Towards aggregate monitoring of spatio-temporal properties

Towards Aggregate Monitoring of Spatio-temporal Properties

Giorgio Audrito
Dipartimento di Informatica,
Università degli Studi di Torino
Turin, Italy
giorgio.audrito@unito.it

Gianluca Torta
Dipartimento di Informatica,
Università degli Studi di Torino
Turin, Italy
gianluca.torta@unito.it

ABSTRACT

As the cost of computing devices continues to decrease, swarms of low-end intelligent devices become a more interesting solution for safety-critical applications. The safe execution of such systems, however, usually requires mechanisms ensuring that relevant global properties, expressed as logical formulas, are satisfied. These formulas need to capture properties of the system evolution in time, and of its distribution in space, thus requiring a mix of spatial and temporal logic modalities. Furthermore, in scenarios where access to the cloud might not be available, monitoring their validity should be performed autonomously by the distributed system itself.

Previous works show that through the aggregate computing approach, and targeting the field calculus language, automatic translations of spatial or temporal logic formulas into distributed decentralized monitors are possible. However, the definition and translation of properties mixing space and time has not been considered so far. In this paper, we start the investigation on integrating space and time modalities through examples, outlining a roadmap for a fully-fledged distributed monitoring of space-time properties.

CCS CONCEPTS

• **Computing methodologies** → **Self-organization**; • **Computer systems organization** → *Embedded and cyber-physical systems*; • **Theory of computation** → **Modal and temporal logics**.

KEYWORDS

runtime verification, aggregate computing, spatial logic, temporal logic

ACM Reference Format:
Giorgio Audrito and Gianluca Torta. 2021. Towards Aggregate Monitoring of Spatio-temporal Properties. In *Proceedings of the 5th ACM International Workshop on Verification and mOnitoring at Runtime EXecution (VORTEX '21), July 12, 2021, Virtual, Denmark*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3464974.3468448

VORTEX '21, July 12, 2021, Virtual, Denmark

ACM ISBN 978-1-4503-8546-6/21/07…$15.00
https://doi.org/10.1145/3464974.3468448

1 DISTRIBUTED RUNTIME VERIFICATION

Runtime monitoring is a lightweight verification technique dealing with the observation of a system execution with respect to a specification [12]. Specifications are usually trace- or stream-based, with events mapped to atomic propositions in the underlying logic of the specification language. Popular specification languages include regular expressions and the Linear Time Logic (LTL). Distributed runtime monitoring comprises both monitoring of distributed systems, and using distributed systems for monitoring. Distribution is particularly challenging for verification purposes, as it requires to deal with issues such as synchronisation, communication faults, lack of global time, and so on.

In this paper, we address the design of distributed and decentralised runtime monitors [8], assuming that every agent of the system executes independently, and occasionally synchronizes or communicates with other agents via a given communication platform. Following Francalanza et al's terminology, we model agents as *processes* and consider any two processes as *remote* to each other. Every process produces a *local trace of events*, which is a sequence of sets of observable values derived from the agent's sensors or behaviour. Agents are allowed to appear or disappear from the system, thus different local traces are never aligned in time, i.e., events in the same position of each trace do not necessarily happen at the same time. Monitors check properties of the system by analysing their traces. We follow an *online* evaluation strategy, where the monitors are executed together with the processes themselves, being hosted at the same location and communicating with neighbour monitors. We assume that every agent is executing the same monitor, and this allows us to connect with the traditional setup of runtime verification despite the distributed setting: from the perspective of a single monitor, a single trace is evaluated, although this trace may contain events from remote nodes. Our approach is able to ignore *failures*, which usually make distributed systems harder to manage: a non-responsive node does not disrupt the distributed monitoring process, although influencing its verdict. We do not explicitly address message corruption or faulty sensors, delegating this issue to integrity measures on the communication layer. We improve over previous works on distributed runtime verification [7] by proposing an automated synthesis of monitors from high-level specifications in a logic with modalities for both space and time.

