Bridging Discrete and Continuous Time Models with Atoms

George Ungureanu
Department of Electronics
KTH Royal Institute of Technology
Stockholm, Sweden
ugeorge@kth.se

José E. G. de Medeiros
Electrical Engineering Department
University of Brasília
Brasília, DF, Brazil
j.edil@ene.unb.br

Ingo Sander
Department of Electronics
KTH Royal Institute of Technology
Stockholm, Sweden
ingo@kth.se

***Abstract*—Recent trends in replacing traditionally digital components with analog counterparts in order to overcome physical limitations have led to an increasing need for rigorous modeling and simulation of hybrid systems. Combining the two domains under the same set of semantics is not straightforward and often leads to chaotic and non-deterministic behavior due to the lack of a common understanding of aspects concerning *time*. We propose an algebra of primitive interactions between continuous and discrete aspects of systems which enables their description within two orthogonal layers of computation. We show its benefits from the perspective of modeling and simulation, through the example of an RC oscillator modeled in a formal framework implementing this algebra.**

I. Introduction

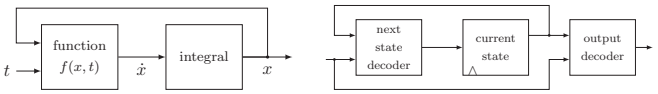
Abstraction is probably the most important tool available for science and engineering. While scientists employ it to build better understanding about the physical world, engineers often use it to describe desired artifacts from a design flow [1]. The abstraction process is thus distinct between the two activities. While in science we often choose different abstractions from established paradigms to analyze different aspects of a system, in engineering we have more freedom to actually design new abstractions to be able to express novel ideas even though sometimes we are not fully aware of how to implement them.

Cyber-physical system (CPS) design is a recent engineering branch that combines computers and physical systems. Despite its success to deliver systems for military, medical, automotive, aircraft and many other applications, CPS design is well-known to be a vibrant research area mainly because it tries to combine different design approaches from distinct mature areas with established, and often incompatible, abstractions.

This paper presents the approach adopted by the ForSyDe-Atom framework for modeling CPS [2] and our discussion gravitates around the circuit shown in Fig. 3a. Despite its apparent simplicity, it reveals the kind of complexity which we believe lies in the core of the CPS design problem, to find synergy between different well-understood abstractions for a system's perspective of time. The system consists of a continuous time RC low-pass filter with switches sw_1 and sw_2 that change the circuit topology under the control of some digital machine. Our approach deconstructs the system into its underlying computational structures in a disciplined manner as we discuss in the following sections.

II. Discrete Meets Continuous

Humans have an intuitive and subjective notion of time. When designing complex systems and choosing proper abstractions, however, one important task is to give clear and unambiguous semantics to *time*. Different models of computation (MoCs) thus provide different abstractions for time. We are interested in classifying two interacting types of systems, building upon the *tagged signal model*, a formal framework pioneered by Lee and Sangiovanni-Vincentelli [3] for unifying the main domains of CPS in terms of causality, models of time, precedence relationships, synchronization points, and other key properties.



(a) ODE (b) Mealy state machine

Fig. 1: Structures of discrete and continuous computation

a) Continuous Time (CT) Systems: are described by a class of *timed* behaviors, frequently used when modeling physical phenomena. From the perspective of the tagged signal model, CT is a MoC where the set of tags is a *continuum* (i.e. a closed connected set) and has a *metric* (i.e. is a totally ordered Abelian group). This framework formalizes the fact that CT behaviors are defined over *a continuous span of time*, but also derives properties of causality and determinism between different observations of the system. In engineering practices rather than talking about causality, continuous systems are described by their *dynamics*, i.e., how the system evolves over time. In the most general case dynamics are modeled as a set of *ordinary differential equations (ODEs)* of form $\dot{x} = f(x,t)$, traditionally designed as structures similar to Fig. 1a.

b) Discrete Systems: are denoting in this paper all *causal* or *monotonic* systems which are driven by a notion of *discrete* events. In this case discrete does not necessarily refer to physical time, but rather the causal computation based on quantified observations. We can encompass a variety of MoCs spanning from timed to untimed, all having in common the ability to model digital systems at different levels of abstraction. While a rigorous analysis on the semantics of execution exposes important properties of causality and determinism for each of these MoCs [3], the common property we are interested in is

