From Structured Requirement for Cyber-Physical Systems to Process Algebra: A Research Preview

Mathilde Arnaud[1], Boutheina Bannour[1], Guillaume Giraud[2], and Arnault Lapitre[1][0000-0002-2185-4051]

[1] Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France
[2] RTE

Abstract. [**Context and motivation**] Cyber-Physical Systems (CPS) are made up of complex real-time computational components that control physical entities. The design of these components must take into account non-determinism, intrinsic temporality and resilience to unwanted behaviors. [**Question/problem**] These aspects make it quite difficult to formulate requirements that describe CPS behaviors precisely, with the risk of being misunderstood or of conveying unintended design choices. [**Principal ideas/results**] We propose to use a controlled natural language to structure CPS requirements which has the advantage of: i) being easily graspable by stakeholders with various levels of proficiency so as to communicate clearly, ii) enabling requirements analysis using simulation or formal validation. [**Contribution**] To automate this analysis of structured requirements, we propose to translate them into a temporal process algebra. Our approach is implemented and is motivated by a real-world use case from european project CPS4EU. [3]

Keywords: Structured Requirements · CPS design · Process Algebra

1 Introduction

Context. Early validation of Cyber-Physical Systems (CPS) requires the consolidation of requirements. But it turns out to be a tedious task due the nature of CPS behavior. In fact, the control logic of physical devices can quickly become complex but the behavior of CPS shall remain reactive, available and resilient within acceptable times. In industry, CPS requirements are still mostly expressed in natural language. One major challenge is still the cross-check of such requirements. Missing or contradictory requirements can create a costly misunderstanding in the CPS development process.

[3] This work was financially supported by European commission through CPS4EU project that has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 826276. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Spain, Hungary, Italy, Germany.

Related work. The use of formal methods can help validate CPS requirements. Some approaches transform natural language requirements into LTL using NLP [1]. However, due to structuring and style variability, the formal set of requirements generated may not represent the intended meaning [2]. To prevent problems with the semantics obtained, other approaches offer to write the requirements in a strongly constrained language [7] which allows an automatic translation to temporal logic. The semantics obtained through any of these methods must be checked by a specialist. We are interested in works which propose to specify natural language requirements with template structures such as EARS [5] and Rupp [6]. These are a kind of fill-in templates which facilitate the specification of event-driven, state-driven system behaviors. Automating writing and analyzing such semi-formal requirements for particular domains of application is still a challenge [4].

Contribution. We ground our approach on EARS templates for the specification of CPS requirements in which we introduce timing details to refine the event-driven, state-driven system behaviors. This paper presents an automatic transformations of such requirements into a process algebra that we designed in the DIVERSITY tool [3]. Via the transformation, behaviors of CPS systems specified by the requirements can be explored in the tool. A simplified version of a CPS4EU case study illustrates our approach throughout the paper.

2 Requirement specification

In order to handle real-world applications of state-driven, event-triggered systems with timing, we enrich process algebras with timing concepts and modalities. Our goal is to express the set of requirements with process algebra so as to be able to generate scenarios displaying the behavior of the system as specified by the requirements to help evaluate their accuracy.

```
1 viewRequirements
```

executed in 5ms, finished 18:11:04 2020-11-16

ID	Requirement Statement
R1	**every** 5 seconds, **the** **NAZA Core** **shall** calculate levers setpoints.
R2	**when** new levers setpoints have been determined **upon** levers setpoints calculation, **the** **NAZA Core** **shall** determine common levers by using consensus.
R3	**every** 5 seconds **when** consensus **upon** common levers determination, **the** **NAZA Core** **shall** send batteries setpoints.
R4	**every** 5 seconds **when** consensus **upon** common levers determination, **the** **NAZA Core** **shall** send topological order.
R5	**every** 5 seconds **when** consensus **upon** common levers determination, **the** **NAZA Core** **shall** send modulation orders.
R6A	**if** no result **upon** levers setpoints calculation, **the** **NAZA Core** **shall** execute back up algorithm within [10,60] seconds.
R6B	**if** no result **upon** levers setpoints calculation **while** in nominal mode, **the** **NAZA Supervisor** **shall** enter in backup mode.
R7	**when** entering in backup mode, **the** **NAZA Supervisor** **shall** execute back up algorithm within [10,60] seconds, and return in nominal mode.
R8	**when** new setpoints **upon** levers setpoints calculation **while** in backup mode, **the** **NAZA Supervisor** **shall** enter in nominal mode.