2 AGGREGATE COMPUTING

Intensive research has revolved on the search for suitable programming abstractions for ensembles of devices [4, 14]. Among those, *aggregate computing* [5] has sprouted as a generalisation of the previous approaches, aiming to define a programming model able to express complex distributed processes through function composition,

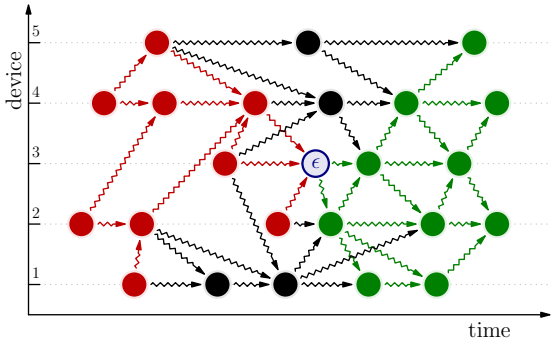

Figure 1: A sample event structure, split in events ϵ' in the causal past of ϵ ($\epsilon' < \epsilon$, in red), events in the causal future ($\epsilon < \epsilon'$, in green) and concurrent (non-ordered, in black).

with a semantics defined in terms of a gossip-like computational process, and supporting reusability of collective adaptive behaviour. Inspired by "fields" in physics, this aim is achieved through the notion of *computational field*, defined as a global data structure mapping devices of the distributed system to computational values. Such fields can be computed from a set of input fields (e.g., from sensors) either at a low-level, through simple programming language constructs, or at a high-level by composition of general-purpose building blocks of reusable behaviour, and can be ultimately fed to actuators to realise a whole collective adaptive services.

The *field calculus* [3] is a minimal but universal [1] language for aggregate computations over distributed networks of mobile devices, each capable of asynchronously performing simple local computations, and of interacting with a neighbourhood by local exchanges of messages. The field calculus provides mechanisms to express and compose such distributed computations, on a level of abstraction that avoids the explicit management of message exchanges, device position and quantity, and so on. In this context, a single program is periodically and asynchronously executed on every device, according to a cyclic schedule:

(1) first, the device gathers contextual information from sensors, local memory, and recently collected messages; the latter in the form of a *neighbouring value*, which is a map from neighbour devices δ to values v;
(2) then, when a computation round starts, the device evaluates the program with the information just gathered as input;
(3) finally, the computation result is stored locally, broadcast to neighbours, and possibly fed to actuators (e.g., motors, robotic arms, user interfaces, and the like).

Through the repetitive execution of rounds as above, across space (where devices are located) and time (when devices start a new cycle), a global behaviour emerges [13], which can be understood as occurring on the overall network of interconnected devices, modelled as a single *aggregate machine* equipped with a neighbouring relation. We give a formal semantics to field calculus programs through the classical notion of *event structure* [10], which we will also use later to interpret temporal logic formulas.

Definition 2.1 (Event Structure). An *event structure* $\mathbf{E} = \langle E, \rightsquigarrow \rangle$ is a finite set of events E together with an acyclic *neighbouring relation* $\rightsquigarrow \subseteq E \times E$ modelling message passing. We say that a sequence of neighbour events $\epsilon_1 \rightsquigarrow \ldots \rightsquigarrow \epsilon_n$ is a *message path*.

Event neighbouring induces the *causality relation* $< \subseteq E \times E$, defined as the transitive closure of $\rightsquigarrow$ and modelling causal dependence. An example structure is shown in Figure 1. In practice, event structures arise from device neighbourhood graphs changing over time. For instance, device 3 in Figure 1 appears at a certain point in time, with devices 4 and 1 as neighbours, but after a few steps its neighbours become devices 2 and 4.

In the following, we will present some small snippets of field calculus code, exploiting standard programming language notation together with the following two domain-specific constructs.

- `nbr(e0){e}`. Each device δ evaluates this expression by broadcasting the value of e to neighbours, and producing a neighbouring value mapping each neighbour δ' of δ (including δ itself) to the latest value that δ' has shared for e. If this is the first execution of the expression, there is no previously shared value for δ, and the value of e_0 is used in its place. For example, consider program `nbr(false){q()}` where q is a Boolean built-in function returning an observable. In the event ϵ of Figure 1, the program computes a map associating devices 2, 3 and 4 to their value of q in their last red event. In the first event on device 3, the default value is used to compensate the lack of a previous value, thus producing a map associating device 3 to `false`.
- `share(e0){(x) => e}`. In each device, the result of such an expression is obtained by first:
  - gathering a neighbouring value n similarly as nbr above;
  - evaluating e by substituting n to x, obtaining the overall value v for e; and finally
  - broadcasting v to neighbours, which will use it in their following rounds to produce their neighbouring value n.