978-3-9819263-0-9/DATE18/©2018 EDAA

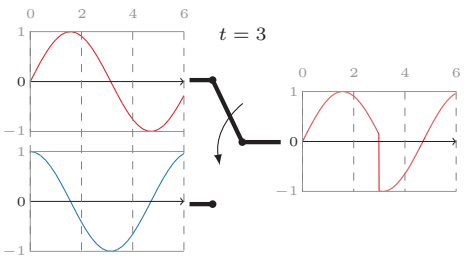


Fig. 2: Interpretation of CT signal as bounded sub-signals. This example represents the signal $\langle \sin(t)_{@0}, \cos(t)_{@3} \rangle$

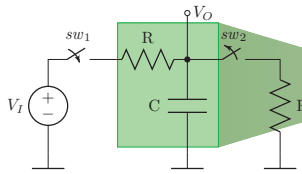


Fig. 3a: RC oscillator circuit

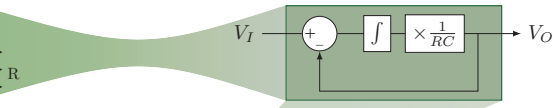


Fig. 3b: Block diagram for RC filter

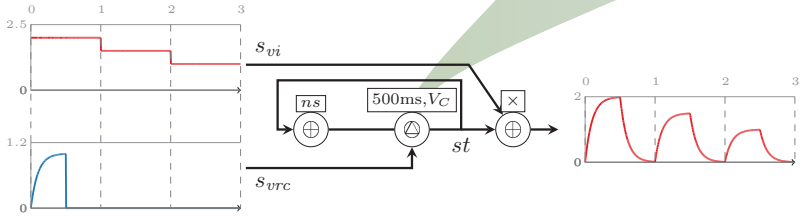


Fig. 3c: ForSyDe-Atom model of RC oscillator

the ability to perform computation. Computation is commonly modeled in digital domains as *state machines*, such as Fig. 1b.

While the tagged signal model enable a common framework to reason about different notions of time and precedence, it does not solve a much more subtle problem: how domains perceive each other. In the world of physics continuous functions are well-behaved whereas discontinuities are notoriously difficult to handle using complex mathematical models for approximating the discrete phenomena. In the domain of software, continua do not exist and are merely projections and predictions based on incomplete data, whereas edges are always abrupt [1]. The intrinsic nature of CPS forces the integration of the two views: a discrete component will always react to continuous stimuli using its own semantics whereas physical models need to be able to deal with the resulting discontinuities in a way which is useful to the modeler, e.g. flipping a switch.

III. The Atom Approach

We address the problems posed in the previous section from the perspective of the ForSyDe-Atom framework for modeling CPS [2] in order to describe analog and mixed signal systems. This framework is defined as a domain-specific language (DSL) embedded in the functional programming language Haskell, and is founded on three basic principles of modeling: 1) the separation of the manifold concerns in CPS as individual interacting *layers*; 2) the deconstruction of semantics of each layer to their core in form of primitive building blocks called *atoms*; 3) the reconstruction of functional modules by composing atoms in form of *patterns*.

The concepts of *layer* and *pattern* come in naturally when considering how functional programming is dealing with algebraic data types (i.e. structures) to hierarchically abstract data and how each variable itself is a program abstraction. This paradigm is associated with an algebra of combining forms which grants the concept of higher-order function through which we can encase entities belonging to a certain layer (atoms operating on certain structures) within entities of other layers. Patterns are simply the result of defining function composition as program abstractions. Atoms, first mentioned in [4], are possible thanks to the applicative style of programming [5], where systematic partial application of arbitrary functions on structured data is done through "atomic" rules. The layer which is studied in this paper is called the *MoC layer* and is concerning timing and synchronization issues.

