A minimalist formal framework for systems design

B. Golden

LIX, École Polytechnique, 91128 Palaiseau Cedex, France

Y. Hourdel

LIX, École Polytechnique, 91128 Palaiseau Cedex, France

Abstract

We introduce a formal framework to deal with heterogeneous integrated systems during the systems architecting process. Two main problems are addressed in this paper: how to deal with the underspecification of systems during their design process, and how to formalize the structure of a sytem. We consider a minimalist design process, consisting of requirements analysis and systemic recursion.

Keywords: Systems modeling, Systems architecture, Systems Engineering, Architecture framework

Introduction

Previous studies, such as [1] or [2], introduce systems architecture as a field of good practices more than a formal science with tangible notions. On the contrary, some formal studies introduce new frameworks like [3] or even new concepts such as hybrid systems (first in [4] and [5], then in [6] and others) as tools to handle such large-scale designs. This particularity of systems architecture, being at a boundary between formal and real worlds, has lead us to a point where architects from different fields of study still lack a common formal language to share designs or prove properties of systems.

The start point of this work is a model described in [7]: a unified model for heterogeneous integrated systems and their integration. This attempt to create a model that could be used in many different fields has proved its value, so our work goes on with this aim by providing a formal framework to extend the formal approach to the design process of such systems.

To this aim, two fundamental questions must be addressed:

- how to deal with the underspecification of systems during the design process?
- how to formalize the internal structure of a sytem?

Most of the time, when an architect designs a new product, he has in mind the information that is missing from the model. The problem is that if he needs to give this model to another person, he is unable to give all this information. This is why the second question is also very important: by using an appropriate formalism, one can be sure to store all the design knowledge in transmissible elements.

We focus in this paper on a minimalist systems architecting process in which the only possible actions are:

- breaking a system into a set of smaller systems (and conversely composing together a set of systems)
- "concretizing" a system into a finer grain system (and conversely "abstracting" a system)
- expressing requirements on a system (and checking if a system verifies such requirements).

Such a design process encompasses the essential aspects of systems architecting, i.e. requirements analysis and systemic recursion.

1. Preliminary definitions

In this section, we recall the definitions introduced in [7] to formalize the notion of *system*, a timed extension of Mealy machines to model heterogeneous integrated systems and their integration.

1.1. Time

Time is an underlying, yet very important, point of our formal approach. Indeed, real-life systems are naturally described according to various types of "times". As a result, we need to deal uniformaly with both continuous and discrete times. While this problem has been shown hard by [8], some solutions have been found in studies like [6] to introduce formal models for mixed discrete-continuous systems. We give here a set of definitions to handle such systems.

Email addresses: boris.golden@polytechnique.edu (B. Golden), yann.hourdel@polytechnique.edu (Y. Hourdel)

Informally, as very well expressed in [7], time is "a linear quantity composed of ordered moments, pairs of which define durations".

Definition 1.1 (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)

Once we agree upon this definition of time, we can build time scales as sets of moments that are important for a certain description of a system.

Definition 1.2 (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 $pred^{\mathbb{T}}(t)$
- the principle of induction[1] is true on $\mathbb{T}$.

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

1.2. Dataflows

Together with this unified definition of time, we need a definition of data that allows to handle the heterogeneity of data among real-life systems. We rely on the previous definitions to describe data carried by dataflows.

Definition 1.3 (ϵ-alphabet). A set D is an **ϵ-alphabet** if $\epsilon \in D$. For any set B, we can define an ϵ-alphabet by $\overline{B} = B \cup \{\epsilon\}$.

Definition 1.4 (System dataset). A **system dataset**, or **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[2]:
  - $r(\epsilon) = \epsilon$ $(R1)$
  - $r\big(r(d)\big) = r(d)$ $(R2)$
  - $r\big(w(d, d')\big) = r(d')$ $(R3)$
  - $w\big(r(d'), d\big) = d$ $(W1)$
  - $w\big(w(d, d'), r(d')\big) = w(d, d')$ $(W2)$

Definition 1.5 (Dataflow). Let $\mathbb{T}$ be a time scale. A **dataflow** over $(\mathcal{D}, \mathbb{T})$ is a mapping $X : \mathbb{T} \to D$.

Definition 1.6 (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 $\mathcal{D}^T = \bigcup_{\mathbb{T} \in Ts(T)} \mathcal{D}^{\mathbb{T}}$.

1.3. Systems and integration operators

Given the previous definitions, we are now able to give a mathematical definition of systems. Informally, our definition is very similar to timed Mealy machines with two important differences: the set of states may be infinite and the transfer function transforms dataflows. The key point is to see those systems as black boxes that just behave the way they are supposed to.

Definition 1.7 (System). A **system** is a 7-tuple $\int = (\mathbb{T}, X, Y, Q, q_0, \mathcal{F}, \delta)$ where:

- $\mathbb{T}$ is a time scale,
- X, Y are *input* and *output* datasets,
- Q is a nonempty ϵ-alphabet [3] *of states*,
- q_0 is an element of Q, called *initial state*,
- $\mathcal{F} : X \times Q \times \mathbb{T} \to Y$ describes a *functional behavior*,
- $\delta : X \times Q \times \mathbb{T} \to Q$ describes a *state behavior*.