For example, consider the following function declaration:

```
def dist(source) {
    share (infinity) { (d) =>
        if(source) {0} else {minHood(d)+1}
}   }
```

Function `dist` computes hop-count distances from the closest device where `source` holds, through a single **share** construct. This construct gathers neighbours' distance estimates into a neighbouring value n (mapping neighbour devices to their distance estimate), which is then substituted for variable `d`. In the first round of execution, the distance estimate `infinity` is used for the current device, modelling that no information about a source is yet available. The **share** body returns zero on source devices, or the minimum across neighbours' estimates `minHood(d)` increased by one otherwise, mimicking the classical Bellman-Ford routine.

3 TEMPORAL AND SPATIAL LOGICS

In order to express properties of systems distributed in space and time, both spatial and temporal logics have been studied. On the former, the SLCS spatial logic has been shown to be naturally translatable into field calculus monitors [2]. On the latter, preliminary results hint that the past-CTL temporal logic [9, 11] should also be naturally translatable into field calculus monitors. However, the

$$\phi ::= \bot \mid \top \mid q \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \Rightarrow \phi) \mid (\phi \Leftrightarrow \phi) \quad \text{logical}$$
$$\mid (\mathrm{P}\,\phi) \mid (\mathrm{AP}\,\phi) \mid (\mathrm{EP}\,\phi) \mid (\mathrm{H}\,\phi) \mid (\mathrm{AH}\,\phi) \mid (\mathrm{EH}\,\phi) \quad \text{temporal}$$
$$\mid (\mathrm{Y}\,\phi) \mid (\mathrm{AY}\,\phi) \mid (\mathrm{EY}\,\phi) \mid (\phi\,\mathrm{S}\,\phi) \mid (\phi\,\mathrm{AS}\,\phi) \mid (\phi\,\mathrm{ES}\,\phi)$$
$$\mid (\Box\,\phi) \mid (\Diamond\,\phi) \mid (\partial\,\phi) \mid (\partial^{-}\,\phi) \mid (\partial^{+}\,\phi) \quad \text{spatial}$$
$$\mid (\phi\,\mathcal{R}\,\phi) \mid (\phi\,\mathcal{T}\,\phi) \mid (\phi\,\mathcal{U}\,\phi) \mid (\mathcal{G}\,\phi) \mid (\mathcal{F}\,\phi)$$

Figure 2: Syntax of past-CTL and SLCS.

interplay between the two logics has not been studied so far, as their interpretation is apparently incompatible: SLCS is interpreted on *graphs*, while past-CTL is interpreted on *event structures*. Section 4 will present a preliminary investigation on a combination of the two. We now present the two logics together, but assuming for the moment that temporal and spatial modalities are never mixed.

Figure 2 presents the syntax of the past-CTL and SLCS logics. They are based on atomic propositions q representing observables, share usual logical operators, and enrich them with either temporal or spatial modalities. The temporal modalities are almost identical to those in traditional CTL, with two main differences:

- temporal operators are interpreted in the past (and thus their names are changed accordingly), along *message paths* that all happened (and are not alternative realities); this ensures that formulas have a definite truth value computable at runtime;
- there are un-quantified versions of the operators along with quantified versions, which refer to the linear past on a same device (and thus behave as past-LTL operators).

Past-CTL formulas can be interpreted in event structures (Fig. 1), giving a truth value for each event. The modalities take inspiration from the words *Yesterday*, *Since*, *Previously*, *Historically*. We choose Y, EY, S, AS, ES as primitive, with the following informal meaning:

- $\mathrm{Y}\,\phi$ means "ϕ held in the previous event on the same device";
- $\mathrm{EY}\,\phi$ means "ϕ held in some previous event on any device";
- $\phi\,\mathrm{S}\,\psi$ means "ψ held in some past event on the same device, and ϕ has held on the same device since then";
- similarly, $\phi\,\mathrm{AS}\,\psi$ (resp. $\phi\,\mathrm{ES}\,\psi$) mean "for all paths (resp. exists a path) of messages reaching the current event, ψ held in some event of the path and ϕ has held since then".

The other operators can be derived from them by usual means, through $\mathrm{AY}\,\phi \triangleq \neg\,\mathrm{EY}\,\neg\phi$; $\mathrm{P}\,\phi \triangleq \top\,\mathrm{S}\,\phi$ (similarly for AP, EP with AS, ES); $\mathrm{H}\,\phi \triangleq \neg\,\mathrm{P}\,\neg\phi$ (similarly for AH, EH with EP, AP).

