Decentralised Evaluation of Temporal Patterns over Component-Based Systems at Runtime

Olga Kouchnarenko[1,2] and Jean-François Weber[1(✉)]

[1] FEMTO-ST CNRS, University of Franche-Comté, Besançon, France
{okouchnarenko,jfweber}@femto-st.fr
[2] Inria/Nancy-Grand Est, Villers-lès-Nancy, France

Abstract. Self-adaptation allows systems to modify their structure and/or their behaviour depending on the environment and the system itself. Since reconfigurations must not happen at any but in suitable circumstances, guiding and controlling dynamic reconfigurations at runtime is an important issue. This paper contributes to two essential topics of the self-adaptation—a runtime temporal properties evaluation, and a decentralization of control loops. It extends the work on the adaptation of component-based systems at runtime via policies with temporal patterns by providings (a) specific *progressive semantics* of temporal patterns and (b) a *decentralised* method which is suitable to deal with temporal patterns of component-based systems at runtime. The implementation with the *GROOVE* tool constitutes a practical contribution.

1 Introduction

Self-adaptation—the ability of systems to modify their structure and/or their behaviour in response to their interaction with the environment and the system itself, and their goals—is an important and active research field with applications in various domains [1]. Since dynamic reconfigurations that modify the architecture of component-based systems without incurring any system downtime must not happen at any but in suitable circumstances, adaptation policies are used to guide and control reconfigurations at runtime. For triggering adaptation policies and specifying behaviours of component-based systems, a linear temporal logic based on Dwyer's work on patterns and scopes [2], called FTPL, has been used in [3]. In this adaptation context, choosing a suitable adaptation policy in a current component-based system configuration depends on a *runtime temporal patterns evaluation* which is one of the essential topics of the self-adaptation [1].

We consider *open* component-based systems interacting with their environment, therefore, their behaviour depends on both external and internal events. Since our component-based systems are modelled by infinite state transition

This work has been partially funded by the Labex ACTION, ANR-11-LABX-0001-01.

© Springer International Publishing Switzerland 2015
I. Lanese and E. Madelaine (Eds.): FACS 2014, LNCS 8997, pp. 108–126, 2015.
DOI: 10.1007/978-3-319-15317-9_7

systems, for our pattern-based verification to remain tractable, we consider a non-blocking environment with incomplete information about the component-based system that enables all the external events.

In this setting, providing values for temporal patterns is a difficult task. In [3], a centralised evaluation of temporal patterns at runtime has been proposed. In order to contribute to *decentralization of control loops*—another self-adaptation topic, this paper addresses the FTPL *decentralised* evaluation problem on a reconfiguration path, and presents a method that is suitable to deal with temporal patterns of component-based systems. Indeed, as these patterns contain conjunctions or disjunctions of properties over components' parameters and relations, the evaluation of temporal patterns in a decentralised manner makes sense, and the sooner the better.

Inspired by the LTL decentralised evaluation [4] for closed systems, this paper provides a progressive FTPL semantics allowing a decentralised evaluation of FTPL formulae over open component-based systems – the first contribution. The second contribution consists of an algorithm to answer the temporal pattern decentralised evaluation problem in $\mathbb{B}_4$ and of the correctness and uniqueness results saying that whenever an FTPL property is evaluated in the decentralised manner, it matches the FTPL evaluation using the basic semantics in [3]. The implementation with the *GROOVE* tool [5] constitutes a practical contribution.

Related work. When checking properties of open systems, the idea is to satisfy a property no matter how the environment behaves. For non-universal temporal logics, this problem, called *module-checking*, is in general much harder than model-checking of closed systems in finite as well as in infinite settings [6,7], and it becomes undecidable with imperfect information about the control states [8]. Fortunately, for universal temporal logics as LTL, the module checking problem with both complete or incomplete information remains decidable in finite setting [6]; in particular, it is PSPACE-complete for LTL.

As temporal properties often cannot be evaluated to true or false during the system execution, in addition to *true* and *false* values, *potential true* and *potential false* values are used whenever an observed behaviour has not yet led to an acceptance or a violation of the property under consideration, forming the $\mathbb{B}_4$ domain like in RV-LTL [9]. Like in [10], in our framework external events can be seen as invocations of methods performed by (external) sensors when a change is detected in their environment.

Let us remark that this work is motivated by applications in numerous frameworks that support the development of components together with their monitors/controllers, as, e.g., Fractal [11], CSP‖B [12], FraSCAti [13], etc.

More generally, this paper aims to contribute to the development of new verification approaches for complex systems that integrate ideas of distributed algorithms [14].

Layout of the paper. After a short overview of a component-based model and of a linear temporal patterns logic in Sects. 2, 3 presents a specific progressive semantics of temporal patterns. Afterwards, Sect. 4 addresses the temporal pattern decentralised evaluation problem on a reconfiguration path by providing

an algorithm for such an evaluation in $\mathbb{B}_4$. Section 5 describes the implementation with the *GROOVE* tool and details an example of a location composite component. Finally, Sect. 6 presents our conclusion.

2 Background: Reconfiguration Model and Temporal Patterns

The reconfigurations we consider here make the component-based architecture evolve dynamically. They are combinations of basic reconfiguration operations such as instantiation/destruction of components; addition/removal of components; binding/unbinding of component interfaces; starting/stopping components; setting parameter values of components. In the remainder of the paper, we focus on reconfigurations that are combinations of basic operations.

In general, a system configuration is the specific definition of the elements that define or prescribe what a system is composed of. As in [15], we define a configuration to be a set of architectural elements (components, required or provided interfaces, and parameters) together with relations (binding, delegation, etc.) to structure and to link them, as depicted in Fig. 1.

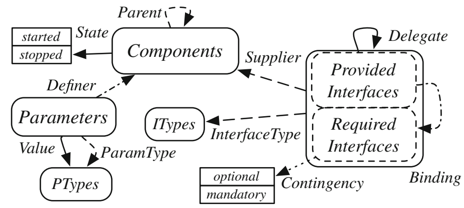


Fig. 1. Configurations = architectural elements and relations

Given a set of configurations $\mathcal{C} = \{c, c_1, c_2, \ldots\}$, let CP be a set of configuration properties on the architectural elements and the relations between them specified using first-order logic formulae. The interpretation of functions, relations, and predicates is done according to basic definitions in [16] and in [15]. A configuration *interpretation* function $l : \mathcal{C} \to CP$ gives the largest conjunction of $cp \in CP$ evaluated to true on $c \in \mathcal{C}$[1].

Among all the configuration properties, the architectural *consistency constraints* CC express requirements on component assembly common to all the component architectures. These constraints introduced in [17] allow the definition of *consistent configurations* regarding, in particular, the following rules:

- a component *supplies* one provided interface, at least;
- the composite components have no parameter;
- a sub-component must not include its own parent component;
- two bound interfaces must have the same interface type and their containers are sub-components of the same composite;
- when binding two interfaces, there is a need to ensure that they have not been involved in a delegation yet; similarly, when establishing a delegation link between two interfaces, the specifier must ensure that they have not yet been involved in a binding;

[1] By definition in [16], this conjunction is in CP.

- a provided (resp. required) interface of a sub-component is delegated to at most one provided (resp. required) interface of its parent component; the interfaces involved in the delegation must have the same interface type;
- a component is *started* only if its mandatory required interfaces are bound or delegated.

Definition 1 (Consistent configuration). *Let $c = \langle Elem, Rel\rangle$ be a configuration and CC the architectural consistency constraints. The configuration c is* consistent*, written* **consistent**(c)*, if $l(c) \Rightarrow CC$.*

Let $\mathcal{R}$ be a finite set of reconfiguration operations, and *run* a generic running operation. The possible evolutions of the component architecture via the reconfiguration operations are defined as a transition system over $\mathcal{R}_{run} = \mathcal{R} \cup \{run\}$.