Figure 1 illustrates this definition.

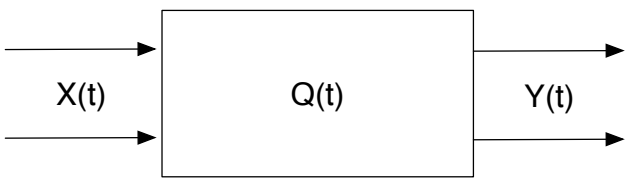

Figure 1: Illustration of a system

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

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

[3] Defining Q as an ϵ-alphabet (therefore containing ϵ) and not just as a set will make it possible to define a dataflow of states, which is convenient.

It is important to understand here that at each time instant of the time scale, the state of the system changes instantly and before $\mathcal{F}$ computes the resulting output. At $m^{\mathbb{T}_s}$, the beginning of the time scale, the state of the system is q_0. But as soon as the first input data arrives, at $succ^{\mathbb{T}}(m^{\mathbb{T}_s})$, the state of $\int$ changes so that the functional behaviour ignores q_0. Figure 2 illustrates this behaviour and we give the following formal definition for timed executions of systems.

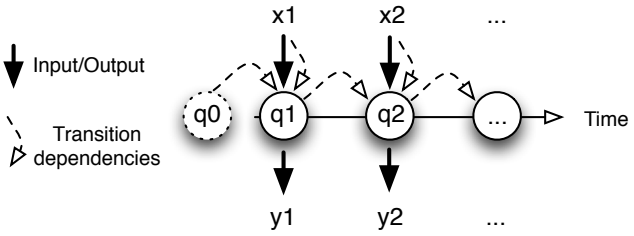

Figure 2: Transitions of a system throughout its time scale

Definition 1.8 (Execution of a system). Let $\int = (\mathbb{T}, X, Y, Q, q_0, \mathcal{F}, \delta)$ be a system. Let $In \in X^T$ be an input dataflow for $\int$ and $\tilde{In} = In_{\mathbb{T}}$. The **execution of $\int$ on the input dataflow** In is the 3-tuple (In, S, Out) where:

- $S \in Q^{\mathbb{T}}$ is recursively defined by:
  - $S(m^{\mathbb{T}}) = \delta\big(\tilde{In}(m^{\mathbb{T}}), q_0, m^{\mathbb{T}}\big)$
  - $\forall t \in \mathbb{T},\ S(t^+) = \delta\big(\tilde{In}(t^+), S(t), t^+\big)$ where $t^+ = succ^{\mathbb{T}}(t)$
- $Out \in Y^{\mathbb{T}}$ is defined by:
  - $Out(m^{\mathbb{T}}) = \mathcal{F}\big(\tilde{In}(m^{\mathbb{T}}), q_0, m^{\mathbb{T}}\big)$
  - $\forall t \in \mathbb{T},\ Out(t^+) = \mathcal{F}\big(\tilde{In}(t^+), S(t), t^+\big)$ where $t^+ = succ^{\mathbb{T}}(t)$

In, S and Out are respectively input, state and output dataflows.

Definition 1.9 (Product of systems on a time scale). Let $(\int^i)_i = (\mathbb{T}, X_i, Y_i, Q_i, q_{0i}, \mathcal{F}_i, \delta_i)_i$ be n systems of time scale $\mathbb{T}$. The **product** $\int^1 \otimes \cdots \otimes \int^n$ is the system $\big(\mathbb{T}, X, Y, Q, q_0, \mathcal{F}, \delta\big)$ where:

- $X = X_1 \otimes \cdots \otimes X_n$ and $Y = Y_1 \otimes \cdots \otimes Y_n$
- $Q = Q_1 \times \cdots \times Q_n$ and $q_0 = (q_{01}, \ldots, q_{0n}) = q_{01\ldots n}$
- $\mathcal{F}\big(x_{1\ldots n}, q_{1\ldots n}, t\big) = \big(\mathcal{F}_1(x_1, q_1, t), \ldots, \mathcal{F}_n(x_1, q_1, t)\big)$
- $\delta\big(x_{1\ldots n}, q_{1\ldots n}, t\big) = \big(\delta_1(x_1, q_1, t), \ldots, \delta_n(x_1, q_1, t)\big)$

Remark 1. This definition can be extended to systems that do not share a time scale, thanks to a technical operator introduced in [7]. This operator builds a timed-extension of a system, which is a system that has an equivalent input-output behaviour as the original system, but on a wider time scale. Figure 3 illustrates this idea.