The spatial modalities are those of SLCS [2, 6], and can be divided into *local* and *global* modalities. The local modalities are:

- $\Box\,\phi$ (interior): true at points where all neighbours satisfy ϕ;
- $\Diamond\,\phi$ (closure): true at points where a neighbour satisfies ϕ;
- ∂, ∂^{-} and ∂^{+} represent respectively *boundary* (closure without interior), *interior boundary* (set without the interior) and *closure boundary* (closure without the set).

We choose $\Diamond$ as primitive, expressing the others through the equivalences $\Box\,\phi \triangleq \neg \Diamond \neg\phi$, $\partial\,\phi \triangleq (\Diamond\,\phi) \wedge \neg(\Box\,\phi)$, $\partial^{-}\,\phi \triangleq \phi \wedge \neg(\Box\,\phi)$, $\partial^{+}\,\phi \triangleq (\Diamond\,\phi) \wedge \neg\phi$. The global modalities are:

- $\phi\,\mathcal{R}\,\psi$ (reaches): true at the ending points of paths in the graph whose starting point satisfies ψ, and where ϕ holds on each point on the path;
- $\phi\,\mathcal{T}\,\psi$ (touches): true at the end of paths whose start satisfies ψ and where ϕ holds in the rest of the path;
- $\phi\,\mathcal{U}\,\psi$ (surrounded by): true at points in an area satisfying ϕ, whose closure boundary satisfies ψ;
- $\mathcal{G}\,\phi, \mathcal{F}\,\phi$ (everywhere, somewhere): true where ϕ holds in every (resp. some) point of every (resp. some) incoming path.

We choose $\mathcal{R}$ as primitive, expressing the others through $\phi\,\mathcal{T}\,\psi \triangleq \phi\,\mathcal{R} \Diamond \psi$, $\phi\,\mathcal{U}\,\psi \triangleq \phi \wedge \Box \neg(\neg\psi\,\mathcal{R}\,\neg\phi)$, $\mathcal{F}\,\phi \triangleq \top\,\mathcal{R}\,\phi$, $\mathcal{G}\,\phi \triangleq \neg\,\mathcal{F}\,\neg\phi$.

A possible translation of SLCS and past-CTL formulas into field calculus is shown in Figure 3, by recursion on sub-formulas. We translate atomic propositions q into built-in function calls `q()` getting their value from some external environment, and logical operators into their field calculus representation. We assume that:

- **`nbr`**, **`share`** and `dist` are as in Section 2;
- `D` is an upper bound to the network diameter (either fixed at design time or estimated through an aggregate process);
- `anyHood`, `allHood`, `locHood` are built-in operators collapsing a Boolean neighbouring value ϕ into the conjunction (resp. disjunction, local value) of its constituent values.

Notice that as function `dist` in the translation of $\phi_1\,\mathcal{R}\,\phi_2$ is computed within a branch, it only receives messages from neighbours which selected the same branch, i.e., for which ϕ_1 is true. Thus, that function call computes the shortest distance from a point satisfying ϕ_2, restricted within the region where ϕ_1 is true. Note also that the translations of $\Diamond$ and EY (and thus $\Box$ and AY) are the same, since *neighbour devices* can only be accessed through *previous events*.

4 TOWARDS A SPACE-TIME LOGIC

Temporal formulas in past-CTL characterise properties of the *past cone of events* for each event of an event structure. This "cone" is a set distributed in time as well as in space: it follows that past-CTL formulas are already able to capture some space-related properties. This is achieved by exploiting the "path quantifiers" A and E, allowing paths of messages to span the network.

On the other side, even though the semantics of spatial formulas in SLCS is defined on timeless graphs, their implementation

$\top$	`true`	q	`q()`	$\neg\phi$	`!`ϕ	$\phi_1 \vee \phi_2$	ϕ_1 `||` ϕ_2
$\Diamond\,\phi$	`anyHood(nbr(false){`ϕ`})`			$\phi_1\,\mathcal{R}\,\phi_2$	**`if`** `(`ϕ_1`) {dist(`ϕ_2`)<D}` **`else`** `{false}`		
$\mathrm{EY}\,\phi$	`anyHood(nbr(false){`ϕ`})`			$\mathrm{Y}\,\phi$	`locHood(nbr(false){`ϕ`})`		
$\phi_1\,\mathrm{S}\,\phi_2$	**`share`** `(false) {(old) =>` ϕ_2 `||` `(`ϕ_1 **`&&`** `locHood(old))}`						
$\phi_1\,\mathrm{AS}\,\phi_2$	**`share`** `(false) {(old) =>` ϕ_2 `||` `(`ϕ_1 **`&&`** `allHood(old))}`						
$\phi_1\,\mathrm{ES}\,\phi_2$	**`share`** `(false) {(old) =>` ϕ_2 `||` `(`ϕ_1 **`&&`** `anyHood(old))}`						