Fig. 1: Requirements for NAZA (Nouveaux Automates de Zones Adaptatifs)

2.1 Motivating example/Illustration/Use case

We illustrate our approach by showing how it can be applied on a real-world use case : electrical networks involving intermittent energy sources. To avoid

overload without raising the overall network capacity, it is necessary to manage dynamically the flow of electricity through levers such as batteries or production modulation. Which mechanisms to trigger must be determined very quickly and this role must therefore be entrusted to a software component called NAZA.

We used our approach to analyze functional requirements R1 to R6A given in Fig. 1 written in Structured Natural Language (SNL) presented in Section 2.2. In this use case, the NAZA automaton is in charge of computing levers setpoints (cf R1). When the computation is successful, it then uses consensus (cf R2) and sends the results to middleware (cf R3,R4,R5). When the computation fails, it must launch a backup algorithm (cf R6A). Our analysis revealed possible deadlocks and we proposed to replace requirement R6A by requirements R6B, R7 and R8, using two modes (*nominal* and *backup*) to express the behavior more precisely.

2.2 Structured Natural Language

We have structured requirement statements by using a grammar based on well-established method EARS [5]. A user-defined glossary is tailored to the needs of the requirement engineer : it defines systems, triggers, and also equivalence for ease of use (e.g. "calculate levers setpoints"/"levers setpoints calculations"). Each requirement statement is expressed by a (possibly complex) precondition, followed by a realization, which specifies the action of the system.

Nominal and **unwanted** behavior requirements are initiated only when a triggering event occurs, they are constructed respectively with keywords **when** (e.g. R7) and **if** (e.g. R6B). **State-Driven** requirements are active while the system is in a defined state and are constructed with keyword **while** (e.g. R6B).

We introduce details to enhance the sequencing of system behaviors: they can be triggered periodically, subsequently to other behaviors, or within some time slot. **Periodic behaviors** are initiated according to a specified period, they are expressed through pattern "**Every** ⟨ period ⟩" (e.g. R1, R3, R4, R5). The context in which requirements are executed can be detailed through two constructs : "**upon** ⟨ system response ⟩" specifies that the behavior happens subsequently to some other behavior; "**within** ⟨ timing interval ⟩" specifies that the behavior happens within the specified interval. R6B is an example of a combined use of these constructs, exemplifying that requirements can be complex and use several of these constructs at the same time.

In order to prevent ambiguity arising from the use of synonyms, we favor the use of repetitions of expressions in the glossary (e.g. R3, R4, R5). This makes requirements as simple as possible and thus preserves readability and unity.

3 Target process algebra

Time domain and datatype. Clocks are typed in a dense time domain T isomorphic to the set of positive rational numbers $\mathbb{Q}_+$. Given a set of clocks Clk, a clock valuation v is a mapping $v : Clk \to T$. The set $\mathcal{F}(Clk)$ of clock formulas is

built up recursively out of logical conjunction, disjunction and atomic formulas of the form $True$, $False$, $clk \bowtie d$, where d is a constant duration (typed in T) and $\bowtie \in \{<, \leq, >, \geq\}$. The set of clock invariants $\mathcal{I}(Clk)$ is defined by conjunctions of formulas of the form $clk \leq d$ or $clk < d$. Valuations can be canonically extended to formulas as usual.

Actions and states. Let Act be a set of actions which contains the silent action $\tau \in Act$. Let $A \subseteq Act \setminus \{\tau\}$ be a partition $I \cup O$. Elements a of I (resp. of O) are called inputs and denoted by $?a$ (resp. called outputs and denoted by $!a$). In a parallel composition, inputs and outputs can synchronize resulting in τ. We denote $\overline{?a} = !a$ (and vice versa). Let S be set of states with initial state $s_0 \in S$.