Definition 2 (Reconfiguration model). *The operational semantics of component systems with reconfigurations is defined by the labelled transition system $S = \langle \mathcal{C}, \mathcal{C}^0, \mathcal{R}_{run}, \rightarrow, l\rangle$ where $\mathcal{C} = \{c, c_1, c_2, \ldots\}$ is a set of* consistent *configurations, $\mathcal{C}^0 \subseteq \mathcal{C}$ is a set of initial configurations, $\mathcal{R}$ is a finite set of reconfigurations, $\rightarrow \subseteq \mathcal{C} \times \mathcal{R}_{run} \times \mathcal{C}$ is the reconfiguration relation.*

Let us write $c \overset{ope}{\rightarrow} c'$ when c' is reached from c by $ope \in \mathcal{R}_{run}$. An evolution path σ (or a path for short) in S is a (possibly infinite) sequence of configurations $c_0, c_1, c_2, \ldots$ such that $\forall i \geq 0 . (\exists\, ope_i \in \mathcal{R}_{run}.(c_i \overset{ope_i}{\rightarrow} c_{i+1}))$. Let $\sigma(i)$ denote the i-th configuration of a path σ, σ_i a suffix path starting with $\sigma(i)$, and Σ the set of paths. An execution is a path σ in Σ such that $\sigma(0) \in \mathcal{C}^0$.

In this section, we also briefly recall the FTPL[2] logic patterns introduced in [18]. In addition to configuration properties (*cp*) in CP mentioned above, the proposed logic contains external events (*ext*), as well as events from reconfiguration operations, temporal properties (*tpp*) together with trace properties (*trp*) embedded into temporal properties. Let $Prop_{FTPL}$ denote the set of the FTPL formulae obeying the FTPL grammar below. The FTPL semantics from [3] is summarized in the long version of this paper [19].

<*FTPL*> ::= <*tpp*> | <*events*> | *cp*
<*tpp*> ::= **after** <*events*><*tpp*> | **before** <*events*><*trp*> | <*trp*> **until** <*events*> | <*trp*>
<*trp*> ::= **always** *cp* | **eventually** *cp* | <*trp*> ∧ <*trp*> | <*trp*> ∨ <*trp*>
<*events*> ::= <*event*>,<*events*> | <*event*>
<*event*> ::= *ope* **normal** | *ope* **exceptional** | *ope* **terminates** | *ext*

In the rest of the paper, let AE be the set of atomic events composed of atomic propositions from CP and of basic FTPL events. An *event* θ is an element of $\Theta = 2^{AE}$. Let us suppose that each component C_i of the component-based system has a local monitor M_i attached to it, from the set $\mathcal{M} = \{M_0, \ldots, M_{n-1}\}$ of monitors[3]. Let us introduce the projection function Π_i to restrict events to

[2] FTPL stands for TPL (Temporal Pattern Language) prefixed by 'F' to denote its relation to Fractal-like components and to first-order integrity constraints over them.

[3] Implemented as controllers in CSP‖B, Fractal, FraSCAti, etc.

the local view of the monitor M_i. For atomic events, $\Pi_i : 2^{AE} \rightarrow 2^{AE}$, and we write $AE_i = \Pi_i(AE)$. We assume $\forall i, j \leq n.i \neq j \Rightarrow AE_i \cap AE_j = \varnothing$[4]. Similarly, for events, we define $\Pi_i : 2^{\Theta} \rightarrow 2^{\Theta}$, with $\Theta_i = \Pi_i(\Theta)$, and we assume $\forall i, j \leq n.i \neq j \Rightarrow \Theta_i \cap \Theta_j = \varnothing$.

Let $ev : \mathcal{C} \rightarrow \Theta$ be a function to associate events with configurations. Given a configuration $\sigma(j)$ of a path σ with $j \geq 0$, the corresponding event is $\theta(j) = ev(\sigma(j))$. In this setting, an individual *behaviour* of a component C_i can be defined as a (finite or infinite) sequence of events $\theta_i = \theta_i(0) \cdot \theta_i(1) \cdots \theta_i(j) \cdots$ s.t. $\forall j \geq 0.\theta_i(j) = \Pi_i(ev(\sigma(j))) \in \Theta_i$, also called a *trace*. Finite (resp. infinite) traces over Θ are elements of Θ^* (resp. Θ^ω); the set of all traces is $\Theta^\infty = \Theta^* \cup \Theta^\omega$.

3 FTPL Progression and Urgency

This section provides the underpinnings to allow a decentralised evaluation of FTPL formulae. Inspired by definitions in [4], our notions of progression and urgency are adapted to the FTPL semantics: they take into account external and internal events as well as scopes of linear temporal patterns.

For decentralised evaluation of the FTPL formulae, instead of the set $\mathbb{B}_4$ as in [3], let us consider the set $\mathbb{B}_5 = \{\bot, \bot^p, \#, \top^p, \top\}$, where $\bot, \top$ stand resp. for *false* and *true* values, $\bot^p, \top^p$ for *potential false* and *potential true* values, and # for *unknown* value. We consider $\mathbb{B}_5$ together with the truth non-strict ordering relation $\sqsubseteq_5$ satisfying $\bot \sqsubseteq_5 \bot^p \sqsubseteq_5 \top^p \sqsubseteq_5 \top \sqsubseteq_5 \#$. On $\mathbb{B}_5$ we define two binary symmetric operations $\sqcap_5$, $\sqcup_5$ resp. as the minimum and maximum interpreted wrt. $\sqsubseteq_5$. Thus, $(\mathbb{B}_5, \sqsubseteq_5)$ is a finite lattice but not a Boolean nor a de Morgan lattice. Let $\forall \varphi \in Prop_{FTPL}.\varphi \sqcap_5 \# = \varphi$. We write $\sqcup$ and $\sqcap$ instead of $\sqcup_5$ and $\sqcap_5$ when it is clear from the context. For any formula $\varphi \in Prop_{FTPL}$, let $\hat{\varphi}$ denote the value of φ in $\mathbb{B}_5$.

In the context of a decentralised evaluation, each monitor may not be aware of information related to a given property and may be not able to evaluate it. This property is then written as a formula in terms of the current configuration. However, after the transition to the next configuration, such a formula may be not relevant. To compensate for this, we define the progression function to rewrite FTPL formulae in a way relevant to the next configuration of a path. Intuitively, given an FTPL formula and a set of atomic events, the progression function provides either the value of the property, if available, or the rewritten formula otherwise.

Definition 3 (Progression Function for *Events*). *Let $\varepsilon, \varepsilon_1, \varepsilon_2 \in AE$, $e = e_1, e_2 \dots e_m$, a list of FTPL events from AE, and $\theta(i)$ an* event. *The progression function $P : Prop_{FTPL} \times \Theta \rightarrow Prop_{FTPL}$ is inductively defined by:*

[4] For relations involving two components (like *Delegate* or *Parent*) we consider that only the parent component is *aware* of the relation. For the *Binding* relation, only the component owning the *required* (or *client*) interface is *aware* of the binding.

$$\begin{array}{llll}
P(\varepsilon,\theta(i)) = \top \ \textit{if}\ \varepsilon \in \theta(i), \bot \ \textit{otherwise} & ; & P(\bot,\theta(i)) & = \bot \\
P(\varepsilon_1 \vee \varepsilon_2,\theta(i)) = P(\varepsilon_1,\theta(i)) \vee P(\varepsilon_2,\theta(i)) & ; & P(\bot^p,\theta(i)) & = \bot^p \\
P(\neg\varepsilon,\theta(i)) = \neg P(\varepsilon,\theta(i)) & ; & P(\top^p,\theta(i)) & = \top^p \\
P(e,\theta(i)) = \bigvee_{1 \le j \le m} P(e_j,\theta(i)) & ; & P(\top,\theta(i)) & = \top
\end{array}$$

Let us now introduce, in order to establish progression formulae, the $\overline{\mathbf{X}}$-operator that precedes an FTPL property to denote its evaluation at the configuration preceding the current one, i.e., $P(\overline{\mathbf{X}}\xi,\theta(i)) = P(\xi,\theta(i-1))$. We write $\overline{\mathbf{X}}^m\xi$ to denote $\overbrace{\overline{\mathbf{X}}\overline{\mathbf{X}}\ldots\overline{\mathbf{X}}}^{m}\xi$. Also, when $m=0$, $\overline{\mathbf{X}}^m\xi = \xi$.

Because of lack of room, the progression function is not given for every type of FTPL property. Instead, we provide a definition for the **always** trace property (Definition 4), lists of events (Definition 5), and the **before** temporal property (Definition 6). The reader can extrapolate these definitions for the remaining FTPL properties, using the FTPL progressive semantics introduced in [3].