Figure 3: Translation of a primitive set of SLCS and past-CTL operators into field calculus.

according to Figure 3 is executed through time, inducing a well-defined behaviour on event structures. Future work may attempt at developing a formal characterisation of such behaviour. For the scope of this paper, we may understand this behaviour intuitively, by interpreting an SLCS formula in an event ϵ as being evaluated with respect to the *subjective present* of ϵ. This subjective present is a graph consisting of a single event for every device in the network: the most recent event on that device in the past of ϵ.

Combining these two observations, it follows that past-CTL and SLCS can indeed be used together, and possibly fused into a single space-time logic, that can be monitored according to the translation in Figure 3. In this combined logic, the concepts of "immediate past" and "immediate neighbourhood" fuse together, as shown by the equivalence of $\Diamond$ with EY and of $\Box$ with AY, since the most recent view possible of the immediate neighbourhood is that coming from the immediate past of neighbours, which is what the past-CTL operators EY and AY are designed to access. While local spatial modalities happen to be expressible in past-CTL, global spatial modalities strictly improve over the capabilities of past-CTL, by allowing to capture a notion of "present" that would be ineffable otherwise. For instance, consider a simple formula such as $\mathcal{F}\, q$, which holds given that q held somewhere in the subjective present of the current event. An analogous concept cannot be captured by any past-CTL formula: the closest relative would be $\mathrm{EP}\, q$, which however produces a very different behaviour. In fact, $\mathcal{F}\, q$ can start false, then turn true as q starts being true somewhere, and then turn false again if q stops holding. On the other hand, $\mathrm{EP}\, q$ can start false, but when it turns true it has no way of turning back to false.

The proposed space-time logic, obtained by merging past-CTL with SLCS, strictly improves over the expressibility of both, allowing to express properties that were previously out of reach. In the remainder of this paper, we substantiate this claim through examples in a network monitoring scenario. We consider atomic propositions s identifying *servers*, and d which is true on devices which are *down* (regardless on whether they are servers or not).

There is currently a server that has always been down. We can express that the current device has always been down through the past-CTL formula $\mathrm{H}\, d$; and assert that it is also a server through $s \wedge \mathrm{H}\, d$. Finally, we can check whether there is currently such a server through the *somewhere* modality of SLCS, as $\mathcal{F}(s \wedge \mathrm{H}\, d)$. In field calculus, using that $\mathcal{F}\, \phi \triangleq \top\, \mathcal{R}\, \phi$ and $\mathrm{H}\, \phi \triangleq \neg(\top\, \mathrm{S}\, \neg\phi)$, and simplifying tautologies, this formula gets translated to:

```
dist(s && !share (false) {(old) => !d || locHood(old)}) < D
```

At some point in the past, every server was simultaneously down. Firstly, we need to express that everywhere there is a server, it must be down. This can be written as $\mathcal{G}(s \Rightarrow d)$, which is an SLCS formula. In order to check whether this formula has ever been true, we resort to the EP modality of past-CTL, obtaining the space-time formula $\mathrm{EP}\, \mathcal{G}(s \Rightarrow d)$ formalising the target property.

Servers can always be reached through trustworthy devices only, i.e., devices that have never been down in their past. We can characterise trustworthy devices within past-CTL, as those that have never previously been down, through the simple formula $\neg\, \mathrm{P}\, d$. In order to check whether a server can be reached through those devices only, we exploit the $\mathcal{R}$ operator in SLCS obtaining $(\neg\, \mathrm{P}\, d)\, \mathcal{R}\, s$, which can be read as "never previously down (i.e., trustworthy) devices can reach servers". As a final step, we check that this property has always been satisfied by adding an AH modality: $\mathrm{AH}((\neg\, \mathrm{P}\, d)\, \mathcal{R}\, s)$.

5 CONCLUSION

In this paper, we presented the temporal and spatial logics past-CTL and SLCS, together with their automatic translation into aggregate monitors in field calculus. Then, we discussed how the two logics could be blended together, pointing out similarities and differences, and finally presenting some examples mixing space and time modalities to showcase the effectiveness of the proposed approach.

