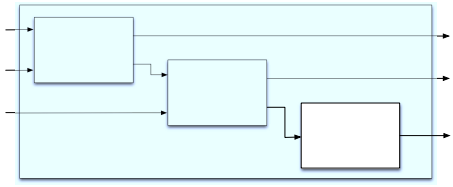


Figure 3: Composition of systems

We define two operators for systems composition: the *product* (allowing to define a new overall system from a set of systems, without interconnecting them) and the *feedback* (allowing to define a new system by interconnecting an input and an output of the same system). Dividing composition into two steps allows to distinguish between the aggregation of systems, and the interconnections within the new overall system, and makes it easier to prove theorems.

5.1.1. Extension

We first introduce a "technical" operator called *extension* that will facilitate the definition of the product by allowing to define on a shared time scale a finite[33] number of systems. The extension concentrates all technical difficulties (which are resulting from the introduction of time) in defining the composition of systems.

Definition 5.1 (Extension of a transfer function). *Let F be a transfer function of time scale $\mathbb{T}_s$ on signature $(Input, Output)$. The **extension of** F **to** $\mathbb{T}$ (such that $\mathbb{T}_s \subseteq \mathbb{T}$) is the equivalent transfer function F' of time scale $\mathbb{T}$ on signature $(Input, Output)$ defined by:*

$$\forall X \in Input^T,\ F'(X) = F(X)_{\mathbb{T}}$$

[33] Since it is not possible, in a generic time reference, to define a time scale from an infinite union of time scales.

The instantaneous behaviors transition functions of a system make it possible to extend the transition functions of a system to any moment of time, by the introduction of virtual extension buffers for input and output in the state (so that the new state is augmented with a data of input and a data of output, memorizing respectively the data to be received and emitted).

Definition 5.2 (Instantaneous behaviors transition functions). *Let $\int = (\mathbb{T}_s, Input, Output, S, q_0, \mathcal{F}, Q)$ be a system. We note w_i the writing function for Input and r_i, r_o the reading functions for Input and Output. Writing $x' = w_i(b_i, x)$* [34]*, we define the* ***instantaneous behaviors transition functions*** *of a system:*

$$\begin{array}{rcl} \tilde{\mathcal{F}} : In \times (S \times In \times Out) \times T & \longrightarrow & Out \\ (x, (q, b_i, b_o), t) & \longmapsto & \begin{cases} \mathcal{F}(x', q, t) & \text{if } t \in \mathbb{T}_s \\ r_o(b_o) & \text{else} \end{cases} \end{array}$$

and

$$\begin{array}{rcl} \tilde{Q} : In \times (S \times In \times Out) \times T & \longrightarrow & Out \\ (x, (q, b_i, b_o), t) & \longmapsto & \begin{cases} (Q(x', q, t), r_i(x'), \mathcal{F}(x', q, t)) & \text{if } t \in \mathbb{T}_s \\ (q, x', r_o(b_o)) & \text{else} \end{cases} \end{array}$$

The new transition functions $\tilde{\mathcal{F}}$ and $\tilde{Q}$ are defined for every moment of the time reference T and work with extended states containing virtual extension buffers allowing to synchronize inputs and outputs with the time scale of the system. These new transition functions can be restricted to any time scale $\mathbb{T}$, noted $\tilde{\mathcal{F}}_{\mathbb{T}}$ and $\tilde{Q}_{\mathbb{T}}$.
The extension of a system consists in defining it on a finer time scale (making it possible to define a finite number of systems on a shared time scale, i.e. the union of their time scales).