Definition 4 (Progression of the *always* FTPL trace property's evaluation formulae on a (suffix) path). *Let cp be a configuration property and ϕ a trace property of the form $\phi =$* **always** *cp. The progression function P for the* **always** *property on a (suffix) path is defined by:*

$$P(\phi_{\sigma_k},\theta(i)) = \begin{cases} P(cp,\theta(i)) \sqcap \top^p & \textit{for } i = k \\ P(cp,\theta(i)) \sqcap P(\overline{\mathbf{X}}\phi_{\sigma_k},\theta(i)) & \textit{for } i > k \end{cases} \tag{1}$$

Definition 5 (Progression of FTPL list of events properties' evaluation formulae on a (suffix) path). *Let e be a list of FTPL events, the progression function P for FTPL lists of events on a (suffix) path is defined by:*

$$P(e_{\sigma_k},\theta(i)) = \begin{cases} P(e,\theta(i)) & \textit{for } i = k \\ P(e,\theta(i)) \sqcup (\top^p \sqcap P(\overline{\mathbf{X}}e_{\sigma_k},\theta(i))) & \textit{for } i > k \end{cases} \tag{2}$$

Definition 6 (Progression of the *before* FTPL temporal property's evaluation formulae on a (suffix) path). *Let e be a list of FTPL events, trp a trace property, and β a temporal property of the form $\beta =$* **before** *e trp. The progression function P for the* **before** *property on a (suffix) path is defined by:*

$$P(\beta_{\sigma_k},\theta(i)) = \begin{cases} \top^p & \textit{for } i = k \\ F_{\mathcal{B}}(P(e_{\sigma_k},\theta(i)), P(\overline{\mathbf{X}}trp_{\sigma_k},\theta(i)), P(\overline{\mathbf{X}}\beta_{\sigma_k},\theta(i))) & \textit{for } i > k \end{cases} \tag{3}$$

where $F_{\mathcal{B}}$ is based on the FTPL progressive semantics and defined as follows:

$$F_{\mathcal{B}}(\varepsilon,trp,tpp) = \begin{cases} \top^p & \textit{if } \varepsilon = \bot \\ \bot & \textit{if } \varepsilon = \top \wedge trp \in \{\bot,\bot^p\} \\ tpp & \textit{otherwise} \end{cases} \tag{4}$$

Example 1. Let be $\varphi = \textbf{before}\ e\ trp$ where e is an FTPL list of events and trp a trace property. To evaluate φ at the configuration of index $i > 0$ on the suffix path σ_0, let us set $P(e_{\sigma_0}, \theta(i)) = e_{\sigma_0}(i) = \top$ and $P(trp_{\sigma_0}, \theta(i-1)) = trp_{\sigma_0}(i-1) = \bot^p$. Then by Equalities (3) and (4) we have:

$$\begin{aligned} P(\varphi_{\sigma_0}, \theta(i)) &= F_{\mathcal{B}}(P(e_{\sigma_0}, \theta(i)), P(\overline{\mathbf{X}}trp_{\sigma_0}, \theta(i)), P(\overline{\mathbf{X}}\varphi_{\sigma_0}, \theta(i))) \\ &= F_{\mathcal{B}}(P(e_{\sigma_0}, \theta(i)), P(trp_{\sigma_0}, \theta(i-1)), P(\varphi_{\sigma_0}, \theta(i-1))) \\ &= F_{\mathcal{B}}(\top, \bot^p, P(\varphi_{\sigma_0}, \theta(i-1))) \\ &= \bot \end{aligned}$$

In order to perform evaluation in a decentralised manner, we define below the *Normalised Progression Form* (NPF) to describe the point up to which a formula should be developed, using the progression function.

Definition 7 (NPF). *Let φ be an FTPL property and θ an event. A formula $P(\varphi, \theta)$ is in NPF if the $\overline{\mathbf{X}}$-operator only precedes atomic events.*

Theorem 1 (Existence of NPF). *Let φ be an FTPL property and θ an event. Every $P(\varphi, \theta)$ can be rewritten into an equivalent[5] formula in NPF.*

Proof. The proof is by induction on the indexes of the events (i.e., on the trace) using Definitons 4 to 6 (and definitions for the remaining FTPL properties).

Example 2. Let be $\varphi = \textbf{before}\ e\ trp$, $e = a, b$, and $trp = \textbf{always}\ cp$, where a and b are FTPL events s.t. a, b, and $cp \in CP$ are atomic events. The resulting formula in NPF is obtained using Eq. 3.

$$\begin{aligned} P(\varphi_{\sigma_0}, \theta(0)) &= \top^p \\ P(\varphi_{\sigma_0}, \theta(1)) &= F_{\mathcal{B}}(P(e_{\sigma_0}, \theta(1)), P(\overline{\mathbf{X}}trp_{\sigma_0}, \theta(1)), P(\overline{\mathbf{X}}\varphi_{\sigma_0}, \theta(1))) \\ &= F_{\mathcal{B}}(P(e, \theta(1)) \sqcup (\top^p \sqcap P(\overline{\mathbf{X}}e_{\sigma_0}, \theta(1))), P(trp_{\sigma_0}, \theta(0)), P(\varphi_{\sigma_0}, \theta(0))) \\ &= F_{\mathcal{B}}(P(a, \theta(1)) \sqcup P(b, \theta(1)) \sqcup (\top^p \sqcap P(e_{\sigma_0}, \theta(0))), P(cp, \theta(0)) \sqcap \top^p, \top^p) \\ &= F_{\mathcal{B}}(P(a, \theta(1)) \sqcup P(b, \theta(1)) \sqcup (\top^p \sqcap P(e, \theta(0))), P(cp, \theta(0)) \sqcap \top^p, \top^p) \\ &= F_{\mathcal{B}}(P(a, \theta(1)) \sqcup P(b, \theta(1)) \sqcup (\top^p \sqcap (P(a, \theta(0)) \sqcup P(b, \theta(0)))), P(cp, \theta(0)) \sqcap \top^p, \top^p) \\ &= F_{\mathcal{B}}(P(a, \theta(1)) \sqcup P(b, \theta(1)) \sqcup (\top^p \sqcap (P(\overline{\mathbf{X}}a, \theta(1)) \sqcup P(\overline{\mathbf{X}}b, \theta(1)))), P(\overline{\mathbf{X}}cp, \theta(1)) \sqcap \top^p, \top^p) \end{aligned}$$

As in [4] for LTL, a monitor M_j for the component C_j accepts as input an event $\theta(i)$ and FTPL properties. Applying Definition 3 to atomic events could lead to wrong results in a decentralised context. For example, if $\varepsilon \notin \theta(i)$ holds locally for the monitor M_j it could be due to the fact that $\varepsilon \notin AE_j$. The decentralised progression rule should be adapted by taking into account a local set of atomic events. Hence, the progression rule for atomic events preceded by the $\overline{\mathbf{X}}$-operator is given below.

$$P(\overline{\mathbf{X}}^m \varsigma, \theta(i), AE_j) = \begin{cases} \top & \text{if } \varsigma = \varsigma' \text{ for some } \varsigma' \in AE_j \cap \Pi_j(\theta(i-m)), \\ \bot & \text{if } \varsigma = \varsigma' \text{ for some } \varsigma' \in AE_j \setminus \Pi_j(\theta(i-m)), \\ \overline{\mathbf{X}}^{m+1} \varsigma & \text{otherwise.} \end{cases} \tag{5}$$

[5] wrt. the semantics.

We complete the specification of the progression function with the special symbol $\# \notin AE$ for which the progression is defined by $\forall j.P(\#, \theta, AE_j) = \#$. Finally, among different formulae to be evaluated, the notion of urgency allows determining a set of urgent formulae. In a nutshell, the urgency of a formula in NPF is 0 if the formula does not contain any $\overline{\mathbf{X}}$-operator or the value of the greatest exponent of the $\overline{\mathbf{X}}$-operator. Using formulae in NPF, any sub-formula ς following an $\overline{\mathbf{X}}$-operator is atomic ($\exists j.\varsigma \in AE_j$) and can only be evaluated by a single monitor M_j. A formal definition of urgency can be found in [19].

4 Decentralised Evaluation Problem

As FTPL patterns contain conjunctions or disjunctions of properties over components' parameters and relations, the evaluation of temporal patterns in a decentralised manner makes sense. Section 4.1 addresses the temporal pattern decentralised evaluation problem on a reconfiguration path by providing an algorithm for such an evaluation in $\mathbb{B}_4$. Its properties are studied in Sect. 4.2.

