Tuple-Based Coordination in Large-Scale Situated Systems

Tuple-based Coordination in Large-Scale Situated Systems

Roberto Casadei[0000−0001−9149−949X], Mirko Viroli[0000−0003−2702−5702], Alessandro Ricci[0000−0002−9222−5092], and Giorgio Audrito[2][0000−0002−2319−0375]

[1] Alma Mater Studiorum–Università di Bologna, Italy
{roby.casadei, mirko.viroli, a.ricci}@unibo.it
[2] Università di Torino, Italy
{giorgio.audrito}@unito.it

Abstract. *Space* and *time* are key elements for many computer-based systems and often elevated to first-class abstractions. In tuple-based coordination, Linda primitives have been independently extended with space (with tuples and queries spanning spatial regions) or time information (mostly for tuple scoping). However, recent works in collective adaptive systems and aggregate computing show that space and time can naturally be considered as two intertwined facets of a common coordination abstraction for situated distributed systems. Accordingly, we introduce the *Spatiotemporal Tuples* model, a natural adaptation of Linda model for physically deployed large-scale networks. Unlike prior research, spatiotemporal properties – expressing where and when a tuple should range and has to be deposited/retrieved – naturally turn into specifications of collective adaptive processes, to be carried on in cooperation by the devices filling the computational environment, and sustaining tuple operations in a resilient way, possibly even in mobile and faulty environments. Additionally, the model promotes decentralised implementations where tuples actually reside where they are issued, which is good for supporting peer-to-peer and mobile ad-hoc networks as well as privacy. In this paper, we *(i)* present and formalise the Spatiotemporal Tuples model, based on the unifying notion of computational space-time structure, *(ii)* provide an implementation in the ScaFi aggregate computing framework, turning tuple operations into aggregate processes, and finally *(iii)* provide evaluation through simulation and a rescue case study.

Keywords: tuple-based coordination; spatial tuples; self-organisation; aggregate computing; ScaFi

1 Introduction

Space and time are fundamental aspects of our reality. *Space*, logical or physical, plays a fundamental role for many computer-based systems. This has been recognised, e.g., in the Dagstuhl seminar on space-oriented computation [20], where *(i) coping with space* for computational efficiency, *(ii) embedding in space*, and

(iii) representing space are shown to be dimensions characterising a wide range of computing applications. Accordingly, several research fields have elevated space to a first-class abstraction [7]. A notable example in the coordination field, and specifically in tuple-space coordination [21], is the *Spatial Tuples* model [30], where tuples are situated in regions of space and Linda coordination primitives also depend on the spatial situation of the coordinating agents. Dually, *time* has also been investigated as an explicit abstraction, leading to notions of time for tuple-based systems [26,25]. However, recent works in collective adaptive systems and field-based coordination/aggregate computing [8,32,6] show that space and time can naturally be considered as intertwined facets of a common coordination abstraction for situated distributed systems.

Therefore, in this paper we introduce a tuple-based coordination model that considers space and time in combined form. Differently from prior research, spatiotemporal properties – expressing where and when a tuple should range and has to be deposited/retrieved – naturally turn into specifications of collective adaptive processes, to be carried on in cooperation by the devices filling the computational environment, and sustaining tuple operations in a resilient way even in mobile and faulty environments. Most specifically, an `out` creates a tuple that spreads in a dynamically changing region of space, `rd` similarly spreads a query that unblocks the initiator if a match is found in the intersection of a tuple region, and finally `in` performs like `rd` but additionally disables/removes the tuple, making it inaccessible to other queries. Therefore, our contribution is threefold[3]:

1. we present the *Spatiotemporal Tuples model*, in terms of a declarative semantics of coherent tuple space evolution, and a compliant protocol solution by a set of processes running on a computational space-time structure;
2. we describe an implementation in the ScaFi *aggregate computing* toolkit [13,14], where tuple operations and spatiotemporal properties are expressed as aggregate processes [15] creating so-called *computational fields*; and
3. we evaluate model and implementation by means of simulation.