Definition 5.3 (Extension of a system). *Let $\mathbb{T}$ be a time scale such that $\mathbb{T}_s \subseteq \mathbb{T}$. The* ***extension of $\int$ to*** *$\mathbb{T}$ is the new system:*

$$\int_{\mathbb{T}} = (\mathbb{T}, Input, Output, S \times In \times Out\ ^{35}, (q_0, \epsilon, \epsilon), \tilde{\mathcal{F}}_{\mathbb{T}}, \tilde{Q}_{\mathbb{T}})$$

Theorem 2 (Equivalence of a system by extension). *Let $\int$ be a system and $\int_{\mathbb{T}}$ be its extension to a finer time scale. Then $\int$ and $\int_{\mathbb{T}}$ have equivalent transfer functions:*

$$F_{\int} \sim F_{\int_{\mathbb{T}}}$$

Moreover, the state dataflows in their execution are equivalent when projected on the initial ϵ-alphabet of states S.

Proof. Let X be an input dataflow for $\int$ and $\int_{\mathbb{T}}$. $\int$ will work on the projected dataflow $X_{\mathbb{T}_s}$ and $\int_{\mathbb{T}}$ will work on the projected dataflow $X_{\mathbb{T}}$. But $\tilde{\mathcal{F}}_{\mathbb{T}}$ and $\tilde{Q}_{\mathbb{T}}$ are defined to simulate the following behavior during their execution: they project the dataflow $X_{\mathbb{T}}$ on the time scale $\mathbb{T}_s$, then compute the transitions for the system $\int$ and finally project the output dataflow of time scale $\mathbb{T}_s$ on the finer time scale $\mathbb{T}$. Thus, $\int_{\mathbb{T}}$ will in fact compute $F_{\int_{\mathbb{T}}}(X) = (F_{\int}((X_{\mathbb{T}})_{\mathbb{T}_s}))_{\mathbb{T}}$, which by Propositions 4 is in fact $F_{\int}(X_{\mathbb{T}_s})_{\mathbb{T}}$. But as $\mathbb{T}_s \subseteq \mathbb{T}$, by Proposition 3, $F_{\int}(X_{\mathbb{T}_s})_{\mathbb{T}} \sim F_{\int}(X_{\mathbb{T}_s})$, which leads us to the desired result since $F_{\int}(X_{\mathbb{T}_s}) = F_{\int}(X)$.

The proof for the equivalence of the state dataflows (projected on the initial ϵ-alphabet of states S) is straightforward as S is associated with a persistent behavior. □

[34] x' corresponds to the data waiting on the input.
[35] $(\epsilon, \epsilon, \epsilon)$ is considered as the blank symbol ϵ.

5.1.2. Product

We now define the product of transfer functions and of systems, and show that they are mutually consistent. We first define the associative product $\otimes$ of datasets.

Definition 5.4 (Product of datasets). *Let $\mathcal{D}_1 = (D_1, (r_1, w_1))$ and $\mathcal{D}_2 = (D_2, (r_2, w_2))$ be two datasets. $\mathcal{D}_1 \otimes \mathcal{D}_2 = (D, (r, w))$ is a new dataset called **product of** $\mathcal{D}_1$ **and** $\mathcal{D}_2$ and defined by* [36]

- $D = D_1 \times D_2$ [37]
- $r((d_1, d_2)) = (r_1(d_1), r_2(d_2))$
- $w((d_1, d_2), (d'_1, d'_2)) = (w(d_1, d'_1), w(d_2, d'_2))$

The associative product of datasets allows to define an associative product of dataflows.

Definition 5.5 (Product of dataflows). *Let X be a dataflow on $(\mathcal{D}_X, \mathbb{T}_X)$ and Y be a dataflow on $(\mathcal{D}_Y, \mathbb{T}_Y)$. The **product** $X \otimes Y$ **of** X **and** Y is the dataflow on $(\mathcal{D}_X \otimes \mathcal{D}_Y, \mathbb{T}_X \cup \mathbb{T}_Y)$ defined by :*

$$\forall t \in \mathbb{T}_X \cup \mathbb{T}_Y,\ X \otimes Y(t) = (X_{\mathbb{T}_X \cup \mathbb{T}_Y}(t), Y_{\mathbb{T}_X \cup \mathbb{T}_Y}(t))$$

We define the projection of a dataflow on a dataset, allowing to consider only a part of the aggregated datasets of the dataflow.