4.1 Problem Statement and Local Monitor Algorithm

Let $\hat{\varphi}_{\sigma_k}(s)$ denote the value of φ at configuration of index s on the suffix path σ_k. While considering components with their monitors, because of a decentralised fashion, the evaluation of $\varphi_{\sigma_k}(s)$ by a monitor M_i may be delayed to configuration $\sigma(t)$ with $t > s$, and the progression comes into play. In this case, let ${}_i\varphi^s_{\sigma_k}(t)$ denote the decentralised formula as progressed to the configuration $\sigma(t)$ by M_i, for the evaluation of φ started at configuration $\sigma(s)$. Therefore, we consider the following decision problem.

Temporal Pattern Decentralised Evaluation on a Path (TPDEP)
Input: an FTPL temporal property φ, a suffix path σ_k with $k \geq 0$, a configuration $\sigma(s)$ with $s \geq k$, and a number $n = |\mathcal{M}|$ of monitors.

Output: $i, j < n$, and ${}_i\hat{\varphi}^s_{\sigma_k}(s+j) \in \mathbb{B}_4$, the value of φ at $\sigma(s+j)$ by M_i.

We consider as the basic TPDEP case the situation when only *run* operations occur after the TPDEP problem input, and until an output is returned, communications between monitors being covered by *run* operations.

The idea of a decentralised evaluation is as follows. Similarly to [4], at the configuration $\sigma(t)$, if ${}_i\varphi^s(t)$ cannot be evaluated in $\mathbb{B}_4$, a monitor M_i progresses its current formula ${}_i\varphi^s(t)$ to ${}_i\varphi^s(t+1) = P({}_i\varphi^s(t), \theta(t), AE_i)$ and sends it to a monitor that can evaluate its most urgent sub-formula. After ${}_i\varphi^s(t+1)$ is sent, M_i sets ${}_i\varphi^s(t+1) = \#$. When M_i receives one or more formulae from others monitors, each of them is added to the current formula using the $\sqcap$ operator.

Unlike [4], where LTL decentralised monitoring determines the *steady* value of a property in $\mathbb{B}_2$, our decentralised method allows values of FTPL properties in $\mathbb{B}_4$ to vary at different configurations, depending notably on the property scopes and on external events. To this end, a result in $\mathbb{B}_4$ obtained by a monitor is broadcast to other monitors, allowing them to maintain a complete but

```
1  (*LDMon*)                                              14 |  | IF (Prop(φ)∩AE_i ≠ ∅ ∧ _iφ^s_{σ_k}(t) ≠ #) DO
2  Input                                                  15 |  | | send(_iφ^s_{σ_k}(t))
3     i      (*current monitor index*)                    16 |  | | _iφ^s_{σ_k}(t) = #
4     AE_i   (*current monitor atomic events*)            17 |  | FI
5     s      (*input configuration index*)                18 |  | receive({_jφ^s_{σ_k}(t)}_{j≠i})
6     φ      (*FTPL property to evaluate*)                19 |  | _iφ^s_{σ_k}(t) := _iφ^s_{σ_k}(t) ⊓ ⨅_{j≠i} _jφ^s_{σ_k}(t)
7     σ_k    (*suffix path*)                              20 | ENDWHILE
8  Variables                                              21 | IF (∀j ≠ i._jφ̂^s_{σ_k}(t) ∉ 𝔹_4) DO
9     t := s : integer                                    22 | | broadcast(_iφ^s_{σ_k}(t))
10 Begin                                                  23 | FI
11 | WHILE (_iφ̂^s_{σ_k}(t) ∉ 𝔹_4) DO                      24 | RETURN _iφ̂^s_{σ_k}(t)
12 | | _iφ^s_{σ_k}(t+1) := P(_iφ^s_{σ_k}(t), θ_i(t), AE_i) 25 End
13 | | t := t + 1
```

Fig. 2. Algorithm LDMon

bounded history that can be used to invoke the TPDEP problem at the following configurations.

To answer the TPDEP problem, we propose the LDMon algorithm displayed in Fig. 2. It takes as input the index i of the current monitor, its set AE_i of atomic events, the index s of the current configuration, an FTPL temporal property φ to be evaluated, and the index k of the suffix path on which φ is supposed to be evaluated. An integer variable t indicates the index of the current configuration as it evolves. The algorithm broadcasts to all monitors, as soon as it is determined, the result of the evaluation of φ in $\mathbb{B}_4$. We chose this method to transmit the results because we prefer to focus on the feasibility of a decentralised evaluation of temporal patterns and we consider that the transmission of result is a related issue outside of the scope of this paper.

Three functions are used in this algorithm: (a) `send`(φ), sends φ (as well as its sub-formulae evaluated at the current configuration) to monitor M_j (different from the current monitor) where ψ is the most urgent sub-formula[6] such that $Prop(\psi) \subseteq AE_j$ holds, with $Prop : Prop_{FTPL} \rightarrow 2^{AE}$ yielding the set of events of an FTPL formula; (b) `receive`($\{\varphi, \dots\}$), receives formulae sent (and broadcast) by other monitors; and (c) `broadcast`(φ), broadcasts φ to all other monitors.

As long as an evaluation of φ in $\mathbb{B}_4$ is not obtained (line 11), the LDMon algorithm loops in the following way: the evaluation formula is progressed to the next configuration (line 12) and the current configuration index t is incremented (line 13). If at least one event of the current formula belongs to the set of atomic events AE_i ($Prop(\varphi) \cap AE_i \neq \varnothing$) and if no progressed formula was sent (or if such a formula was sent and at least one from another monitor was received) at the previous configuration (${}_i\varphi^s_{\sigma_k}(t) \neq \#$), the progressed formula is sent to the monitor that can solve its most urgent sub-formula (line 15) and is set to # (line 16). Progressed formulae (and broadcast results) from other monitors are received (line 18) and are combined to the local formula using the $\sqcap$-operator

[6] In the case where there are two or more equally urgent formulae, φ is sent to a monitor determined by an arbitrary order with the function $Mon : \mathcal{M} \times 2^{AE} \rightarrow \mathcal{M}$. $Mon(M_i, AE') = M_{j_{min}}$ s.t $j_{min} = min(j \in [1, n] \backslash \{i\} | AE' \cap AE_j \neq \varnothing)$.

(line 19). If the result is not in $\mathbb{B}_4$, the loop continues, otherwise if the result of the formula has not already been provided by another monitor (line 21), the result is broadcast (line 22) and returned (line 24).

4.2 Correctness, Uniqueness, and Termination

In this section several properties of the `LDMon` algorithm are studied. Proposition 1, below, guarantees that the `LDMon` algorithm provides an output within a finite number of configurations, communications being covered by *run* operations.

Proposition 1 (Existence). *Let $\varphi \in Prop_{FTPL}$, σ_k a suffix path, $k \geq 0$. For a given configuration $\sigma(s)$ with $s \geq k$, when using a number $n = |\mathcal{M}|$ of monitors, the* `LDMon` *algorithm provides an output such that $\exists i, j. i, j < n \wedge {}_i\hat{\varphi}^s_{\sigma_k}(s+j) \in \mathbb{B}_4$.*

Proof. (*Sketch.*) Let $M_0, M_1, \ldots, M_{n-1}$ be n monitors. At a given configuration of index s, if one of the monitors $M_i \in \mathcal{M}$ is able to evaluate its formula in $\mathbb{B}_4$, the proposition holds with $j = 0$. Otherwise, each monitor M_i $(0 \leq i < n)$ progresses its formula ${}_i\varphi^s_{\sigma_k}(s)$ into ${}_i\varphi^s_{\sigma_k}(s+1)$ and sends it to another monitor, according to Mon, able to answer its most urgent sub-formula.