Processes. A process term is defined by the following syntax:

$$p ::= [s][\phi]a\{R\} \triangleright s' \mid \mathsf{inv}(\psi) \mid \mathsf{nil} \mid p;p \mid \mathsf{alt}(p,p) \mid \mathsf{par}(p,p) \mid \mathsf{loop}(p)$$

The basic building block of a process is of the form $atom = [s][\phi]a\{R\} \triangleright s'$ with an enabling state $s \in S$, an enabling clock formula $\phi \in \mathcal{F}(Clk)$, an action $a \in Act$, a set of clocks $R \subseteq Clk$ to be reset, and a target state $s' \in S$ to evolve into. We may consider simpler atoms where some of these elements are dropped. For instance $[\phi]a\{R\}$ denotes that enabling state can match any arbitrary state and that no state change is to be made. $\mathsf{inv}(\psi)$ with $\psi \in \mathcal{I}(Clk)$ defines the clock invariant that has to be satisfied on time passing; nil is the empty process; processes can be composed in sequence (;); using alternatives (alt); in parallel (par); or repeated in sequence zero or more times(loop).

Small-step execution. Parallel processes can interleave but they must comply with current invariants. We introduce a decomposed form of invariants whose purpose is to identify invariant of left or right process of a parallel composition. A concurrent invariant, or *co-invariant*, is defined as follows, in which $\psi \in \mathcal{I}(Clk)$:

$$\Psi ::= \psi \mid \Psi \wedge_{\_} \Psi \mid \Psi \wedge_L \Psi \mid \Psi \wedge_R \Psi$$

We define functions L, R and f on co-invariants that return respectively the left side of the co-invariant, the right side, and the conjunction formula representing the conditions at the time of evaluation. If Ψ is in a decomposed form $\Psi_1 \wedge_X \Psi_2$ with $X \in \{\_, L, R\}$, $L(\Psi)$, $R(\Psi)$ and $f(\Psi)$ denote Ψ_1, Ψ_2 and $f(\Psi_1) \wedge f(\Psi_2)$ respectively, otherwise $L(\psi)$, $R(\psi)$, $f(\psi)$ is ψ.

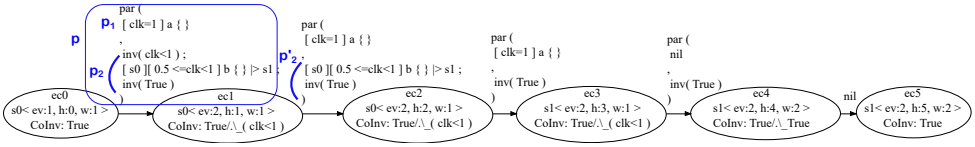


Fig. 2: Execution graph of a toy process

Operational rules of process execution are given in Table 1. The execution is defined up to an execution context $ec = (s, v, \Psi)$ which represents the necessary information to perform an execution step, namely the current state s, the current valuation v of clocks and the current co-invariant Ψ to be applied. The execution process is inductively defined on the form of the process term. If p is an $atom$ then it evolves to nil under the constraint that time elapsing is compatible with its clock formula and current co-invariant. If p is an invariant $\mathsf{inv}(\psi)$, then function *coinv_upd* is called to update the relevant side of the co-invariant. The case

of parallel composition is the most subtle when dealing with interleaving (other rules are classic). It uses a function *coinv* which inductively computes the formula of a co-invariant of process p given a current co-invariant Ψ (of an ec). Concerned part of Ψ is returned in case no new invariant is encountered.