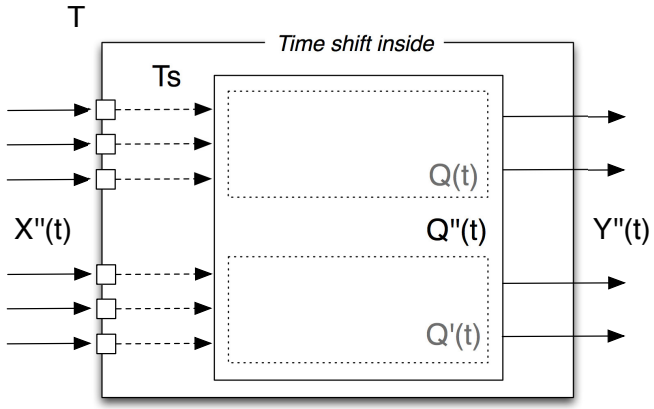

Figure 3: Product of systems

Definition 1.10 (Feedback of a system). Let $\int = \big(\mathbb{T}, (D \times In, \mathcal{I}), (D \times Out, \mathcal{O}), Q, q_0, \mathcal{F}, \delta\big)$ be a system such that there is no instantaneous influence of dataset D from the input to the output[4], i.e. $\forall t \in \mathbb{T}, \forall x \in In,\ \forall d \in D,\ \mathcal{F}\big((d, x), q, t\big)_D = \mathcal{F}\big((\epsilon, x), q, t\big)_D$. The **feedback of D in** $\int$ is the system $\int_{FB(D)} = \big(\mathbb{T}, (In, \mathcal{I}'), (Out, \mathcal{O}'), Q, q_0, \mathcal{F}', \delta'\big)$ where:

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

where $d_{x,q,t}$ stands for $\mathcal{F}\big((\epsilon, x), q, t\big)_D$.
Figure 4

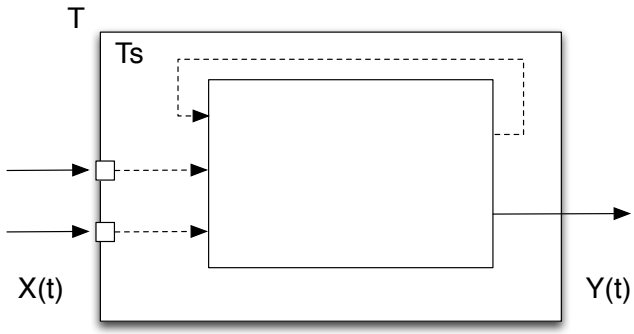

Figure 4: Feedback of a system

Definition 1.11 (Abstraction of a transfer function). Let $F : X^T \to Y^{\mathbb{T}}$ be a transfer function. Let $A_x : X^{\mathbb{T}} \to {Y_a}^{\mathbb{T}_a}$ be an abstraction for input dataflows

[4] As explained informally in [7], this condition makes it possible to define a unique feedback, i.e. without having to solve a fixed point equation that could lead to zero or multiple solutions

and $A_y : Y^{\mathbb{T}} \to {Y_a}^{\mathbb{T}_a}$ an abstraction for output dataflows. The **abstraction of** F for input and output abstractions (A_x, A_y) with events $\mathcal{E}$ is the new transfer function

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

defined by:

$$\forall x \in X^T,\ \exists e \in \mathcal{E}^{\mathbb{T}_a},\ F_a\big(A_x(x_{\mathbb{T}}) \otimes e\big) = A_y\big(F(x)\big)$$

Figure 5 illustrates this definition.

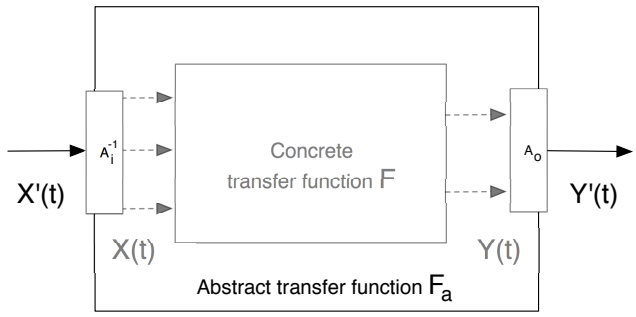

Figure 5: Abstraction of a transfer function

Definition 1.12 (Abstraction of a system). Let $\int = \big(\mathbb{T}, X, Y, Q, q_0, \mathcal{F}, \delta\big)$ be a system. $\int' = \big(\mathbb{T}_a, X_a \otimes \mathcal{E}, Y_a, Q_a, q_{a0}, \mathcal{F}_a, \delta_a\big)$ is an abstraction of $\int$ for input and output abstractions (A_x, A_y) if, and only if: $\exists A_q : Q^{\mathbb{T}} \to {Q_a}^{\mathbb{T}_a}$, for all execution (x, q, y) of $\int$, $\exists E \in \mathcal{E}^{\mathbb{T}_a}$, $\big(A_x(x_{\mathbb{T}}) \otimes E, A_q(q), A_y(y)\big)$ is an execution of $\int'$. Conversely, $\int'$ is a concretization of the system $\int$.

A system captures the behavior of a system that can be observed (functional and states behavior, called together *systemic behavior*). However, a system has two limitations:

- it requires to fully specify the functional and states behaviors
- it gives no information on what the system is composed of[5].