We assume that ${}_{i_1}\varphi^s_{\sigma_k}(s+1)$ is sent to the monitor $M_{i_2 \neq i_1}$. At the next configuration of index $s+1$, the monitor M_{i_2} receives ${}_{i_1}\varphi^s_{\sigma_k}(s+1)$ and combines it with ${}_{i_2}\varphi^s_{\sigma_k}(s+1)$ as well as other formulae (if any) received from other monitors using the $\sqcap$-operator. If one of these formulae (or a sufficient number of subformulae) can be evaluated in $\mathbb{B}_4$, the proposition holds with $j = 1$ and ${}_i\hat{\varphi}^s_{\sigma_k}(s+1)$. Otherwise, each monitor M_i progresses the formula ${}_i\varphi^s_{\sigma_k}(s+1)$ into ${}_i\varphi^s_{\sigma_k}(s+2)$ and sends it to another monitor according to Mon which is able to answer its most urgent sub-formula.

We assume that ${}_{i_2}\varphi^s_{\sigma_k}(s+2)$ is sent to the monitor M_{i_3} with $i_3 \neq i_2$. Also $i_3 \neq i_1$ because previously, all sub-formulae of ${}_{i_1}\varphi^s_{\sigma_k}(s+1)$ that could be solved using the set of atomic events AE_{i_1} were already solved. This way, the problem is reduced from n to $n-1$ monitors. Since for a single monitor the output of the algorithm is $\hat{\varphi}_{\sigma_k}(s)$ with $j = 0$, we can infer that for n monitors, there is at least one monitor M_{i_0} such that ${}_{i_0}\hat{\varphi}^s_{\sigma_k}(s+j) \in \mathbb{B}_4$ with $j < n$. □

As explained before, when evaluating $\varphi_{\sigma_k}(s)$, the formula ${}_i\varphi^s_{\sigma_k}(t)$ at configuration of index t by M_i either has a result ${}_i\hat{\varphi}^s_{\sigma_k}(t) \in \mathbb{B}_4$ or progresses to #. The latter is written ${}_i\hat{\varphi}^s_{\sigma_k}(t) = \#$. Thus ${}_i\hat{\varphi}^s_{\sigma_k}(t) \in \mathbb{B}_5$.

Theorem 2 (Semantic Correctness). ${}_i\hat{\varphi}^s_{\sigma_k}(t) \neq \# \Leftrightarrow {}_i\hat{\varphi}^s_{\sigma_k}(t) = \hat{\varphi}_{\sigma_k}(s)$.

Proof. (*Sketch.*)

$\Rightarrow$ If ${}_i\hat{\varphi}^s_{\sigma_k}(t) \neq \#$, a result has been obtained in $\mathbb{B}_4$, otherwise ${}_i\hat{\varphi}^s_{\sigma_k}(t)$ would equal #. Therefore, we only have to verify that the progression function of Definitions 3 to 6 (and definitions for the remaining FTPL properties) matches the FTPL semantics in $\mathbb{B}_4$ as defined in [3]. It is done by induction on the path length.

$\Leftarrow$ ${}_i\hat{\varphi}^s_{\sigma_k}(t) = \hat{\varphi}_{\sigma_k}(s) \Rightarrow {}_i\hat{\varphi}^s_{\sigma_k}(t) \in \mathbb{B}_4 \Rightarrow {}_i\hat{\varphi}^s_{\sigma_k}(t) \neq \#$.

Corollary 1 (Uniqueness). *If* ${}_i\hat{\varphi}^s_{\sigma_k}(t) \neq \#$ *and* ${}_j\hat{\varphi}^s_{\sigma_k}(t) \neq \#$ *for* $i \neq j$, *then* ${}_i\hat{\varphi}^s_{\sigma_k}(t) = {}_j\hat{\varphi}^s_{\sigma_k}(t)$.

Corollary 2 (Generalised Uniqueness). *Let be* ${}_S\varphi^s_{\sigma_k}(t) = \underset{i \in S}{\sqcap} {}_i\varphi^s_{\sigma_k}(t)$ *for* $S \subseteq [1, n]$. *If* ${}_S\hat{\varphi}^s_{\sigma_k}(t) \neq \#$ *then for all* $j \in S$, ${}_j\hat{\varphi}^s_{\sigma_k}(t) \neq \#$ *implies* ${}_j\hat{\varphi}^s_{\sigma_k}(t) = {}_S\hat{\varphi}^s_{\sigma_k}(t)$.

Corollary 2 allows a monitor to simplify the combination of formulae with the operator $\sqcap$. For a given property, a conjunction in $\mathbb{B}_4$ of formulae received from other monitors with the formula of the current monitor can be replaced by any of these formulae provided that its value is different from #.

Example 3. Let us consider again $\varphi =$ **before** $e\ trp$. Let A,B, and C be the components with their respective monitors M_A, M_B, and M_C such that ${}_i\varphi^s(t) = F_{\mathcal{B}}({}_ie^s(t), {}_itrp^{s-1}(t), {}_i\varphi^{s-1}(t))$ for $i \in \{A, B, C\}$ (Definition 6). Let us assume $\varphi(s) = F_{\mathcal{B}}(e(s), trp(s-1), \varphi(s-1))$, with $\varphi(s)$, $e(s)$, $trp(s-1)$, and $\varphi(s-1)$ being evaluated in $\mathbb{B}_4$. By Corollary 2, $e(s) = {}_Ae^s(t) \sqcap {}_Be^s(t) \sqcap {}_Ce^s(t)$ (resp. $trp(s-1) = {}_Atrp^{s-1}(t) \sqcap {}_Btrp^{s-1}(t) \sqcap {}_Ctrp^{s-1}(t)$, $\varphi(s-1) = {}_A\varphi^{s-1}(t) \sqcap {}_B\varphi^{s-1}(t) \sqcap {}_C\varphi^{s-1}(t)$) if it exists at least one i such that the value of ${}_ie^t(s)$ (resp. ${}_itrp^{s-1}(t)$, ${}_i\varphi^{s-1}(t)$) is in $\mathbb{B}_4$; in this case, ${}_ie^s(t) = e(s)$ (resp. ${}_itrp^{s-1}(t) = trp(s-1)$, ${}_i\varphi^{s-1}(t) = \varphi(s-1)$).

For example, if ${}_A\varphi^s(t) = F_{\mathcal{B}}(\top, \phi, {}_A\varphi^{s-1}(t))$, ${}_B\varphi^s(t) = F_{\mathcal{B}}(\varepsilon, \top^p, {}_B\varphi^{s-1}(t))$, and ${}_C\varphi^s(t) = F_{\mathcal{B}}(\epsilon, \psi, \top^p)$, with ϕ, ${}_A\varphi^{s-1}(t)$, ε, ${}_B\varphi^{s-1}(t)$, ϵ, and ψ not being evaluated in $\mathbb{B}_4$. It implies ${}_Ae^s(t) = e(s) = \top$ (resp. ${}_Btrp^{s-1}(t) = trp(s-1) = \top^p$, ${}_C\varphi^{s-1}(t) = \varphi(s-1) = \top^p$) and $\varphi(s) = F_{\mathcal{B}}(\top, \top^p, \top^p) = \top^p$.

Proposition 2 (Correctness and Uniqueness). *The output provided by the* `LDMon` *algorithm answers the TPDEP problem. For a given configuration* $\sigma(s)$, *this answer is unique.*

Proof. (*Sketch.*) By Proposition 1 `LDMon` provides an output ${}_i\hat{\varphi}^s_{\sigma_k}(s + j)$ for at least one monitor M_i, within a finite number j of configurations. By Theorem 2 this output answers the TPDEP problem. Furthermore Corollary 1 establishes that for any i_0, if ${}_{i_0}\hat{\varphi}^s_{\sigma_k}(s + j)$ is the output of the `LDMon` algorithm for the monitor M_{i_0} then ${}_{i_0}\hat{\varphi}^s_{\sigma_k}(s + j) = {}_i\hat{\varphi}^s_{\sigma_k}(s + j)$. □

Proposition 3 (Termination). *The* `LDMon` *algorithm always terminates, either at the configuration when an output is provided or at the next one. Furthermore, the number of configurations needed to reach a result is at most* $|\mathcal{M}|$.

Proof. (sketch) Propositions 1 and 2 establish that the `LDMon` algorithm terminates and answers the TPDEP problem for at least one monitor M_i after a finite number of reconfigurations $j < |\mathcal{M}|$. Such monitor M_i broadcasts the result to all other monitors before finishing (line 22 of the `LDMon` algorithm, Fig. 2). This enables any monitor for which the `LDMon` algorithm did not finish at configuration

$s + j$ to receive the result of the broadcast and to finish its instance of the `LDMon` algorithm at configuration $s + j + 1 \leq s + |\mathcal{M}|$. □

In general, decentralised algorithms tend to be very hard for creating a consensus and moreover they require significant communication overhead. Let us emphasize the fact that Proposition 2 guarantees the correctness and uniqueness of a result, which implies such a consensus. As a consequence of Propositions 2 and 3 adaptation policies relying on the decentralised evaluation of FTPL temporal properties can be applied to component-based systems for their dynamic reconfiguration at runtime.