A toy process p is given in Fig.2 for illustration. It shows how actions a and b will be interleaved in the context of the parallel composition. From initial context $ec_0 = (s_0, [clk \to 0, clk \in Clk], \Psi_0)$ with $\Psi_0 = True$, both left process p_1 and right process p_2 are evaluated. p_2 evolves into p_2' which allows to compute a new context ec_1. On the other hand p_1 cannot be executed. In fact, the rule PAR1L requires the execution of p_1 under co-invariant $L(\Psi_0) \wedge_L coinv(p_2, R(\Psi_0))$ in which $L(\Psi_0) = True$ and $coinv(p_2, R(\Psi_0)) = \mathsf{clk} < 1$. Yet the execution of p_1 is enabled by formula $\mathsf{clk} = 1$, while time elapses of a duration < 1 at this point of execution in accordance with co-invariant. Later from context ec_3, the execution of process p_1 becomes possible: rule PAR1L applies as co-invariant becomes $True \wedge_L True$ (was $True \wedge_L \mathsf{clk} < 1$ as explained before) which allows time elapsing with any duration. Once the co-invariant is applied on some leaf process (here left), it is returned in a neutral form $True \wedge_\_ True$. When both parallel processes end, the co-invariant is merged by rule $\mathsf{PAR3}_{\mathsf{join}}$ into $True$.

$$\mathsf{ATOM1}\ \frac{}{[s][\phi]a\{R\}\triangleright s'\ (s,v,\Psi)\ \overset{d.a}{\rightsquigarrow}\ \mathsf{nil}\ (s',v',\Psi)}\ \begin{pmatrix} v_0 = v[clk \to clk + d, clk \in Clk], d \in T \\ v_0 \vDash \phi \wedge f(\Psi) \\ v' = v_0[clk \to 0, clk \in R] \end{pmatrix}$$

$$\mathsf{INV}\ \frac{}{\mathsf{inv}(\psi)\ (s,v,\Psi)\ \overset{\epsilon}{\rightsquigarrow}\ \mathsf{nil}\ (s,v,coinv\_upd(\Psi,\psi))}\ (\, v \vDash f(\Psi)\,)$$

$$\mathsf{ALT1L}\ \frac{p_1\ ec\ \overset{\ell}{\rightsquigarrow}\ p_1'\ ec'}{\mathsf{alt}(p_1,p_2)\ ec\ \overset{\ell}{\rightsquigarrow}\ ec'}\qquad \mathsf{SEQ1}\ \frac{p_1\ ec\ \overset{\ell}{\rightsquigarrow}\ p_1'\ ec'}{p_1;p_2\ ec\ \overset{\ell}{\rightsquigarrow}\ p_1';p_2\ ec'}\qquad \mathsf{LOOP1}\ \frac{p\ ec\ \overset{\ell}{\rightsquigarrow}\ p'\ ec'}{\mathsf{loop}(p)\ ec\ \overset{\ell}{\rightsquigarrow}\ p';\mathsf{loop}(p)\ ec'}$$

$$\mathsf{ALT2L}\ \frac{}{\mathsf{alt}(\mathsf{nil},p)\ ec\ \overset{\epsilon}{\rightsquigarrow}\ \mathsf{nil}\ ec}\qquad \mathsf{SEQ2}\ \frac{}{\mathsf{nil};p\ ec\ \overset{\epsilon}{\rightsquigarrow}\ p\ ec}\qquad \mathsf{LOOP2}\ \frac{}{\mathsf{loop}(p)\ ec\ \overset{\epsilon}{\rightsquigarrow}\ \mathsf{nil}\ ec}$$

$$\mathsf{PAR1L}\ \frac{p_1\ (s,v,L(\Psi)\wedge_L coinv(p_2,R(\Psi)))\ \overset{\ell}{\rightsquigarrow}\ p_1'\ (s',v',\Psi')}{\mathsf{par}(p_1,p_2)\ (s,v,\Psi)\ \rightsquigarrow\ \mathsf{par}(p_1',p_2)\ (s',v',coinv\_upd(L(\Psi)\wedge_L R(\Psi),\Psi'))}$$