Definition 5.6 (Projection of a dataflow on a dataset). *Let $\mathcal{D} = \mathcal{D}_1 \otimes \mathcal{D}_2$ be a dataset. Let $X \in \mathcal{D}^T$ be a dataflow of time scale $\mathbb{T}_X$. The **projection of** X **on** $\mathcal{D}_i$ $(i = 1, 2)$ is the dataflow $X_{\mathcal{D}_i}$ on $(\mathcal{D}_i, \mathbb{T}_X)$ defined by :*

$$\forall t \in \mathbb{T}_X,\ X_{\mathcal{D}_i}(t) = d_i \text{ where } X(t) = (d_1, d_2) \in D_1 \otimes D_2.$$

We can now define the product of transfer functions.

Definition 5.7 (Product of transfer functions). *Let $F_1 : {Input_1}^T \to {Output_1}^{\mathbb{T}_1}$ and $F_2 : {Input_2}^T \to {Output_2}^{\mathbb{T}_2}$ be two transfer functions. The **product of** F_1 **and** F_2 is the function $F_1 \otimes F_2 : (Input_1 \otimes Input_2)^T \to (Output_1 \otimes Output_2)^{\mathbb{T}_1 \cup \mathbb{T}_2}$ defined by:*

$$\forall X \in (Input_1 \otimes Input_2)^T,\ F_1 \otimes F_2(X) = F_1(X_{Input_1}) \otimes F_2(X_{Input_2})$$

This product defines a transfer function and is associative:

Proposition 5 (Closure and associativity for the product of transfer functions). *$F_1 \otimes F_2$ is a transfer function and $\otimes$ on transfer functions is associative.*

Proof. Let $\mathcal{D} = \mathcal{D}_1 \otimes \cdots \otimes \mathcal{D}_n$ be a dataset. Let $X, Y \in \mathcal{D}^T$ be two dataflows. Then: $\forall t \in T$: $(X \sim_t Y) \Leftrightarrow (\forall i,\ X_{\mathcal{D}_i} \sim_t Y_{\mathcal{D}_i})$. The proof can be easily obtained from this property. □

We finally define the product of n systems (sharing the same time scale) as the new system resulting from the aggregation of those systems (called "subsystems" of the new system)[38].

[36] It is easy to show that the new reading and writing functions comply with the axioms of a data behavior.

[37] (ϵ, ϵ) is considered as the blank symbol ϵ.

[38] Defining the product of n systems (and not just 2) is to give a semantics to the notion of subsystem.

Definition 5.8 (Product of systems). *Let* $(\int_i)_i = (\mathbb{T}_s, Input_i, Output_i, S_i, q_{0i}, \mathcal{F}_i, Q_i)_i$ *be n systems of time scale* $\mathbb{T}_s$*. The* ***product*** $\int_1 \otimes \cdots \otimes \int_n$ *is the system* $(\mathbb{T}_s, Input, Output, S, q_0, \mathcal{F}, Q)$ *where:*

- $Input = Input_1 \otimes \cdots \otimes Input_n$ *and* $Output = Output_1 \otimes \cdots \otimes Output_n$
- $S = S_1 \times \cdots \times S_n$ *and* $q_0 = (q_{01}, \ldots, q_{0n})$
- $\mathcal{F}((x_1, \ldots, x_n), (q_1, \ldots, q_n), t) = (\mathcal{F}_1(x_1, q_1, t), \ldots, \mathcal{F}_n(x_1, q_1, t))$
- $Q((x_1, \ldots, x_n), (q_1, \ldots, q_n), t) = (Q_1(x_1, q_1, t), \ldots, Q_n(x_1, q_1, t))$

The product can be generalized to systems with different time scales with the extension.