A. Bounded Sub-signals

Introduced in [2], ForSyDe-Atom defines processes like in (1) as *functional mappings* from signal(s) to signal(s), using the same reasoning as [6]–[9]. As such, the main data type defined by the MoC layer is the *signal* $\mathcal{S}$ described in (2) as a stream of events. Each *event* $\mathcal{E}$ is a structure encoding the tag system of the MoC it belongs to. A stream is modeled as an infinite list [10], thus inferring a total order between events. From lists, we inherit the operation '::' which prepends an element at the beginning of a stream (3).

$$p : \mathcal{S}(\alpha)^m \rightarrow \mathcal{S}(\beta)^n \tag{1}$$

$$\mathcal{S}(\alpha) = \langle \mathcal{E}(\alpha) \rangle \tag{2}$$

$$\alpha :: \langle \alpha \rangle \rightarrow \langle \alpha \rangle \tag{3}$$

$$\mathcal{E}_{CT}(\alpha) = (t', t \mapsto \alpha) \overset{\text{not.}}{\equiv} t \mapsto \alpha_{@t'} \tag{4}$$

Our idea for representing CT signals in a practical manner is to define the events $\mathcal{E}_{CT}$ (2) as wrapping a discrete tag system around *functional mappings* from the *complete span of time* to a *set of values* $F : T \rightarrow V$, as shown in (4). Following the example of [6]–[9], these F are in fact well-defined CT signals themselves. Indeed, this model not only fits well with the idea discussed in [1] that events are discrete changes in a continuous function, but also enforces the view adopted by [11], where continua are conservatively incorporated in programs under the supervision of discrete controllers. We can thus justify the type of events in (4) as containing a *discrete timestamp* $t'_i \in T' \subset T$, and a CT sub-signal bounded on the domain $[t'_i, t'_j)$, where t'_j is either the starting time of the next sub-signal or ∞. In this regard, the domain of a CT signal is a continuum itself since it is composed of connected disjoint sets. Naturally, this interpretation is valid only for one-sided systems, and we consider time 0 as the global start.

Fig. 2 shows a practical interpretation of such a CT signal. On the left side there are two signals $\sin(t)$ and $\cos(t)$ which are continuous in the true sense, i.e. they have always happened and will always happen. The mere presence of an observer defines time 0 as the start of the observation, and

$$\text{LIFT} \quad \odot : (\alpha \to \beta) \to \mathcal{S}(\alpha) \to \mathcal{S}(\beta) \qquad g \odot f_{@t'} :: s = (g \circ f)_{@t'} :: (g \odot s) \tag{5}$$

$$\text{SYNC} \quad \circledast : \mathcal{S}(\alpha \to \beta) \to \mathcal{S}(\alpha) \to \mathcal{S}(\beta)$$
$$g_{@q'} :: s_g \circledast f_{@t'} :: s_f = \begin{cases} (g \circ f)_{@q'} :: (s_g \circledast s_f) & \text{when } q' = t' \\ (g \circ f)_{@q'} :: (s_g \circledast f_{@t'} :: s_f) & \text{when } q' < t' \\ (g \circ f)_{@t'} :: (g_{@q'} :: s_g \circledast s_f) & \text{when } q' > t' \end{cases} \tag{6}$$

$$\text{PREFIX} \quad \oslash : \mathcal{S}(\alpha) \to \mathcal{S}(\alpha) \to \mathcal{S}(\alpha) \qquad f_{@\_} :: \_ \oslash s = f_{@0} :: s \tag{7}$$

$$\text{SHIFT} \quad \oplus : \mathcal{S}(\alpha) \to \mathcal{S}(\alpha) \to \mathcal{S}(\alpha)$$
$$s_d \oplus f_{@t'} :: s = (t \mapsto f(t - d'))_{@t'+d'} :: (s_d \oplus s) \quad \text{where } s_d = \_ :: \_{@d'} :: \_ \tag{8}$$

$$\text{COMB} \quad f \oplus (s_1, s_2, s_3, ...) = f \odot s_1 \circledast s_2 \circledast s_3 \circledast ... \tag{9}$$

$$\text{DELAY} \quad s_d \mathbin{@} s = s_d \oslash (s_d \oplus s) \tag{10}$$

time 3 as the relative (discrete) moment when the observer flips the switch. This behavior is described by the output CT signal on the right side which, as of (2) and (4), is denoted by the structure $\langle \sin(t)_{@0}, \cos(t)_{@3} \rangle$.