$$\mathsf{PAR2}_{\mathsf{sync}}\ \frac{\begin{array}{l} p_1\ (s,v,L(\Psi)\wedge_L R(\Psi))\ \overset{d.a}{\rightsquigarrow}\ p_1'\ (s',v_1',\Psi') \\ p_2\ (s,v,L(\Psi)\wedge_R R(\Psi))\ \overset{d.\overline{a}}{\rightsquigarrow}\ p_2'\ (s',v_2',\Psi') \end{array}}{\mathsf{par}(p_1,p_2)\ (s,v,\Psi)\ \overset{d.\tau}{\rightsquigarrow}\ \mathsf{par}(p_1',p_2')\ (s',v',\Psi')}\ \begin{pmatrix} v'(clk) = \begin{cases} v_1'(clk) & \textbf{if } v_1'(clk) = 0 \\ v_2'(clk) & \textbf{else} \end{cases} \\ clk \in Clk \end{pmatrix}$$

$$\mathsf{PAR3}_{\mathsf{join}}\ \frac{}{\mathsf{par}(\mathsf{nil},\mathsf{nil})\ (s,v,\Psi)\ \overset{\epsilon}{\rightsquigarrow}\ \mathsf{nil}\ (s,v,f(\Psi))}$$

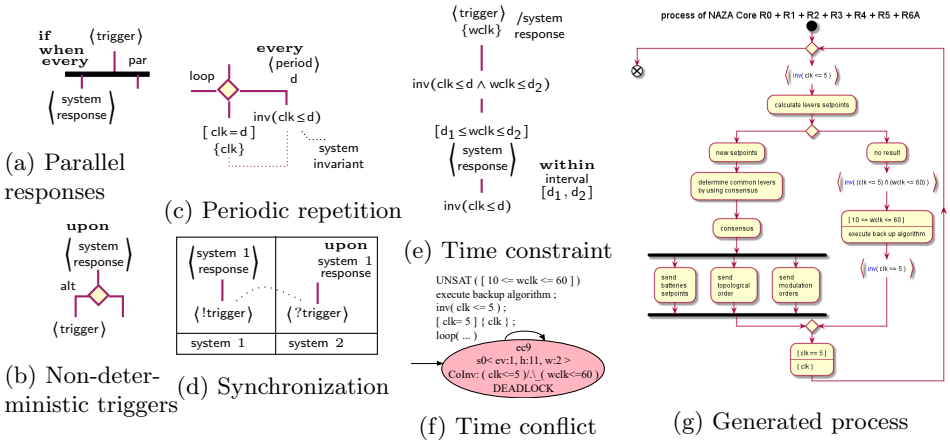
$$\begin{array}{l} coinv\_upd(\Psi,\Psi') = \textbf{match } \Psi \textbf{ with} \\ \quad \mid \psi \to \Psi' \quad \mid \Psi_1 \wedge\_ \Psi_2 \to \Psi' \quad \mid \Psi_1 \wedge_R \Psi_2 \to \Psi_1 \wedge\_ R(\Psi') \quad \mid \Psi_1 \wedge_L \Psi_2 \to L(\Psi') \wedge\_ \Psi_2 \\ coinv(p,\Psi) = \textbf{match } p \textbf{ with} \\ \quad \mid atom \to f(\Psi) \quad \mid \mathsf{par}(p_1,p_2) \to coinv(p_1, L(\Psi)) \wedge coinv(p_2, R(\Psi)) \\ \quad \mid \mathsf{nil} \to f(\Psi) \quad \mid \mathsf{alt}(p_1,p_2) \to coinv(p_1,\Psi) \vee coinv(p_2,\Psi) \\ \quad \mid \mathsf{inv}(\psi) \to \psi \quad \mid \mathsf{loop}(p) \to coinv(\mathsf{alt}(p,\mathsf{nil}),\Psi) \\ \quad \mid p_1;p_2 \to coinv(p_1,\Psi) \end{array}$$