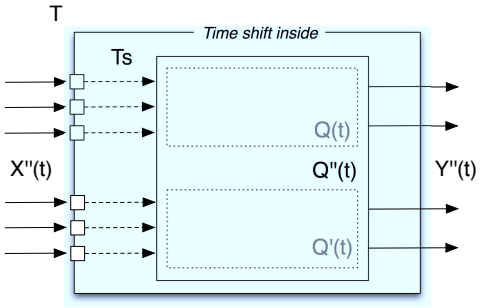


Figure 4: Product of systems

Theorem 3 (Consistency of the product of systems). *The transfer function of the product of n systems* $(\int_i)$ *is equivalent to the product of their transfer functions:*

$$F_{\int_1 \otimes \cdots \otimes \int_n} \sim F_{\int_1} \otimes \cdots \otimes F_{\int_n}$$

Proof . If the *n* systems share the same time scale, the proof is straightforward by definition of the product of systems (and the equivalence is in fact an equality). If not: we consider the extension of the *n* systems to the union of their time scales. By Theorem 2, the corresponding transfer functions are equivalent, and finally the equivalence stated in this Theorem is straightforward. □

5.1.3. Feedback

The feedback consists in defining a new system by connecting one of the output to one of the input of an existing system, sharing the same dataset[39]. However, it is not always possible to feedback an output on an input of same dataset: to define recursively the feedback of a system and express it as a new system with transition functions, it is necessary to establish the non-instantaneous influence of the input on the output concerned.

We first introduce broader and natural definitions of the feedback on transfer functions as fixed point of dataflows, and show later that it captures the feedback on systems.

[39] Sharing only the same ϵ-alphabet is not enough since having different data behaviors would make different resulting feedbacked systems according to the extension considered for the initial system.

Definition 5.9 (Feedback of a transfer function). *Let F be a transfer function of time scale $\mathbb{T}$ on signature $(D \otimes A, D \otimes B)$. F is feedbackable on D if, and only if: $\forall X \in A^T,\ \exists!\ Y_X \in D^{\mathbb{T}_s},\ F(Y_X \otimes X)_D = Y_X$. In this case, the **feedback of** F **on** D is the new transfer function $fb_{(F,D)}$ of time scale $\mathbb{T}_s$ on signature (A, B) defined by:*

$$\forall X \in A^T,\ fb_{(F,D)}(X) = F(Y_X \otimes X)_B$$

Proposition 6 (Equivalence of feedback on a finer time scale). *Let F be a transfer function and $F_{\mathbb{T}}$ be an extension of F to a finer time scale $\mathbb{T}$. Then, $fb_{(F,D)}$ exists if, and only if $fb_{(F_{\mathbb{T}},D)}$ exists, and in this case we have:*

$$fb_{(F,D)} \sim fb_{(F_{\mathbb{T}},D)}$$

Proof. As $F \sim F_{\mathbb{T}}$ and as the feedbacked input and output share the same data behaviors, Y_X for F will work for $F_{\mathbb{T}}$ by considering $(Y_X)_{\mathbb{T}}$, and conversely. □

We now define the feedback of a system by induction, so that it is a constructive definition.

Definition 5.10 (Feedback of a system). *Let $\int = (\mathbb{T}_s, (D \times In, \mathcal{I}), (D \times Out, \mathcal{O}), S, q_0, \mathcal{F}, \mathcal{Q})$ be a system such that there is no instantaneous influence of dataset D from the input to the output, i.e. $\forall t \in \mathbb{T}_s, \forall x \in In,\ \forall d \in D,\ \mathcal{F}((d, x), q, t)_D = \mathcal{F}((\epsilon, x), q, t)_D$. The **feedback of** D **in** $\int$ is the system $\int_{FB(D)} = (\mathbb{T}_s, (In, \mathcal{I}'), (Out, \mathcal{O}'), S, q_0, \mathcal{F}', \mathcal{Q}')$ with:*

- *$\mathcal{I}'$ is the restriction of $\mathcal{I}$ to In, and $\mathcal{O}'$ is the restriction of $\mathcal{O}$ to Out*
- $\mathcal{F}'(x \in In, q \in S, t) = \mathcal{F}((d_{x,q,t}, x), q, t)_{Out}$
- $\mathcal{Q}'(x \in In, q \in S, t) = \mathcal{Q}((d_{x,q,t}, x), q, t)$

where $d_{x,q,t}$ stands for $\mathcal{F}((\epsilon, x), q, t)_D$.