B. Atoms

The ForSyDe-Atom framework currently defines four atoms in the MoC layer as process constructors similar to [8], [12]. Along with the tagged signal (2) these establish a minimal algebra of timed/causal interactions capable of describing a wide variety of CPS models [2]. Each MoC overloads these atoms with its specific (operational) semantics. For the CT MoC, atoms are recursively defined in (5)–(8) as infix operations, where '::' binds tighter than atoms, and the '_' symbol suggests a wildcard. The LIFT atom in (5) applies composition with an arbitrary function on all sub-signals. It is defined as a higher-order function and it has the ability to "lift" an entity from the layer below into the MoC layer. The SYNC atom in (6) synchronizes two CT signals while applying composition on the sub-signals accordingly. The PREFIX atom in (7) ensures *prefix order* in signals (sequences of $\mathcal{E}_{CT}$), by prepending the first sub-signal of the LHS at the head of the RHS. Finally, the SHIFT atom in (8) aids in manipulating the tags in a way that ensures *strict causality*, i.e. "delays" all events in the RHS with the duration of the first sub-signal in the LHS. Eq. (8) defines a simplified behavior of SHIFT for the sake of readability. Let the generic patterns COMB defined in (9) denote a combinational (stateless) process with an arbitrary number of inputs and DELAY defined in (10) denote a state process which, naturally, inflicts a (strictly) causal delay.

The atom definitions (5)–(8) enforce the separation of concerns of discrete and continuous time into orthogonal layers, also suggested in (4). Continuous subsignals interact by means of function composition $\circ$, while discrete events react by means of implicit rules for timestamp manipulation.

C. Modeling Example

To exemplify the continuous-discrete interactions within ForSyDe-Atom, we model the voltage output of the RC oscillator circuit from Fig. 3a. This setup is particularly interesting as the charging and discharging cycles are controlled *discretely* through the switches sw_1 and sw_2. For this particular example we consider the switching of sw_1 and sw_2 as being periodic and part of the system specification. As such, we can model the behavior of the RC oscillator with the discrete signal generator in Fig. 3c[1], which is a special case of Mealy state machine. Fig. 3b shows the computational structure of the continuous time dynamic behavior inside an RC circuit, whereas Fig. 3c shows the computational structure of the discrete time part of the system. As established in the previous sections, the two domains are separated within their own layer, and it is the role of atoms (5)–(8) to provide a clear interface between them, which will be detailed next. For the scope of this paper we take for granted the solving of ODEs, and Fig. 3b is so far meant only to justify eq. (11).

To enable this behavior of the circuit in Fig. 3a, we need to initially load the *state process* $\circlearrowleft$ with the voltage response rule V_{RC} from (11). According to (10), $\circlearrowleft$ needs to extract V_{RC} from the first sub-signal of s_{vrc} which in our case has the duration of $\tau = 500$ ms. s_{vrc} is constructed like in (13).

$$V_{RC}(t) = 1 - e^{\frac{-t}{RC}} \tag{11}$$

$$ns(V(t)) = 1 - V(t) \tag{12}$$

$$s_{vrc} = \langle V_{RC}(t)_{@0}, t \mapsto 0_{@\tau} \rangle \tag{13}$$

In the initial state of the system, the process $\circlearrowleft$, due to its $\oslash$ component defined in (7), is capable of creating the $st^{(0)}$ signal by prepending the first sub-signal of s_{vrc} to a previously existing empty signal $\langle\rangle$, thus creating the singleton in (14). By feeding st through a combinational process back to $\circlearrowleft$ we in fact define a recurrence having the initial function $st^{(0)}$. The *next state* combinational process $\oplus$ alters the current voltage response by applying function (12) to describe the charging/discharging of capacitor C, i.e. alternately opening/-closing of switches sw_1 and sw_2 from Fig. 3a. On account of its $\oplus$ component defined in (8) the recurring $\circlearrowleft$ process is able to take the next state signal $ns \oplus st^{(0)}$, phase-shift it with τ, and use it as base for the same prepending mechanism that created $st^{(0)}$ in order to create $st^{(1)}$ (15). $st^{(2)}$ (16) is similarly constructed by "delaying" $ns \oplus st^{(1)}$ and so on. We thus have a recursive definition for the infinite signal st which describes the dynamics of the circuit in Fig. 3a controlled periodically by discrete switches, with period τ.