2. Systemic behaviour specification

Based on the previous definitions,

In this section, we introduce a formalization based on the previous definitions. The first step consists of giving a formal frame for describing expected behaviours of systems. One key point in this attempt is to give formal tools to study such constraints during the systems design process. Indeed, during this design process, systems are underspecified as their expected behaviour is built through the entire process and not known from the beginning. Therefore, we need formal ways to allow a certain set of behaviours, and to restrict this set while approaching the final design of the system.

Thus, we will need some sort of generic behaviour signatures for systems. These signatures will partially restrict behaviours so that each possible concretization of the system design, by complying with the signature, will have an acceptable behaviour.

Definition 2.1 (Systemic signature). A **systemic signature** is a 4-tuple $(X, Y, Q, \mathbb{T})$ where X, Y and Q are datasets (respectively called *input values*, *output values* and *states*) and $\mathbb{T}$ is a time scale.
Figure 6 illustrates this definition.

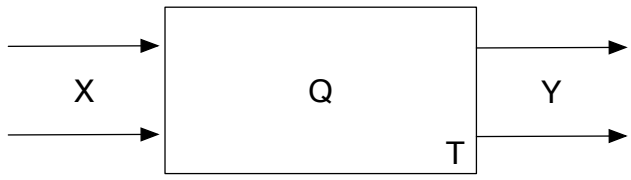

Figure 6: Illustration of a systemic signature

Remark 2. A system naturally induces a systemic signature.

A systemic signature gives us a language in which we can express behaviour constraints. These constraints, according to standard usage of systems architecture, are temporal properties expressed on executions of systems. Therefore, these constraints only use the alphabets of input values, output values and internal states of a system. A minimalist formalism for requirements can be found in [9].

In this work, we will use the following simplified definition of a behaviour constraint, or requirement, that is sufficient for our purpose.

Definition 2.2 (Requirement). Let $\Sigma = (X, Y, Q, \mathbb{T})$ be a systemic signature. A **requirement** on Σ, is a logical formula expressing properties on the behavior of any system of systemic signature Σ. The set of all requirements on this systemic signature is noted $Req(X, Y, Q, \mathbb{T})$ or $Req(\Sigma)$.

During the design process, most of the time we deal with objects that are not fully specified. This is especially true for systems: they are described through their systemic signature, together with a set of expected properties. We thus define a notion that captures such underspecified objects:

Definition 2.3 (Black box). A **black box** is a 5-uplet $(X, Y, Q, \mathbb{T}, r)$ where:

- $(X, Y, Q, \mathbb{T})$ is a systemic signature

[5] since integration operators do not model the structure of a system

- $r \in Req(X, Y, Q, \mathbb{T})$

We note $BB(X, Y, Q, \mathbb{T})$ the set of all black boxes of systemic signature $(X, Y, Q, \mathbb{T})$.
Figure 7 illustrates this definition.

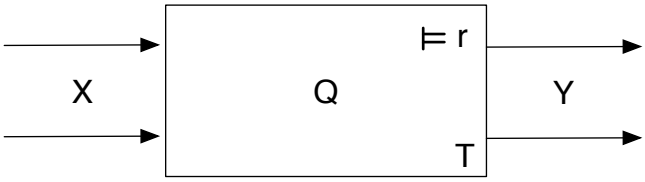

Figure 7: A black box

Remark 3. We only consider one requirement in our definition of a black box, which is equivalent to a finite set of requirements.

A black box induces a set of systems whose systemic signature matches the one from the black box, and that comply with the requirement of the black box:

Definition 2.4 (Realization of a black box). Let $B = (X, Y, Q, \mathbb{T}, r)$ be a black box. A **realization** of B is any system S of systemic signature $(X, Y, Q, \mathbb{T})$ such that $S \vDash r$. When such a system exists, B is said to be *realizable*.
Figure 8 illustrates this definition.

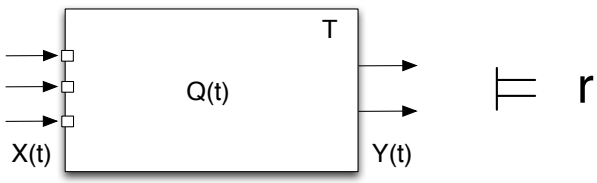

Figure 8: Illustration of the realization of a black box

Remark 4. One of the challenge of systems design, in this framework, is to be able to define only realizable black boxes at each level. In practice, this is an iterative process with trial & error. Note that, in real life, the requirement associated with a black box will constrain its behavior, but also express constraints related to cost, time, feasability & other business metrics.

A key property in systems architecture is the ability to change the granularity of description of a model through mechanisms of abstraction and concretization. Thus, we naturally extend our definition of such mechanisms from systems to black boxes:

Definition 2.5 (Concretization of a black box). Let $B_c \in BB(X_c, Y_c, Q_c, \mathbb{T}_c)$ (called *concrete black box*). Let $B_a \in BB(X_a, Y_a, Q_a, \mathbb{T}_a)$ (called *abstract black box*). Let $\alpha : (X_c, Y_c, Q_c, \mathbb{T}_c) \to (X_a, Y_a, Q_a, \mathbb{T}_a)$ be an abstraction mechanism (as defined in [7]).
We say that B_c **concretizes** B_a ***via*** α if and only if: for any system S_c that is a realization of B_c, $\alpha(S_c)$ is a realization of B_a[6].
Figure 9 illustrates this definition.