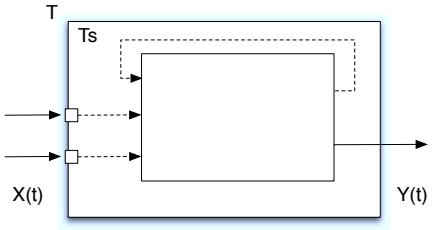


Figure 5: Feedback of a system

A good practice (well-spread in Systems Engineering) when modeling real systems is to always feedback a system with an interface (to model properties of the link).

Theorem 4 (Consistency of the feedback on systems). *The transfer function of the feedback of a system (when it exists) is the feedback of the transfer function of this system:*

$$F_{\int_{FB(D)}} = fb_{(F_{\int},D)}$$

Proof. We easily show by induction that the feedbacked dataflow constructed in the definition of a feedbacked system is a fixed point for the initial transfer function. □

5.2. Abstraction & concretization

The abstraction allows to define from a system a more abstract system, so that it can be integrated in more global ones. Abstraction allows to consider the right systemic level to describe a system, according to modeling needs, and is thus a fundamental tool to deal with the complexity of systems by hiding unnecessary low-level details related to the behavior of the system. It helps people to better understand a system and makes easier the formal analysis by working on abstraction of systems (see [14] for abstract interpretation which is a well-known example of abstraction).

The abstraction can be understood as a zoom out from the point of view of datasets (considering higher level datas for inputs, outputs and states, and eventually merging different dataflows), time (considering intervals of time instead of moments) and overall behavior. For instance, a computer may be considered as an electronic device with electrical signals every microsecond. However, we generally abstract this electronic device into a more abstract device able to process complex data as emails, with a time step being typically the hundredth of second (this simplified example will be modeled below).

The abstraction of a dataflow consists in defining a new dataflow on a more abstract dataset and on a more abstract time scale (typically with a larger step).

Definition 5.11 (Abstraction/concretization of dataflows). *An **abstraction** of dataflows is a surjective function $A : {D_c}^{\mathbb{T}_c} \to {D_a}^{\mathbb{T}_a}$ which is causal:*

$$\forall X, Y \in {D_c}^{\mathbb{T}_c},\ \forall t \in T,\ (X \sim_t Y) \Rightarrow (A(X) \sim_t A(Y))$$

*The associated **concretization** is the function $C : {D_a}^{\mathbb{T}_a} \to \mathcal{P}({D_c}^{\mathbb{T}_c})$ defined by $C(X) = A^{-1}(\{X\})$.*

We remark that an abstraction/concretization of dataflows is in fact a partition of the concrete dataflows whose elements are indexed by the abstract dataflows.

Example 13. We can take the example of a computer whose LAN connection is described by an input dataflow of bits on a regular time scale of step 10^{-6} sec, i.e. $D_c = \{0, 1, \epsilon\}$ and $\mathbb{T}_c = \tau\mathbb{N}$ with $\tau = 0.001$. We can abstract this dataflow to an abstract dataflow on $D_a = \{email, file, picture, video, html, \epsilon\}$ on time scale $\mathbb{T}_a = \tau'\mathbb{N}$ with $\tau' = 0.01$. ◊

The abstraction of a transfer function is a new transfer function working on abstract dataflows, with nondeterministic behaviors modeled by events dataflows (explained below in Example 14).