$$st^{(0)} = V_{RC}(t)_{@0} :: \langle\rangle = \langle V_{RC}(t)_{@0} \rangle \tag{14}$$

$$\begin{aligned} st^{(1)} &= V_{RC}(t)_{@0} :: \langle (ns \circ V_{RC})(t-\tau)_{@0+\tau} \rangle \\ &= \langle V_{RC}(t)_{@0}, (1 - V_{RC})(t-\tau)_{@\tau} \rangle \end{aligned} \tag{15}$$

$$\begin{aligned} st^{(2)} &= V_{RC}(t)_{@0} :: \langle (ns \circ V_{RC})(t-\tau)_{@0+\tau}, \\ &\quad (ns \circ (1 - V_{RC}))(t-\tau-\tau)_{@\tau+\tau} \rangle \\ &= \langle V_{RC}(t)_{@0}, (1 - V_{RC})(t-\tau)_{@\tau}, V_{RC}(t-2\tau)_{@2\tau} \rangle \end{aligned} \tag{16}$$

Finally, the *output decoder* process $\oplus$ scales the sub-signal stored in the current voltage response rule with an arbitrary V_I, by synchronizing st with s_{vi} and applying multiplication. The synchronization mechanism is suggested by the definition of $\circledast$ in (6), and implements a conservative discrete event evaluation, i.e. the output reacts causally whenever a new event occurs, regardless of its origin.

[1]links to report, executable source and extended documentation of the DSL are found at `https://github.com/forsyde/forsyde-atom`

D. Scope. Advantages. Limitations

It is of utmost importace to understand the role of the algebra of atoms presented in this paper which is to describe *discrete interactions* of continuous signals, and *not* to model continuous dynamics. In the current paper, the dynamics themselves are taken for granted as being solved, and are "lifted" into the layer of MoCs from a layer of functions below. It falls in the scope of the function layer to formalize continuous dynamics in form of a proper algebra. Moreover, as described in the previous section, Fig. 3c is a simplified and imperfect model of the circuit in Fig. 3a, and behaves correctly only under certain conditions, e.g. $\tau \geq 2\pi RC$. However, Lee advocates in [1] the usefulness of such simplified models in engineering practices, and even promotes a similar idea to ours for describing hybrid systems: encapsulate continua within state machines as modal models.

An essential advantage derives directly from the functional paradigm: the system, instantiated as a process network, is abstracted from any numerical representation. Indeed, the input and output signals are themselves *functions of time*, the only operations performed being function compositions. The numerical computations along with their associated quantization errors take place *only* when the output is asked to evaluate the value of the output signal at a certain time. Another consequence is that the system is practically independent of the time representation, an (eventual) time resolution being inferred from the plotter/evaluator alone [9].

A final property of the algebra of atoms is that it can describe deterministic feedback loops only if they are *strictly causal*, i.e. are split by a DELAY $\oslash$ process with $\tau > 0$. This can easily be justified considering the machines handling the interactions. Should one assume that the loop is instantaneous ($\tau = 0$), the behavior of Fig. 3c for example is non-deterministic as it describes an infinity of values in the same time instant, which is the equivalent of a Zeno condition, e.g. simulation time never advances. However, instantaneous loops such as Fig. 1a is possible by modeling and solving algebraic and differential equations within the function layer.

IV. Related Work

Hybrid and CPS system modeling is a dynamic research field. Ptolemy II/CyPhySim [13] is concerning multi-domain systems and has a hierarchical notion of time. Its modal MoC promotes a similar idea to ours, to include continuous domains inside state machines. However, conformance of execution semantics are ensured by a central director, whereas atoms are self-sufficient units encapsulating those semantics. In this case, ForSyDe can be rather compared to the synchronous language of parallel ODEs ZÉLUS [11], which conservatively extends the LUSTRE language core with CT constructs and ensures that discrete computations are aligned with "zero-crossings".