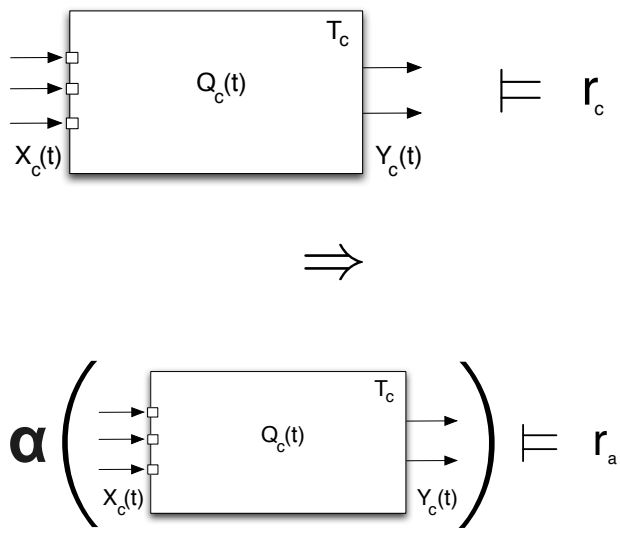

Figure 9: Concretization of a black box

We can now deal with underspecification:

- a systemic signature is the most underspecified object
- a black box is a systemic signature, with expected properties on the systemic behavior
- a system is the algorithmic specification of a systemic behavior.

We will now introduce a formalism to deal with the structure of systems.

3. Systems as recursive structures

The goal of this section is to deal with the structure of more or less specified objects through the design process.

We have defined integration operators for systems and transfer functions, but those operators don't allow storing the internal structure. We thus first introduce an object that formalizes a finite sequence of products and feedbacks on a finite set of systems:

[6] This (only) means that $\alpha(S_c)$ verifies the requirement of B_a. Still, is a very strong property and means that the requirement on the concrete system is "strong" enough to insure that the requirement of its abstraction will be verified

Definition 3.1 (Composition plan). Let $S_0, \ldots, S_{n-1}$ be n systems. A **composition plan** for $S_0, \ldots, S_{n-1}$ is any set $C \subset \{0, \ldots, n-1\}^2$ of couples such that:

- $\forall \big((a,b),(c,d)\big) \in C^2,\ \big[(a \neq c) \wedge (b \neq d)\big] \vee \big[(a = c) \wedge (b = d)\big]$
- $\forall\ (a,b) \in C$, the output Y_a of S_a and the input X_b of S_b have the same dataset.

More informally, C is a set of links between ouputs and inputs of $S_0, \ldots, S_{n-1}$ such that each input (resp. ouput) is linked to at most one output (resp. input). We thus write $C(S_0, \ldots, S_{n-1})$ for the system resulting from the composition of these n back systems according to C.

Remark 5. The definition of a composition plan for systems can easily be extended to black boxes using their systemic signatures.

Remark 6. This definition can easily be extended to systems with multiple inputs and ouputs.

A key element in systems architecture is the ability to refine a system by breaking it down into smaller subsystems:

Definition 3.2 (Refinement of a black box). Let $B \in BB(X, Y, Q, \mathbb{T})$. For all $i \in \{0, \ldots, n-1\}$, let $B_i \in BB(X_i, Y_i, Q_i, \mathbb{T})$. Let C be a composition plan for $B_0, \ldots, B_{n-1}$. $(B_0, \ldots, B_{n-1}, C)$ is a **refinement** of B iff the systemic signature of $C(B_0, \ldots, B_{n-1})$ is $(X, Y, Q, \mathbb{T})$.

We give a first formalization to the systemic recursion by combining a black box together with a refinement:

Definition 3.3 (View). A **view** is a couple $\big(B, (B_0, \ldots, B_{n-1}, C)\big)$:

- B is a black box
- $(B_0, \ldots, B_{n-1}, C)$ is a refinement of B.

Figure 10 illustrates this definition.

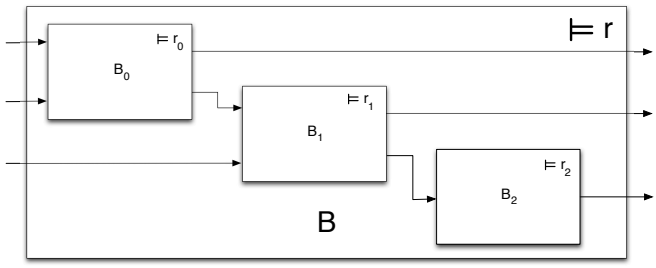

Figure 10: Illustration of a view

Remark 7. The time reference is unique in a view. Changes of time reference are only possible throug abstraction/concretization mechanisms.

A view is thus a formal object that models the ability to refine a black box through a set of interrelated black boxes. We then naturally extend the notion of concretization of a black box to views:

Definition 3.4 (Concretization of a black box by a view). Let $V = \big(B_c, \_\big)$ be a view. V is a **concretization** of a black box B_a *via* an abstraction α iff B_c concretizes B_a *via* α.