Definition 5.12 (Abstraction of a transfer function). *Let $F : Input^T \to Output^{\mathbb{T}_s}$ be a transfer function. Let $A_i : Input^{\mathbb{T}_s} \to {Input_a}^{\mathbb{T}_a}$ be an abstraction for input dataflows and $A_o : Output^{\mathbb{T}_s} \to {Output_a}^{\mathbb{T}_a}$ an abstraction for output dataflows. The **abstraction of** F for input and output abstractions (A_i, A_o) with events $\mathcal{E}$ is the new transfer function*

$$F_a : (Input_a \otimes \mathcal{E})^T \to {Output_a}^{\mathbb{T}_a}$$

defined by:

$$\forall X \in Input^T,\ \exists E \in \mathcal{E}^{\mathbb{T}_a},\ F_a(A_i(X_{\mathbb{T}_s}) \otimes E) = A_o(F(X))$$

Thus, the following diagram commutes (we dismiss events here):

$$\begin{array}{ccc} Input^{\mathbb{T}_s} & \xrightarrow{F} & Output^{\mathbb{T}_s} \\ {\scriptstyle A_i}\downarrow & & \downarrow{\scriptstyle A_o} \\ Input_a{}^{\mathbb{T}_a} & \xrightarrow[F_a]{} & Output_a{}^{\mathbb{T}_a} \end{array}$$

We now define the abstraction of a system.

Definition 5.13 (Abstraction of a system). *Let $\int = (\mathbb{T}_s, Input, Output, S, q_0, \mathcal{F}, Q)$ be a system. $\int' = (\mathbb{T}_a, Input_a \otimes \mathcal{E}, Output_a, S_a, q_{a0}, \mathcal{F}_a, Q_a)$ is an abstraction of $\int$ for input and output abstractions (A_i, A_o) if, and only if: $\exists A_q : S^{\mathbb{T}_s} \to S_a{}^{\mathbb{T}_a}$, for all execution (X, Q, Y) of $\int$, $\exists E \in \mathcal{E}^{\mathbb{T}_a}$, $(A_i(X_{\mathbb{T}_s}) \otimes E, A_q(Q), A_o(Y))$ is an execution of $\int'$.*
Conversely, $\int'$ is a concretization of the system $\int$.

Indeed, an abstraction consists in abstracting inputs, states and outputs dataflows in the execution of a system, and to define on these abstract dataflow a new system that will have abstract behaviors corresponding to the initial behaviors of the initial system. A good abstraction will be based on dataflows abstraction which will define consistent transitions in the abstract system for states and outputs. However, nondeterministic behaviors (modeled by events dataflow E) will generally appear in the abstract system. It is a consequence of regrouping states and input/output data in more abstract ϵ-alphabets, making it impossible to express the abstract behaviors as deterministic transitions on those ϵ-alphabets (for instance, one abstract data may correspond to several concrete data sometimes resulting in several behaviors of the concrete system, and the same may occur for the states). The abstraction of a deterministic system may thus result in nondeterministic behaviors, what does not mean that the real system modeled is nondeterministic.

Example 14. [Nondeterministic behaviors of abstraction of systems] We consider a glass whose state is described by an integer between 0 and 100 modeling the solidity of the glass (0 means broken). This glass can receive physical forces which lower its solidity till it is broken. At this level, the glass is described as a deterministic system. If we consider an abstraction of this model, we may consider the glass has being broken or not (two states) and receiving a shock (i.e. a sequence of physical forces) or nothing. When the glass, not broken yet, receives a shock, it will sometimes become broken, and sometimes remain not broken, depending of the previously received shocks. Therefore, at this level of abstraction, the glass has nondeterministic behaviors (since a shock may break it, with parameters that cannot be explained at this abstraction level). ◊

Theorem 5 (Consistency of the abstraction of a system). *The transfer function of the abstraction of a system is the abstraction of the transfer function of this system.*