Most modeling and simulation frameworks (e.g. Simulink) suffer from the numerical representation of time from the perspective of the host machine, thus enforcing a distorted notion continuum as vector of quantified observations. However, we see an emerging trend in orthogonalizing time from computation revolving around the functional programming paradigm, like FRP [6], FHM [7], and UNITI [9]. These frameworks are able to represent continua as abstractions, i.e. $f(t)$. Combined with strategies like lazy evaluation, it turns out to be a versatile technique where numerical representation becomes apparent only in the late stages of evaluation.

The idea of bounded sub-signals is inspired by approaches such as hiatons [12] or explicit bounds [14] exploiting streams as dataflow engines for evaluating continuous signals. A complementary dataflow-oriented approach revolves around subtyping the representations of time domains [6], [9], [11].

V. Conclusion

We present an algebra of interactions in hybrid systems based on the concepts of atoms and sub-signals within the ForSyDe-Atom framework, which orthogonalizes time representation from computation, but also continuous dynamics from discrete events. We show a simple but representative modeling example and analyze its benefits and limitations in accordance with the challenges posed by current research and industrial practices. This model proved to handle the problem of discrete structural changes in the description of a continuous system (i.e. flipping a switch) in a simple and elegant manner solely through function composition on continuous sub-domains.

Acknowledgment

This work has been partially supported by CNPq, the Brazilian National Scientific Development Council, by CISB, the Swedish-Brazilian Research and Innovation Office and by Saab AB.

References

[1] E. A. Lee, "Fundamental limits of cyber-physical systems modeling," *ACM Trans. Cyber-Phys. Syst.*, vol. 1, no. 1, pp. 3:1–3:26, Nov. 2016.

[2] G. Ungureanu and I. Sander, "A layered formal framework for modeling of cyber-physical systems," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2017, pp. 1715–1720.

[3] E. Lee and A. Sangiovanni-Vincentelli, "A framework for comparing models of computation," *IEEE J. Technol. Comput. Aided Design*, vol. 17, no. 12, pp. 1217–1229, Dec. 1998.

[4] M. K. Jakobsen *et al.*, "Formal methods for design and simulation of embedded systems," Ph.D. dissertation, TU Denmark, 2013.

[5] C. McBride and R. Paterson, "Applicative programming with effects," *J. Funct. Programming*, vol. 18, pp. 1–13, 2008.

[6] P. Hudak *et al.*, "Arrows, robots, and functional reactive programming," in *Lect. Notes Comput. Sc.* Springer, 2003, pp. 159–187.

[7] H. Nilsson *et al.*, "Functional hybrid modeling," in *PADL*, vol. 3. Springer, 2003, pp. 376–390.

[8] I. Sander and A. Jantsch, "System modeling and transformational design refinement in ForSyDe," *IEEE J. Technol. Comput. Aided Design*, vol. 23, no. 1, pp. 17–32, January 2004.

[9] K. C. Rovers and J. Kuper, "UNITI: Unified composition and time for multi-domain model-based design," *Int. J. Parallel Prog.*, vol. 41, no. 2, pp. 261–304, 2013.

[10] R. S. Bird, *An introduction to the theory of lists*. Springer, 1987.

[11] T. Bourke and M. Pouzet, "Zélus: A synchronous language with ODEs," in *Proc. 16th Int. Conf. Hybrid Systems: Computation and Control*, ser. HSCC '13. New York, NY, USA: ACM, 2013, pp. 113–118.

[12] H. J. Reekie, "Realtime signal processing," Ph.D. dissertation, School of Electrical Engineering, University of Technology at Sydney, 1995.

[13] E. A. Lee *et al.*, "Modeling and simulating cyber-physical systems using CyPhySim," in *Proceedings of the 12th International Conference on Embedded Software*, ser. EMSOFT '15, 2015, pp. 115–124.

[14] S.-H. Attarzadeh-Niaki and I. Sander, "An extensible modeling methodology for embedded and cyber-physical system design," *Simulation*, vol. 92, no. 8, pp. 771–794, 2016.