As for black boxes, the existence of systems realizing the back boxes of a view in a consistent way is key[7]:

Definition 3.5 (Realization of a view). Let $V = \big(B, (B_0, \ldots, B_{n-1}, C)\big)$ be a view. A **realization** of V is any realization $S_0, \ldots, S_{n-1}$ of $B_0, \ldots, B_{n-1}$ such that $C(S_0, \ldots, S_{n-1})$ is a realization of B. In this case, $C(S_0, \ldots, S_{n-1})$ is called the *composition* of $S_0, \ldots, S_{n-1}$ *according to* V, and V is said to be *realizable*.

However, a view only captures one level of systemic recursion. We thus introduce a new object capturing a finite number of systemic recursions and allowing the use of abstractions[8]:

Definition 3.6 (Multiscale view). A **multiscale view** W is a tree such that:

- every node of W is labeled with a view
- every edge e of W from a parent node $V_p = \big(\_, (B_0, \ldots, B_{n-1}, \_)\big)$ to a child node V_c is labeled with a couple (k, α) where:
  - $k \in \{0, \ldots, n-1\}$ is called the *index* of the edge e
  - α is an abstraction such that V_p concretizes B_k via α
- for a parent node $V_p = \big(\_, (B_0, \ldots, B_{n-1}, \_)\big)$, there is at most one edge of index $k \in \{0, \ldots, n-1\}$.

Figure 11 illustrates this definition.

Remark 8. For any node N from a multiscale view W, the subtree of W with root N is a multiscale view.

Thus, a multiscale view is a tree of views, such that a black box in a view is either concretized (by another multiscale view), or "free":

[7] It is very difficult in a design process to define such views because many interrelated elements must be consistent. That's a reason why the design process is iterative in practice. Note also that their is no "mechanistic" bottom-up implication: it is possible (and even likely!) to have realizations of the black boxes of a view whose composition is not a realization of the higher level black box.

[8] at each systemic level, all systems resulting from the composition of lower level subsystems are brought to a common level of abstraction through individual abstractions

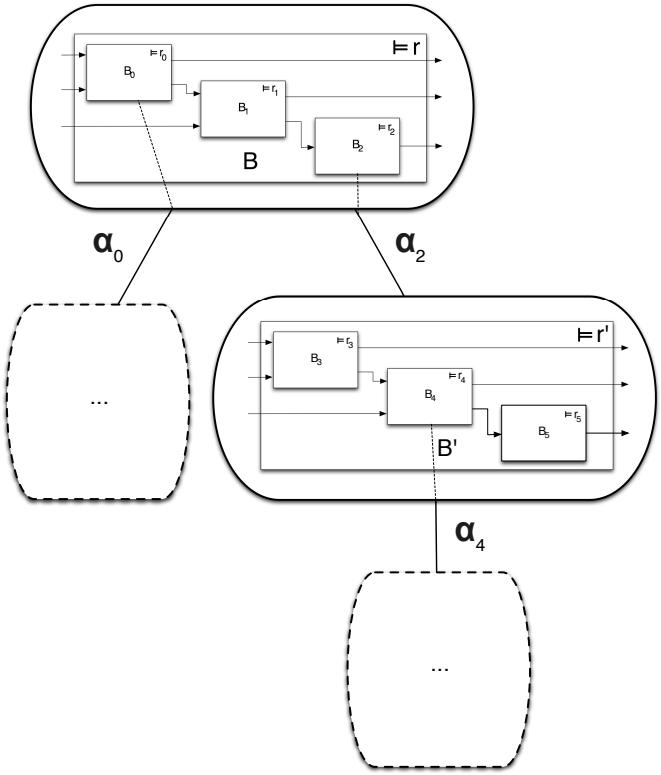

Figure 11: Illustration of a multiscale view

Definition 3.7 (Free box). Let W be a multiscale view. Let V be a view labeling a node of W. A **free box** of W is any black box B of V such that: B is not concretized by any child of V in W. We write $freebox(W)$ for the finite sequence of free boxes of W, enumerated in depth-first order.

Remark 9. For any view V labeling a leaf of a multiscale view W, all the black boxes of V are free boxes of W.

Definition 3.8 (Valuation). Let W be a multiscale view. Let V be a view labeling a node of W and let B be a black box of V. The **valuation** $v_W(B) \in \mathbb{N}$ of B in W is defined as follows:

- $v_W(B) = 1$ when B is a free box of W
- $v_W(B) = \big|freebox(W_B))\big|$ else, where W_B is the multiscale view concretizing B in W.