Proof. The proof of this Theorem is straightforward regarding the definition of the abstraction of a system, which is defined as abstracting the transfer function of the initial system. □

5.3. Systems integration

The *integration* of systems in our framework consists in composing together a finite set of systems, with product (P) and feedback (F), then applying the abstraction (A) to describe the

resulting system at a more abstract level, and repeating those steps recursively till reaching the target overall system. We believe that the recursive *integration* of real systems (as done in Systems Engineering) can be modelled consistently as the corresponding integration of systems in our framework, using only P/F/A. We thus introduce a modeling postulate:

Postulate 1 (Real integration can be modeled with P/F/A). *Any real system $\int^r$ resulting from the "real" integration of elementary real systems ($\int_i^r$) can be consistently modeled as a system $\int$ resulting from recursive applications of operators P/F/A on the elementary systems ($\int_i$) (modeling the elementary real systems ($\int_i^r$)).*

One can remark that we only provided operators to integrate systems together. In reality, systems design involves mixing both bottom-up and top-down approaches. However, the same operators still hold, as the top-down approach can be interpretated as finding the right subsystems that, integrated together, are equivalent to the higher level system.

6. Conclusion

We have introduced a minimalist and unified semantics for heterogeneous integrated systems. This semantics allows us to capture two very important properties of complex systems: *heterogeneity* (being able to deal with various types of systems through rich time & data) and *recursive integration* (taking into account the integrative dimension of complex systems that are build recursively with multiple levels of components).

This work is the theoretical part of a broader project aiming at building an applied science for systems design, extending the models & methods existing for software design. Within the last two years, we have applied our framework to many real industrial cases from various industries (aeronautics, defence, banking, nuclear engineering, automotive) to assess the generality and the effectiveness of our approach. We will publish in further papers architecting methods derivated from this theoretical work to be applied to real life situations, with associated concrete industrial experimentations.

On the semantics itself, we have identified several topics of importance for our future work:

- our semantics can be presented in a more abstract way, in the scope of category theory using a coalgebraic approach (this work will be published very soon)
- in the present work, systems have been defined on static time scales only, regardless of events occuring during the system's life. It might be meaningful to extend this definition of systems to dynamic time scales constructed during the execution of the system
- the most complicated integration operator, i.e. *abstraction*, should be refined by different operators performing specialized kind of abstractions on systems, consistently with the reality of the specialized and meaningful abstractions encountered in Systems Engineering. Another associated improvement to our model would be to define a nondeterministic model of systems, which is necessary to define more specific abstraction operators
- finally, these models are intended to help designing systems. Therefore, we are willing to provide a formal framework to describe a design process using our semantics, providing a formalization of design approaches mixing top-down and bottom-up approaches to explore the recursive structure of integrated systems being designed.

References

[1] E. Alesken, R. Belcher, Systems Engineering, Prentice Hall, 1992.

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

[3] R. Alur, C. Courcoubetis, T. Henzinger, P.H. Ho, X. Nicollin, A. Olivero, J. Sifakis, S. Yovine, The algorithmic analysis of hybrid systems, in: International Conference on Analysis and Optimization of Systems—Discrete-Event Systems, Lecture Notes in Control and Information Sciences, Springer-Verlag, 1994, pp. 331–351.

[4] J. Bacon, J.V.D. Linden, Concurrent Systems: An Integrated Approach to Operating Systems, Distributed Systems and Database, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[5] G. Berry, The foundations of Esterel, MIT Press, 2000.

[6] B.S. Blanchard, W.J. Fabrycky, Systems engineering and analysis, Prentice Hall, 1998.

[7] S. Bliudze, D. Krob, Modelling of complex systems: Systems as dataow machines, Fundamenta Informaticae 91 (2009) 1–24.

[8] I.S. Board, IEEE Standard VHDL Language Reference Manual (IEEE Std. 1076-1993), IEEE, 1994.