In the future, we plan to expand on this investigation by providing an abstract interpretation of SLCS formulas on event structures, possibly expand the set of supported modalities, and finally testing the approach on a simulated realistic case study.

REFERENCES

[1] Giorgio Audrito, Jacob Beal, Ferruccio Damiani, and Mirko Viroli. 2018. Space-Time Universality of Field Calculus. In *Coordination Models and Languages (COORDINATION) (Lecture Notes in Computer Science, Vol. 10852)*. Springer, 1–20. https://doi.org/10.1007/978-3-319-92408-3_1

[2] Giorgio Audrito, Roberto Casadei, Ferruccio Damiani, Volker Stolz, and Mirko Viroli. 2021. Adaptive distributed monitors of spatial properties for cyber-physical systems. *J. Syst. Softw.* 175 (2021), 110908. https://doi.org/10.1016/j.jss.2021.110908

[3] Giorgio Audrito, Mirko Viroli, Ferruccio Damiani, Danilo Pianini, and Jacob Beal. 2019. A Higher-Order Calculus of Computational Fields. *ACM Trans. Comput. Log.* 20, 1 (2019), 5:1–5:55. https://doi.org/10.1145/3285956

[4] Jacob Beal, Stefan Dulman, Kyle Usbeck, Mirko Viroli, and Nikolaus Correll. 2013. Organizing the Aggregate: Languages for Spatial Computing. In *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments*. IGI Global, Chapter 16, 436–501. https://doi.org/10.4018/978-1-4666-2092-6.ch016

[5] Jacob Beal, Danilo Pianini, and Mirko Viroli. 2015. Aggregate Programming for the Internet of Things. *IEEE Computer* 48, 9 (2015), 22–30. https://doi.org/10.1109/MC.2015.261

[6] Vincenzo Ciancia, Diego Latella, Michele Loreti, and Mieke Massink. 2014. Specifying and Verifying Properties of Space. In *8th IFIP International Conference in Theoretical Computer Science (TCS) (Lecture Notes in Computer Science, Vol. 8705)*. Springer, 222–235. https://doi.org/10.1007/978-3-662-44602-7_18

[7] Adrian Francalanza, Andrew Gauci, and Gordon J. Pace. 2013. Distributed system contract monitoring. *J. Log. Algebraic Methods Program.* 82, 5-7 (2013), 186–215. https://doi.org/10.1016/j.jlap.2013.04.001

[8] Adrian Francalanza, Jorge A. Pérez, and César Sánchez. 2018. Runtime Verification for Decentralised and Distributed Systems. In *Lectures on Runtime Verification - Introductory and Advanced Topics*. Lecture Notes in Computer Science, Vol. 10457. Springer, 176–210. https://doi.org/10.1007/978-3-319-75632-5_6

[9] Nicola Gigante, Angelo Montanari, and Mark Reynolds. 2017. A One-Pass Tree-Shaped Tableau for LTL+Past. In *Logic for Programming, Artificial Intelligence and Reasoning (LPAR) (EPiC Series in Computing, Vol. 46)*. EasyChair, 456–473. http://www.easychair.org/publications/paper/340363

[10] Leslie Lamport. 1978. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM* 21, 7 (1978), 558–565. https://doi.org/10.1145/359545.359563

[11] François Laroussinie and Philippe Schnoebelen. 1995. A Hierarchy of Temporal Logics with Past. *Theor. Comput. Sci.* 148, 2 (1995), 303–324. https://doi.org/10.1016/0304-3975(95)00035-U

[12] Martin Leucker and Christian Schallhart. 2009. A brief account of runtime verification. *J. Log. Algebr. Program.* 78, 5 (2009), 293–303. https://doi.org/10.1016/j.jlap.2008.08.004

[13] Mirko Viroli, Giorgio Audrito, Jacob Beal, Ferruccio Damiani, and Danilo Pianini. 2018. Engineering Resilient Collective Adaptive Systems by Self-Stabilisation. *ACM Transactions on Modeling and Computer Simulation* 28, 2 (2018), 16:1–16:28. https://doi.org/10.1145/3177774

[14] Mirko Viroli, Jacob Beal, Ferruccio Damiani, Giorgio Audrito, Roberto Casadei, and Danilo Pianini. 2018. From Field-Based Coordination to Aggregate Computing. In *Coordination Models and Languages (COORDINATION) (Lecture Notes in Computer Science, Vol. 10852)*. Springer, 252–279. https://doi.org/10.1007/978-3-319-92408-3_12