A major merit of the overall approach is that it fosters implementations that are inherently suitable to different kinds of system deployments, ranging from fully peer-to-peer systems to centralised, cloud-based architectures.

The paper is organised as follows. Section 2 provides background and related work in the area of tuple-based and space-based coordination. Section 3 describes the Spatiotemporal Tuple model. Section 4 describes an implementation of the model in terms of aggregate processes in ScaFi. Section 5 provides evaluation. Finally, Section 6 provides conclusion and future work.

[3] This work extends the workshop paper in [17] with formalisation, full implementation, and evaluation.

2 Background and Related Work

Tuple-based coordination is a coordination paradigm where a collection of *processes* coordinate by reading and writing *tuples* (ordered groups of values) on a *shared tuple space* [21]. Tuple-based coordination logic can be expressed in a language, such as the progenitor *Linda* [21], specifying process evolution in terms of operations on tuples (e.g., write, read, removal). Tuple-based coordination has been subject of extensive research, giving rise to several variants, extensions, and implementations. In particular, in distributed and pervasive computing scenarios, issues with the notion of a centralised tuple space tend to promote alternative models with several local tuple spaces [22]. In the following literature review, we survey the main contributions focussing on spatiotemporally situated systems.

2.1 Tuple-based coordination in pervasive systems and space(-time)

Works have been proposed in the literature implementing tuple-based coordination for peer-to-peer (P2P) and mobile ad-hoc networks (MANETs) [23]. In these scenarios, challenges and opportunities include mobility, dynamicity, locality, openness. Such features are often found in nature-inspired systems (cf. *SwarmLinda* [31]) and can be exploited to build scalable systems exhibiting *collective intelligence* [16]. These challenges require proper middleware support and, sometimes, model and language extensions to deal with specific aspects including situatedness and mobility. When a device is physically situated, it also acts as a representative of a space-time region, providing a means for representing, measuring, computing space—as exploited by *spatial computing* approaches [7].

$\sigma\tau$-Linda [33] is an approach where Linda operations can be combined with time- and network-oriented operations. Example constructs include *neigh* (to spread an operation to the neighbourhood), *next* (to post-pone an operation to the next program evaluation), and *finally* (to run an operation after a barrier of other operations). A further extension proposed by $\sigma\tau$-Linda is spatiotemporally limited tuple operations, which this paper develops in a principled way.

LIME [27] provides a support for Linda in MANETs. It distinguishes between *logical* and *physical* mobility of agents and hosts, to model both component situation as well as topology change. Agents own a tuple space locally, and group with other co-located and neighbour-host agents to consolidate tuple spaces, supporting access to the overall tuple space of the entire group by a *transient sharing* mechanism. In this paper, we also leverage a notion of "group", to mean the set of devices cooperating to support a space-time tuple operation.

The *TOTA* approach [24] views tuples as dynamic elements, which can be copied and change both their location and shape. Specifically, *scope*, *transformation*, and *maintenance* rules can be specified to define and control how tuples are to be propagated in a network, and how these must evolve and react to environmental events along the path. In this paper, we leverage similar ideas to propagate tuple *operations* (rather than tuples). The idea in TOTA that tuples automatically propagate to neighbours is also at the basis of the approach of this paper, where aggregate processes deal with propagation control (cf. Section 4.1).

In *GeoLinda* [28], tuples spaces are distributed and *geometry-aware*: both tuples and reading operations have a *volume* (spatial extension). They call the volume of a tuple its *shape*, and the volume of a reading operation its *addressing shape*. Shapes can take various geometric forms (spheres, cones, etc.) and are expressed relatively to a device's location and orientation. In this paper, we leverage the same idea of giving tuple operations a space-time extension, and also define a way to express them in spatiotemporally situated networks of devices.