To define a system from a multiscale view, the free boxes need to be specified:

Definition 3.9 (Integration tree). Let W be a multiscale view and let $freebox(W) = B_0^f, \ldots, B_{n-1}^f$ be its free boxes, such that all free boxes of W are realizable. Let $S_0, \ldots, S_{n-1}$ be n systems respectively realizing $B_0^f, \ldots, B_{n-1}^f$. The **integration tree** of $(S_0, \ldots, S_{n-1})$ according to W, which we note $\mathcal{I}\big(W, (S_0, \ldots, S_{n-1})\big)$, is a tree (whose nodes are labelled with systems) recursively defined as follows:

let $V = \big(B, (B_0, \ldots, B_{p-1}, C)\big)$ be the label of the root node of W. $\forall i \in \{0, \ldots, p\}$, let $x_i = \sum_{j=0}^{i-1} v_W(B_j)$[9]. $\forall i \in \{0, \ldots, p-1\}$, we define M_i as:

- when $B_i \in freebox(W)$, M_i is a single node labeled with S_{x_i}
- when $B_i \notin freebox(W)$, we define $M_i' = \mathcal{I}\big(W_i, (S_{x_i}, \ldots, S_{x_{i+1}-1})\big)$[10], where W_i is the multiscale view concretizing B_i in W following α_i. M_i is a tree consisting in a root node labeled by $\alpha_i\big(root(M_i')\big)$ and with a unique child M_i'.

We then define $\mathcal{I}\big(W, (S_0, \ldots, S_{n-1})\big)$ as the tree composed of p subtrees $M_0, \ldots,$
M_{p-1} children of a root labelled by the composition of the systems labeling the roots of $M_0, \ldots, M_{p-1}$, i.e. $C\big(root(M_0), \ldots, root(M_{p-1})\big)$.

The integration tree is *consistent* iff the system labeling each node verifies the requirement of its associated black box.

We can now define the system resulting from the integration of a sequence of systems following a multiscale view:

Definition 3.10 (Integration). Let W be a multiscale view and let $freebox(W) = B_0^f, \ldots, B_{n-1}^f$ be its free boxes, such that all free boxes of W are realizable. Let $S_0, \ldots, S_{n-1}$ be n systems respectively realizing $B_0^f, \ldots, B_{n-1}^f$. The **integration according to** W of $S_0, \ldots, S_{n-1}$ is the system labeling the rood node of $\mathcal{I}\big(W, (S_0, \ldots, S_{n-1})\big)$. Such an integration is *consistent* iff the corresponding integration tree is consistent.

As for views, the existence of systems realizing a multiscale view in a consistent way is key:

Definition 3.11 (Realization of a multiscale view). Let W be a multiscale view. Let $S_0, \ldots, S_{n-1}$ be n systems. $(S_0, \ldots, S_{n-1})$ is a **realization** of W if it is a consistent integration according to W. In this case, W is said to be *realizable*.

We finally introduce a model of systems where the structure is described. A white system describes the structure of a system in terms of successive compositions and abstractions:

Definition 3.12 (White system). A **white system** is a tree where:

[9] as the free boxes are in depth-first order, x_i represents the index of the first free box of W which is a free box of B_i

[10] such tree is well-defined because: 1) the free boxes are enumerated in depth-first order 2) the recursion is applied to a strictly smaller multiscale view (it insures that the recursion will terminate).

- all leaves are labelled with a system
- nodes with an even depth are labelled with a couple (S, C), where S is a system and C is a composition plan
- nodes with an odd depth are labelled with a couple (S, α), where S is a system and α is an abstraction function
- for each even node (S, C) of children $(S_0, \_), \ldots, (S_{n-1}, \_)$; we have: $S = C(S_0, \ldots, S_{n-1})$
- for each odd node (S, α), its unique child $(S', \_)$ is such that: $S = \alpha(S')$.

Figure 12 illustrates this definition.

Remark 10. The integration tree of a sequence of systems according to a multiscale view naturally induces a white system.

We thus have introduced a formalism allowing to define underspecified systems (with black boxes), to define a recursive structure on them (through multiscale views), and finally to define fully specified systems with a recursive structure (through white systems). Altogether, they form a minimalist framework for systems architecture in our formalism.

Conclusion

The result of this work is a coherent set of formal notions that allow architects to describe a system through different points of view that can be proved coherent from one abstraction level to another. However, despite the fact it was one of our initial goals, these notions appear to be much too complex to be used as a common architecture language. Indeed, to prove some parts of it we had to put a couple of heavy mathematical tools forward. As a result, we get a set of formal notions that can be used but should be adapted to more practical tools.

This work tends to be a first contribution to mathematical formalization behind architecture models, so we hope it brings some missing notions to this field. We wish to provide, in future work, complete designs of real systems modeled using this set of notions.

References

[1] Perry D, Wolf A. Foundations for the study of software architectures. ACM SIGSOFT Software Engineering Notes 1992;17(4):40–52.

[2] Sangal N, Jordan E, Sinha V, Jackson D. Using dependency models to manage complex software architecture. In: Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. OOPSLA '05; New York, NY, USA: ACM. ISBN 1-59593-031-0; 2005, p. 167–76. URL: http://doi.acm.org/10.1145/1094811.1094824. doi:10.1145/1094811.1094824.

[3] Aiguier M, Boulanger F, Kanso B. A formal abstract framework for modelling and testing complex software systems. Theor Comput Sci 2012;455:66–97. URL: http://dx.doi.org/10.1016/j.tcs.2011.12.072. doi:10.1016/j.tcs.2011.12.072.