[9] O. Bournez, M.L. Campagnolo, New computational paradigms. changing conceptions of what is computable, New Computational Paradigms. Changing Conceptions of What is Computable, Springer-Verlag, 2008, pp. 383–423.

[10] M. Broy, G. Stefănescu, The algebra of stream processing functions, Theor. Comput. Sci. 258 (2001) 99–129.

[11] M. Broy, K. Stølen, Specification and development of interactive systems: focus on streams, interfaces, and refinement, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.

[12] D. Cha, J. Rosenberg, C. Dym, Fundamentals of Modeling and Analysing Engineering Systems, Cambridge University Press, 2000.

[13] A.H. Clifford, G.B. Preston, The Algebraic Theory of Semigroups, Math. Surveys 7, Amer. Math. Soc., R.I., 1961.

[14] P. Cousot, R. Cousot, Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints, in: Conference Record of the Fourth Annual ACM Symposium on Principles of Programming Languages.

[15] F. Diener, G. Reeb, Analyse Non Standard, Hermann, 1989.

[16] N. Halbwachs, P. Caspi, P. Raymond, D. Pilaud, The synchronous data flow programming language lustre, in: Proceedings of the IEEE, volume 79.

[17] J. Harthong, éléments pour une théorie du continu, Astérisque 109/110 (1983) 235–244.

[18] J. Harthong, une théorie du continu, in: H. Barreau, J. Harthong (Eds.), La mathématique non standard, Éditions du CNRS, 1989, pp. 307–329.

[19] G. Kahn, The semantics of a simple language for parallel programming, in: J.L. Rosenfeld (Ed.), Information processing, North Holland, Amsterdam, Stockholm, Sweden, 1974, pp. 471–475.

[20] P. Kosiuczenko, M. Wirsing, Timed rewriting logic with an application to object-based specification, Science of Computer Programming 28 (1997) 225–246.

[21] D. Krob, Modelling of complex software systems: A reasoned overview, in: E. Najm, J. Pradat-Peyre, V. Donzeau-Gouge (Eds.), Formal Techniques for Networked and Distributed Systems - FORTE 2006, volume 4229 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2006, pp. 1–22.

[22] J. Lygeros, Lecture notes on hybrid systems, in: Notes for an ENSIETA Workshop.

[23] M.W. Maier, E. Rechtin, The art of system architecturing, CRC Press, 2002.

[24] P. Marwedel, Embedded System Design, Kluwer, 2003.

[25] G.H. Mealy, A method for synthesizing sequential circuits, Bell System Technical Journal 34 (1955).

[26] E. Nelson, Internal set theory: a new approach to nonstandard analysis, Bulletin of the American Mathematical Society 83 (1977) 1165–1198.

[27] A. Rabinovitch, Automata over continuous time, Theor. Comput. Sci. 300 (2003) 331–363.

[28] F. Robert, Les systèmes dynamiques discrets, Mathématiques et Applications 19 (1994).

[29] A. Robinson, Non-standard analysis, American Elsevier, 2nd. ed. edition, 1974.

[30] A.P. Sage, J.E. Amstrong, Introduction to system engineering, John Wiley, 2000.

[31] Y. Sergeyev, A new applied approach for executing computations with infinite and infinitesimal quantities, Informatica 19 (2008) 567–596.

[32] E. Sontag, Mathematical Control Theory: Deterministic Finite Dimensional Systems, volume 6 of *Textbooks in Applied Mathematics*, Springer-Verlag, 1998.

[33] B. Trakhtenbrot, Understanding basic automata theory in the continuous time setting, Fundam. Inform. 62 (2003).

[34] J. Zaytoun, Systèmes dynamiques hybrides, Hermes, 2001.

[35] B.P. Zeigler, H. Praehofer, K.T. Gon, Theory of Modeling and Simulation — Integrating Discrete Event and Continuous Complex Dynamic Systems, Academic Press, 2000.