Table 1: Process execution rules

Transformation. The main transformation patterns into process algebra are defined as follows: System responses which share the same trigger are composed in parallel (Fig.3a); triggers can be non-determistically produced upon a sys-

tem response (Fig.3b); it is assumed that each (sub-)system is reactive with a topmost enclosing loop (Fig.3c) which can be associated a period d ($\mathsf{inv}(\mathsf{clk} \leq \mathsf{d})$ constraints time elapsing of an iteration, at the end of which the system clock clk is reset, cf $[\mathsf{clk} = 5]\{\mathsf{clk}\}$). A synchronization is inferred if a system response is triggered by some other system behavior (Fig3d). When a system response occurs within an interval $[\mathsf{d}_1, \mathsf{d}_2]$ (Fig.3e), a dedicated clock wclk is used to encode such constraint ($\mathsf{inv}(\mathsf{clk} \leq \mathsf{d} \wedge \mathsf{wclk} \leq \mathsf{d}_2)$ sets the upper bound d_2 on time elapsing until the response occurs). State-driven triggers/responses are transformed by variants patterns involving enabling states or state change constructs of the process algebra. The transformation is illustrated by the NAZA Core process for requirements $\mathsf{R1} + \ldots + \mathsf{R5} + \mathsf{R6A}$ which depicted with the graphical convention of Fig.3g. It shows the compositional modeling using process algebra which provides powerful constructs to built larger processes up from smaller ones specified by the unit requirements of the SNL.

Tool support. We prototyped edition and transformation of the SNL requirements as a web application using Jupyter Notebook environment (Fig.1 & Fig.3g). We have implemented the small-step execution of process algebra in the model-based symbolic execution tool DIVERSITY [3]. Symbolic execution computes compact representation of execution graphs in which rational-valued clocks are assigned symbolic parameters rather than concrete values. SMT solvers are used to check satisfiability of clock formulas and produce a new (symbolic) execution context. The technique allowed us to explore efficiently behaviors of the generated processes and detect deadlocks. Those are in our case timelocks, typically non-compatible delays with system period, cf Fig.3f.



(a) Parallel responses

(b) Non-deterministic triggers

(c) Periodic repetition

(d) Synchronization

(e) Time constraint

(f) Time conflict

(g) Generated process

Fig. 3: Main transformation patterns (a)-(e) – Illustration

4 Conclusion and future work

We have shown how to infer processes from a set of requirements in structured natural language with timing and modal details for CPS. Our approach is applied on a real-world case issued from electricity transmission industry. Resulting processes can be displayed in a readable graphical depiction and can be unfolded by small-step into execution graphs. A challenging continuation is to design methods to elucidate (and visualize) relevant executions of the processes with requirement traceability. We also plan to apply the approach on more use cases in order to challenge the expressiveness of the set of structured requirements.

References

1. A. Fantechi, S. Gnesi, G. Ristori, M. Carenini, M. Vanocchi, and P. Moreschini. Assisting requirement formalization by means of natural language translation. In *Formal Methods in System Design*, volume 4, pages 243–263. Springer, 1994.
2. J. G. Greghi, E. Martins, and A. M. Carvalho. Semi-automatic generation of extended finite state machines from natural language standard documents. In *Int. Conf. DSN Workshops*, pages 45–50. IEEE, 2015.
3. The CEA List Institute. *Eclipse Formal Modeling Project*, 2020 (accessed November 15, 2020). `https://projects.eclipse.org/projects/modeling.efm`.
4. L. Lúcio, S. Rahman, C. Cheng, and A. Mavin. Just formal enough? automated analysis of EARS requirements. In *Int. Conf. NFM*, pages 427–434. Springer, 2017.
5. A. Mavin, P. Wilkinson, and M. Novak. Easy Approach to Requirements Syntax (EARS). In *Int. Conf. RE*, pages 317–322. IEEE, 2009.
6. K.P.C. Rupp. *Requirements Engineering Fundamentals: A Study Guide for the Certified Professional for Requirements Engineering Exam, Foundation Level–IREB Compliant*. Rocky Nook, 2015.
7. L. Wenbin, H. J. Huffman, and T. Mirosław. Temporal action language (TAL): A controlled language for consistency checking of natural language temporal requirements. In *Int. Conf. NFM*. Springer, 2012.