2.2 Spatial Tuples

Spatial Tuples [30] is a coordination model combining tuple-based with space-based coordination. Its idea is to decorate tuples with spatial information, in order to situate them to some *point in space* or some *spatial region* (in which case, the tuple is said to have a *spatial extension*). The Spatial Tuples approach comprises multiple *languages* for working with spatial tuples: a *communication language* is used to express tuples and tuple templates (for matching), a *space description language* is used to express spatial information, and a *coordination language* is used for process interaction and evolution. The latter consists of the following main spatial primitives [30]:

- `out(t @ r)` — for situating a tuple `t` to a spatial location or region `r`.
- `rd(tt @ r)` — for blocking until a tuple `t` matching template `tt` and intersecting region `r` is read (with non-deterministic choice).
- `in(tt @ r)` — for blocking until a tuple `t` matching template `tt` and intersecting region `r` is removed (with non-deterministic choice).

The space description language is application- or domain-specific and may allow expressing geographic locations and regions. The situation of a tuple, however, does not need to be constant. For instance, a tuple can be attached to another situated component, and hence its position would be defined *indirectly*. Given a component `id`, `t @ id` would express that tuple `t` is *bound* to `id`. Such a notion of *binding* is especially relevant in scenarios with *mobility*. Locations and bindings could also be specified implicitly:

- `t @ here`: situates tuple `t` at the current position of the running component;
- `t @ me`: the location of tuple `t` is bound to that of the running component.

The Spatial Tuples approach fosters space-oriented coordination through mechanisms for *situated/stigmergic communication*, where processes deposit and sense data at specific locations, and *spatial synchronisation*, where the actions multiple interacting processes are ordered depending on their spatial situation.

In Spatial Tuples, the key idea is to use spatial information to annotate and retrieve tuples. There are parallelisms with *attribute-based* coordination [1], whereby attributes are used to form and let ensembles interact. In this work, however, we consider spatial information not just as a mere annotation to tuples or components but as a specification driving and evolving spatiotemporally situated computational processes.

3 A Model for Spatiotemporal Tuple-based Coordination

3.1 Requirements

The Spatiotemporal Tuples model is designed to address the following concerns:

- **Space.** The model should capture situations in space, and provide suitable spatial abstractions to capture diverse situations. Namely, we mean to provide a *computational* notion of space, where space locations are associated with computational nodes, and proximity of locations matches the ability of a device to directly perceive its context, there including message reception.
- **Time.** The model should dually capture temporal situations, while abstracting over the notion of time, and hence of system evolution. Also, since we expressly target fully distributed systems, for which no general notion of global time exists [19], the model should provide the expressiveness to specify what/how notions of local time can be used and propagated. Menezes et al. [25] discuss the issues with using external notions of time in Linda-based systems, and propose to measure time locally to observers of fadeable tuples.
- **Consistency.** The model should adhere to the general Linda semantics, namely, ensuring safe interaction of primitives `out/rd/in` as formalised in [11]. Unfortunately, in distributed settings, the *CAP theorem* [10] enters the picture, asserting that you may pick only two among the three properties: consistency, availability, and partition tolerance. This is an issue when implementing atomic consumption of tuples (for `in` operation). However, designers can leverage the many nuances in these properties and combinations.
- **Heterogeneous deployments.** The model should provide for a direct implementation for different kinds of underlying platforms, such as MANETs, P2P networks, client/server, and cloud-based architectures. Namely, it should be sufficiently *general* to capture diverse settings, also considering the architectures and constraints of modern distributed systems.

3.2 Computational space-time model

Defining a spatiotemporal model for Linda primitives requires a suitable underlying notion of computability, since there is need of tracking information propagation in space and time. Thus, we base our model on the notion of *space-time computability* of [2], which in turn founds on the *event structure* framework [34]. In this section, we recall this framework, tailored for the needs of this paper.