Let us now discuss communication overhead. We consider a component-based system of N components reporting their status in $\mathbb{B}_4$ to a central controller at each configuration as described for in [3]. In the centralised context, thanks to the progressive semantics, the evaluation of a given FTPL property φ would mean that N messages should be sent to conclude in $\mathbb{B}_4$. With the decentralised approach, assuming that atomic events of φ would be distributed among n components ($n \leq N$), we would need, at most, $n^2 - 1$ messages to evaluate φ.

This means that to evaluate a formula involving $n = 10$ components of a component-based system of $N = 100$ components, in the worst case the decentralised fashion would need 99 messages versus 100 for the centralised approach to evaluate φ which is a ratio of 99 %. If, however, the total number N of components of the system is much greater than the number n of components involved in the evaluation of φ, the communication overhead ratio can be even lower (e.g., 9.9 % for $N = 1000$). Reciprocally, if a great proportion of the system is involved in the property to evaluate, the centralised method would lead to better results. Let q be such a proportion, i.e., $n = qN$, the communication overhead ratio is $Nq^2 - 1/N$.

This is different from the result in [4] where the decentralised algorithm outperforms its centralised counterpart by a proportion of 1 to 4 in terms of communication overhead, to conclude in $\mathbb{B}_2$. Such a difference is due to the fact that in our case, as soon as a property is evaluated in $\mathbb{B}_4$ for a given configuration of the path, another evaluation is initiated for another configuration. Nevertheless, we have better results while monitoring only components concerned with the temporal property, that can be determined syntactically. To sum up, our approach is suitable for systems with a large number of components when the FTPL property to evaluate involves a small proportion of them.

5 Implementation and Experiment

This section describes how the `LDMon` algorithm has been implemented within the *GROOVE* graph transformation tool [5]. This implementation is then used to experiment with a case study.

5.1 Implementing with *GROOVE*

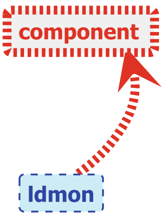


Fig. 3. Rule `remove OrphanMon` (Color figure online)

GROOVE uses simple graphs for modelling the structure of object-oriented systems at design-time, compile-time, and runtime. Graphs are made of nodes and edges that can be labelled. Graph transformations provide a basis for model transformation or for operational semantics of systems. Graphs are transformed by rules consisting of (a) patterns that must be present (resp. absent) for the rule to apply, (b) elements (nodes and edges) to be added (resp.deleted) from the graph, and (c) pairs of nodes to be merged. Colour and shape coding allow these rules to be easily represented. For example, our implementation uses the graph rule `removeOrphanMon` represented Fig. 3 that can be interpreted as follows: (a) The red (dashed fat) "embargo" elements, representing a node of type *component* and an edge defining a monitoring relation between monitors, of type *ldmon*, and *components*, must be absent, (b) the blue (dashed thin) "eraser" element, representing a node of type *ldmon*, must be present, and (c) if both conditions are satisfied, the blue (dashed thin) element is deleted. This means that if a monitor of type *ldmon* is not monitoring a *component*, the monitor node, *ldmon*, must be deleted. The reader interested in *GROOVE* is referred to [5].

Our implementation uses the *GROOVE* typed mode to guarantee that all graphs are well-typed. It consists of generic types and graph rules that can manage assigned priorities in such a way that a rule is applied only if no rule of higher priority matches the current graph. The input is a graph containing an FTPL formula and a component-based system, both represented using the model presented in Sect. 2. Figure 4 shows a screenshot of *GROOVE* displaying, in the main panel, a graph modelling the *location* component-based system used in the case study below. *Components* are represented in blue, *Required* (resp. *Provided) Interfaces* in magenta (resp. red), *Parameters* in black, and both *ITypes* and *PTypes* in grey. The top left panel shows graph rules ordered by priority, whereas the bottom left panel contains *GROOVE* types.

5.2 Case Study

In this section we illustrate the `LDMon` algorithm with an example of a location composite component, and afterwards we provide several details on its implementation in *GROOVE*. The location system is made up of different positioning systems, like GPS or Wi-Fi, a merger and a controller. Thanks to adaptation policies with temporal patterns, the location composite component can be modified to use either GPS or Wi-Fi positioning systems, depending on some properties, such as available energy, occurrences of indoor/outdoor positioning external events, etc. For example, when the level of energy is low, if the vehicle is in a tunnel where there is no GPS signal, it would be useful to remove the GPS

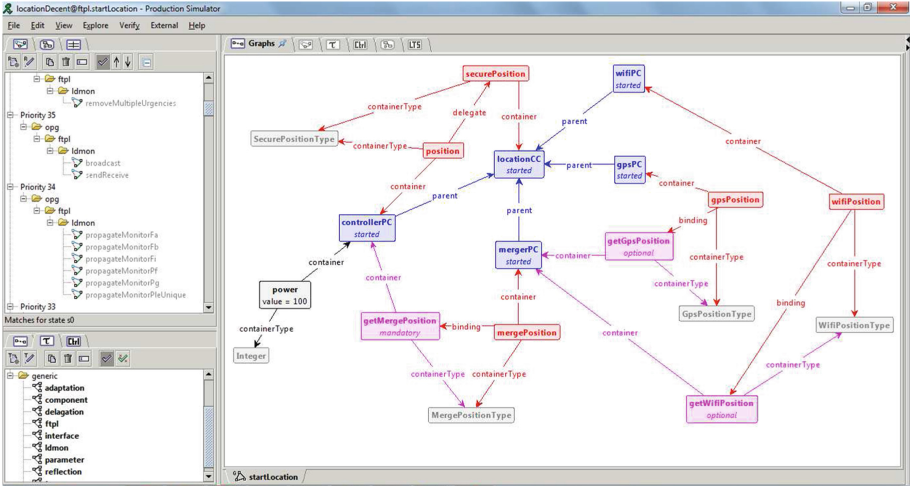


Fig. 4. Model of the location component-based system displayed with *GROOVE*

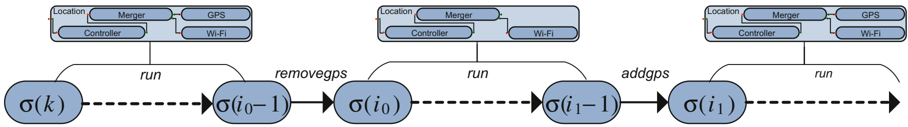


Fig. 5. Representation of the suffix configuration path σ_k

component (cf. Fig. 5). To save energy, this component may not be added back before the level of energy reaches an acceptable value.

This example has been fully implemented with *GROOVE* together with adaptation policies. Let G be the *GROOVE* graph representing this example. Let us consider the FTPL temporal property $\varphi =$ **after** `removegps` **normal** (**eventually** ($power \geq 33$) **before** `addgps` **normal**), which can be written as $\varphi =$ **after** e_0 ϕ, with $e_0 =$ `removegps` **normal**, $\phi = trp$ **before** e_1, $trp =$ **eventually** cp, $e_1 =$ `addgps` **normal**, and $cp = (power \geq 33)$. Intuitively, φ represents the requirement "After the GPS component has been removed, the level of energy has to be greater than 33 % before this component is added back". Figure 6 shows how φ is represented in our implementation.

Let M_c, M_m, M_g, and M_w be four monitors pertaining respectively to the controller, merger, GPS, and Wi-Fi components. Monitor Mc has access to the value of the configuration property `power_ge_33` ($\top$ if $power \geq 33$, or $\bot$ otherwise) while M_m is aware of the values of `addgps_normal` (resp. `removegps_normal`) which are $\top$ at the configuration following the addition (resp. removal) of the GPS component, or $\bot$ otherwise. Since monitors, M_g and M_w do not have access to any atomic event having an influence on the evaluation of φ (i.e., $Prop(\varphi) \cap AE_g = Prop(\varphi) \cap AE_w = \varnothing$), M_g and M_w do not send messages, which has a beneficial

effect on the communication overhead. In our implementation, each monitor is a subgraph of G containing the monitored component via an edge named *monitor*. Communications between monitors are represented by edges named *sentreceived* and *broadcast*. Recall that in the model, communications between monitors are covered by *run* operations as they do not directly affect the system's architecture.