[4] Alur R, Courcoubetis C, Henzinger T, Ho P. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. Hybrid systems 1993;:209–29.

[5] Alur R, Courcoubetis C, Henzinger T, Ho PH, Nicollin X, Olivero A, et al. 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, p. 331–51.

[6] Henzinger TA. The theory of hybrid automata. In: Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science. LICS '96; Washington, DC, USA: IEEE Computer Society. ISBN 0-8186-7463-6; 1996, p. 278–. URL: http://dl.acm.org/citation.cfm?id=788018.788803.

[7] Golden B, Aiguier M, Krob D. Modeling of complex systems ii: A minimalist and unified semantics for heterogeneous integrated systems. Applied Mathematics and Computation 2012;218(16):8039–55. URL: http://dx.doi.org/10.1016/j.amc.2012.01.048.

[8] Bournez O, Campagnolo ML. A survey on continuous time computations. CoRR 2009;abs/0907.3117.

[9] Golden B. A unified formalism for complex systems architecture. Ph.D. thesis; cole Polytechnique; 2013.

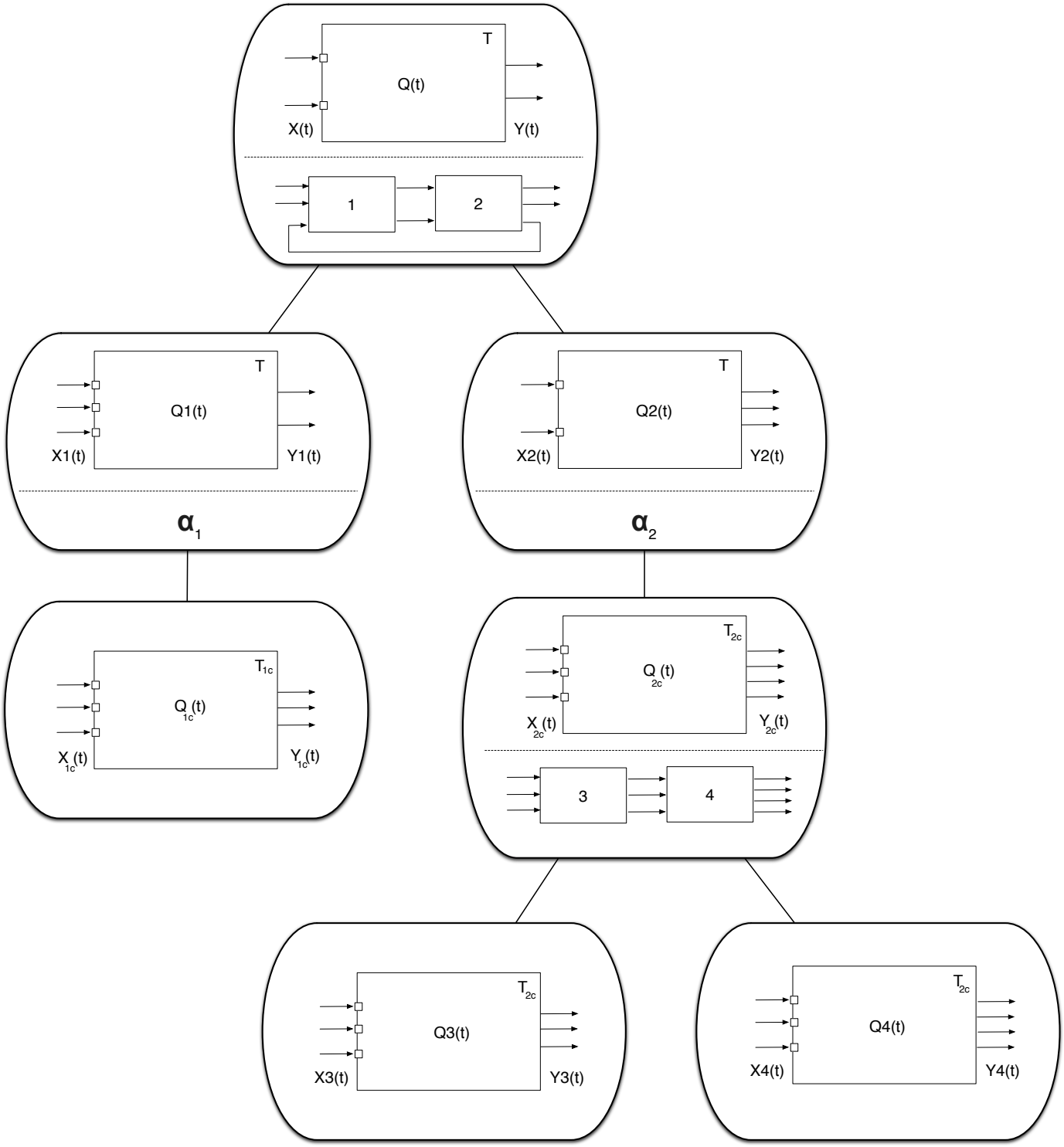

Figure 12: Illustration of a white system