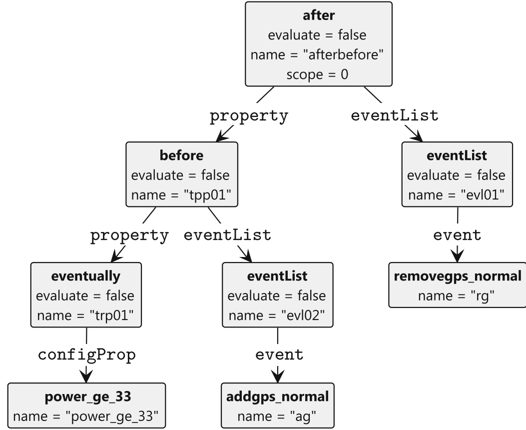


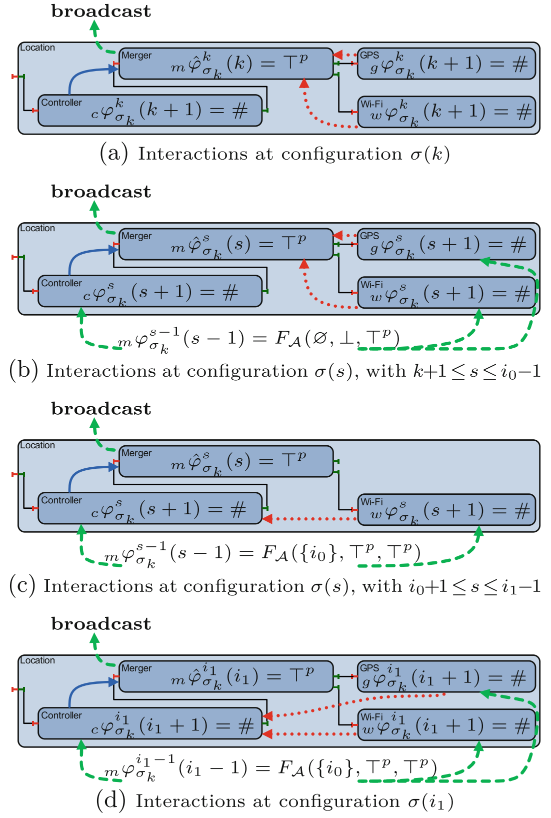
Fig. 6. Representation of the φ FTPL property

Let us consider a reconfiguration path σ representing the sequences of configurations of the location composite component where the transitions between configurations are reconfiguration operations. In the suffix path σ_k displayed in Fig. 5, we suppose that all the reconfiguration operations are *run*, except between $\sigma(i_0 - 1)$ and $\sigma(i_0)$ (resp. $\sigma(i_1 - 1)$ and $\sigma(i_1)$), where it is `removegps` (resp. `addgps`). During runtime, an adaptation controller—in charge of the application of adaptation policies—needs to evaluate FTPL properties. To do so, the adaptation controller uses the `LDMon` algorithm to evaluate φ. When a result is returned by a monitor, the most suitable adaptation policy is applied, and the algorithm is used again to evaluate φ at the following configuration, where it may vary because of the scope, for example. In the following, we describe how, at each configuration of index s $(k \leq s \leq i_1)$ the adaptation controller requests the evaluation of φ to the monitors using the `LDMon` algorithm, and receives the answer at the same configuration $\sigma(s)$. In Fig. 7 green (dashed) arrows represent broadcast communications, blue (plain) arrows represent formulae being sent, and red (dotted) arrows indicate that (a) the destination component is able to solve the most urgent sub-formula of the source component and (b) no communication is made between these components. Because neither M_g nor M_w has access to atomic events impacting their formulae, they do not send any message during the run described below.

At configuration $\sigma(k)$, since M_m can evaluate $e_0 = \bot$, by definition of the **after** FTPL property, ${}_m\hat{\varphi}^k_{\sigma_k}(k) = \top^p$ is established and broadcast. Other monitors progress their formulae and determine that the most urgent sub-formula can be solved by M_m (Fig. 7a); consequently, M_c sends its formulae[7] to M_m.

[7] The formula to evaluate as well as its sub-formulae evaluated at the current state.

At every configuration $\sigma(s)$ *for* $k+1 \leq s \leq i_0-1$, since e_0 does not occur, the decentralised evaluation consists in evaluating φ by M_m that returns and broadcasts the result. Other monitors receive the result from the previous configuration broadcast by M_m[8]. They also progress their current formulae, which cause M_c to send its formulae to M_m. This is diplayed in Fig. 7b, where $F_{\mathcal{A}}$ represents the FTPL temporal property **after** in the same way $F_{\mathcal{B}}$ does for the **before** property in Definition 6. *At configuration* $\sigma(i_0)$, the event e_0 = `removegps` **normal**, signifying the GPS normal removal, occurs. The M_m monitor, being aware of this occurrence, evaluates φ: ${}_m\hat{\varphi}^{i_0}_{\sigma_k}(i_0) = {}_m\hat{\phi}^{i_0}_{\sigma_{i_0}}(i_0) = \top^p$ because the "**before**" FTPL pattern is defined to be $\top^p$ at the first configuration of its scope. The result is then retuned and broadcast. In the meantime, M_c and M_w receive the result broadcast at the previous configuration and M_c sends its formulae to M_m.



(a) Interactions at configuration $\sigma(k)$

(b) Interactions at configuration $\sigma(s)$, with $k+1 \leq s \leq i_0-1$

(c) Interactions at configuration $\sigma(s)$, with $i_0+1 \leq s \leq i_1-1$

(d) Interactions at configuration $\sigma(i_1)$

Fig. 7. Interactions between monitors (Color figure online)

At configuration $\sigma(s)$ *for* $i_0+1 \leq s \leq i_1-1$, because e_0 occurred once, M_m computes ${}_m\hat{\varphi}^s_{\sigma_k}(s) = {}_m\hat{\phi}^s_{\sigma_{i_0}}(s) = \top^p$, since $\phi = trp$ **before** e_1 and e_1 has not yet occurred; the result is then returned and broadcast. M_c and M_w receive the result broadcast at the previous configuration which contains, as a subformula, the information that e_0 occurred at configuration $\sigma(i_0)$. The formula progressed by M_c contains ${}_c\phi^s_{\sigma_{i_0}}(s+1) = F_{\mathcal{B}}(\overline{\mathbf{X}}e_1, \hat{trp}^{s-1}_{\sigma_{i_0}}(s), \top^p)$. We suppose that there is a configuration $\sigma(s')$ s.t. $s' > i_0$, where the power rises over 33 %, i.e., $cp = (power \geq 33) = \top$ and then $\hat{trp}^s_{\sigma_{i_0}}(s) = cp \sqcup \hat{trp}^{s-1}_{\sigma_{i_0}}(s) = \top$ for $s \geq s'$. In this case, the set of formulae M_c sends to M_m (Fig. 7c) contains $F_{\mathcal{B}}(\overline{\mathbf{X}}e_1, \top, \top^p)$ and $trp^s_{\sigma_{i_0}}(s)$.

At configuration $\sigma(i_1)$, e_1 = `addgps` **terminates** just occurred. We assume that the reconfiguration terminated normally and that the GPS component was

[8] This allows all monitors to keep a history of $|\mathcal{M}|+1$ configurations.

Table 1. Graph rules used at configuration $\sigma(s)$

Index s of configuration	Number of graph rules	Reconfiguration	Part of formula to be evaluated
$s = k$	85		**after** `removegps` **normal** ...
$k+1 \leq s \leq i_0 - 1$	111 − 162		**after** `removegps` **normal** ...
$s = i_0$	237	`removegps`	... **before** `addgps` **normal**
$i_0 + 1 \leq s \leq i_1 - 1$	149		... **before** `addgps` **normal**
$s = i_1$	253	`addgps`	... **eventually** ($power \geq 33$) ...

added. M_c, M_g, and M_w receive the result broadcast at the previous configuration. M_c and M_w behave in a way similar than above at configuration $\sigma(s)$ with $i_0 + 1 \leq s \leq i_1 - 1$, whereas M_g behaves like M_w. Finally, M_m evaluates its formula to ${}_m\hat{\varphi}^{i_1}_{\sigma_k}(i_1) = {}_m\hat{\phi}^{i_1}_{\sigma_{i_0}}(i_1) = F_{\mathcal{B}}(\top, \overline{\mathbf{X}}trp_{\sigma_{i_0}}, \top^p) = \top^p$ using the fact that the sub-formula $\overline{\mathbf{X}}trp_{\sigma_{i_0}}$ was sent by M_c at the previous configuration. This result answers correctly the TPDEP.

Back to the implementation, Table 1 gives information on the *GROOVE* graph rules for the case study. The columns show, from left to right, the possible values of the index of the considered configurations, the number of graph rules used, the reconfiguration occurring (if any), and the part of the FTPL formula that must be evaluated to obtain a result in $\mathbb{B}_4$. At configuration $\sigma(k)$ 85 rules are used, rules concerning the evaluation of FTPL events are the ones used the most; as long as the event `removegps` **normal** has not occurred yet, only the evaluation of the part "**after** `removegps` **normal**..." of the formula is needed to obtain a result. At configuration $\sigma(s)$, with $k + 1 \leq s \leq i_0 - 1$, from 111 to 162 graph rules are used, depending of the length of the history being built at the beginning of the run; once the length of history has reach its maximum, i.e., $|\mathcal{M}| + 1$, the most used graph rules are the ones designed to clear outdated history. At configuration $\sigma(i_0)$, the reconfiguration `removegps` occurs, then as long as the event `addgps` **normal** has not occurred yet, only the evaluation of the part "...**before** `addgps` **normal**" of the formula is needed to obtain a result; 237 graph rules are used, most of them doing a cleaning of the elements of the subgraph representing the monitor of the GPS component being removed. At configuration $\sigma(s)$, with $k + 1 \leq s \leq i_0 - 1$, 149 graph rules are used, mainly to clear outdated history. At configuration $\sigma(i_1)$, the reconfiguration `addgps` occurs, then only the evaluation, at the previous configuration, of the part "...**eventually** ($power \geq 33$)..." of the formula is needed to obtain a result; 253 graph rules are used, mainly to clear outdated history and to update the scope of the property.

6 Conclusion

This paper has addressed the decentralised evaluation problem for linear temporal patterns on reconfiguration paths of component-based systems. To this end, we have proposed a specific progressive semantics of temporal patterns, and an

algorithm for their decentralised evaluation using monitors associated with components. We have shown that when reached, the decentralised evaluation results coincide with the results obtained by the centralised evaluation of temporal patterns at runtime. We have described the implementation with *GROOVE* and its application to a location composite component.

In this paper, for the sake of readability, monitors only deal with a single FTPL property. To evaluate several FTPL formulae, we can either use a single monitor (per component) dealing with all the formulae, as herein described, or a monitor per formula of interest. Depending on the context, each method can have its own advantages and drawbacks.

In the case of the removal of a component, the corresponding monitor terminates and is removed. Thanks to the adaptation policies' controller, this should not influence any ongoing temporal pattern evaluation. When a component is added, its monitor starts with a blank history. Furthermore, when a monitored primitive component is replaced with a composite component whose sub-components contain (among other) the same parameters as the original component, the monitor shall keep working seamlessly. Since no additional monitor is added, this mechanism allows us to mitigate the communication overhead that could be incurred by the increase of the number of components.

As a future work, we intend to extend the analysis of the TPDEP problem to the case when several reconfiguration operations occur. It would be possible when reconfigurations lead to configurations whose atomic events do not interfere with the evaluation of the temporal property of interest (the TPDEP input). In this case the adaptation controller can authorize reconfigurations of independent parts of the component-based system. On the implementation side, we plan to exploit the decentralised evaluation method for the implementation handling the adaptation policies. The overall goal is to exploit its results to apply adaptation policies to the component-based system under scrutiny at runtime. So far we have considered the components having monitors to be all on the same architectural level, i.e., they all are siblings. As a future work, we plan to *delegate* part of the monitoring of composite components to their subcomponents.

References

1. de Lemos, R., et al.: Software engineering for self-adaptive systems: a second research roadmap. In: de Lemos, R., Giese, H., Müller, H.A., Shaw, M. (eds.) Software Engineering for Self-Adaptive Systems. LNCS, vol. 7475, pp. 1–32. Springer, Heidelberg (2013)
2. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Software Engineering, pp. 411–420 (1999)
3. Kouchnarenko, O., Weber, J.-F.: Adapting component-based systems at runtime via policies with temporal patterns. In: Fiadeiro, J.L., Liu, Z., Xue, J. (eds.) FACS 2013. LNCS, vol. 8348, pp. 234–253. Springer, Heidelberg (2014)
4. Bauer, A., Falcone, Y.: Decentralised LTL monitoring. In: Giannakopoulou, D., Méry, D. (eds.) FM 2012. LNCS, vol. 7436, pp. 85–100. Springer, Heidelberg (2012)
5. Ghamarian, A.H., de Mol, M., Rensink, A., Zambon, E., Zimakova, M.: Modelling and analysis using GROOVE. Int. J. Softw. Tools Technol. Trans. **14**, 15–40 (2012)

6. Kupferman, O., Vardi, M.Y., Wolper, P.: Module checking. Inf. Comput. **164**, 322–344 (2001)
7. Bozzelli, L., Murano, A., Peron, A.: Pushdown module checking. In: Sutcliffe, G., Voronkov, A. (eds.) LPAR 2005. LNCS (LNAI), vol. 3835, pp. 504–518. Springer, Heidelberg (2005)
8. Aminof, B., Legay, A., Murano, A., Serre, O., Vardi, M.Y.: Pushdown module checking with imperfect information. Inf. Comput. **223**, 1–17 (2013)
9. Bauer, A., Leucker, M., Schallhart, C.: Comparing LTL semantics for runtime verification. Int. J. Logic Comput. **20**, 651–674 (2010)
10. Kim, M., Lee, I., Shin, J., Sokolsky, O., et al.: Monitoring, checking, and steering of real-time systems. ENTCS **70**, 95–111 (2002)
11. Bruneton, E., Coupaye, T., Leclercq, M., Quéma, V., Stefani, J.B.: The fractal component model and its support in java. Softw. Pract. Exper. **36**, 1257–1284 (2006)
12. Schneider, S., Treharne, H.: CSP theorems for communicating B machines. Formal Asp. Comput. **17**, 390–422 (2005)
13. Seinturier, L., Merle, P., Rouvoy, R., Romero, D., Schiavoni, V., Stefani, J.B.: A component-based middleware platform for reconfigurable service-oriented architectures. Softw. Practice Experience **42**, 559–583 (2012)
14. Garavel, H., Mateescu, R., Serwe, W.: Large-scale distributed verification using cadp: Beyond clusters to grids. Electr. Notes Theor. Comput. Sci. **296**, 145–161 (2013)
15. Dormoy, J., Kouchnarenko, O., Lanoix, A.: Runtime verification of temporal patterns for dynamic reconfigurations of components. In: Arbab, F., Ölveczky, P.C. (eds.) FACS 2011. LNCS, vol. 7253, pp. 115–132. Springer, Heidelberg (2012)
16. Hamilton, A.G.: Logic for Mathematicians. Cambridge University Press, Cambridge (1978)
17. Dormoy, J., Kouchnarenko, O., Lanoix, A.: When structural refinement of components keeps temporal properties over reconfigurations. In: Giannakopoulou, D., Méry, D. (eds.) FM 2012. LNCS, vol. 7436, pp. 171–186. Springer, Heidelberg (2012)
18. Dormoy, J., Kouchnarenko, O., Lanoix, A.: Using temporal logic for dynamic reconfigurations of components. In: Barbosa, L.S., Lumpe, M. (eds.) FACS 2010. LNCS, vol. 6921, pp. 200–217. Springer, Heidelberg (2012)
19. Kouchnarenko, O., Weber, J.F.: Decentralised Evaluation of Temporal Patterns over Component-based Systems at Runtime (2014). Long version of the present paper available at: http://hal.archives-ouvertes.fr/hal-01044639