Definition 1 (Augmented Event Structure [2]). *An* augmented event structure *is a quadruple* $\mathbf{E} = \langle E, \rightsquigarrow, d, s \rangle$ *where* E *is a countable set of* events, $\rightsquigarrow \subseteq E \times E$ *is a* messaging *relation,* $d : E \to \Delta$ *is a mapping from events to the devices where they happened,* $s : E \to S$ *is a mapping from events to sensors status (for any choice of a representation of sensors status* $\sigma \in S$*), such that:*

- *for any* $\delta \in \Delta$*, the set of events* $E_\delta = \{\epsilon \in E \mid d(\epsilon) = \delta\}$ *forms a sequence of chains, i.e., there are no distinct* $\epsilon, \epsilon_1, \epsilon_2 \in E_\delta$ *such that either* $\epsilon \rightsquigarrow \epsilon_i$ *for* $i = 1, 2$ *or* $\epsilon_i \rightsquigarrow \epsilon$ *for* $i = 1, 2$*,*

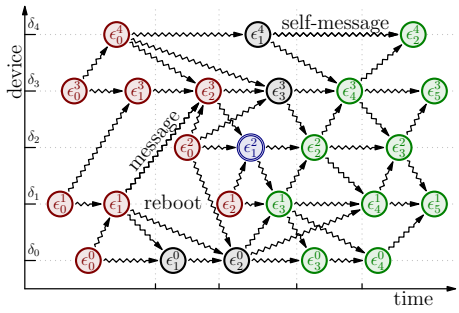


Fig. 1: An augmented event structure $\mathbf{E}$. It depicts events (circle nodes), messaging relations (curly arrows), devices $\delta_0, ..., \delta_4$ (y-axis) and each circle node is labelled with the depicted event of $\mathbf{E}$. Colours denote the causal relation w.r.t. the reference event ϵ_1^2 (doubly-circled, blue), partitioning events into causal past (red), causal future (green) and concurrent (non-ordered, in black).

- *the transitive closure of* $\rightsquigarrow$ *forms an irreflexive partial order* $< \subseteq E \times E$*, called* causality relation*,*
- *the set* $X_\epsilon = \{\epsilon' \in E \mid \epsilon' < \epsilon\} \cup \{\epsilon' \in E \mid \epsilon \rightsquigarrow \epsilon'\}$ *is finite for all* ϵ *(i.e.,* $\rightsquigarrow$ *and* $<$ *are locally finite).*

We say that event ϵ' *is a* supplier *of event* ϵ *iff* $\epsilon' \rightsquigarrow \epsilon$*.*

The intuition of this definition is that the messaging relation between events on the same device represents one-step passage of time, while the messaging relation between events on different devices represents proximity in space (and ability to directly interact). Figure 1 depicts a sample augmented event structure $\mathbf{E} = \langle E, \rightsquigarrow, d, s\rangle$ where $\mathbf{E} = \{\epsilon_0^0, ..., \epsilon_4^0, \epsilon_0^1, ..., \epsilon_5^1, \epsilon_0^2, ..., \epsilon_4^2, \epsilon_0^3, ..., \epsilon_3^3, \epsilon_0^4, ..., \epsilon_5^4\}$ consists of 24 events such that $d(\epsilon_j^i) = \delta_i$.

In this model, spatio-temporal tuple *regions* are definable subsets of the augmented event structure with a unique originating event.

Definition 2 (Spatio-temporal Region). *Let* $\mathbf{E} = \langle E, \rightsquigarrow, d, s\rangle$ *be an augmented event structure. A* spatio-temporal region $\mathtt{r}$ *is a definable*[4] *predicate associating a Boolean value* $\mathtt{r}(\epsilon, \epsilon') \in \{\top, \bot\}$ *to every pair of events* $\epsilon, \epsilon' \in \mathbf{E}$*, such that* $\mathtt{r}(\epsilon, \epsilon') = \top$ *implies that* $\epsilon \leq \epsilon'$*.*

We write $\mathtt{r}_\epsilon = \{\epsilon' \mid \mathtt{r}(\epsilon, \epsilon') = \top\} \subseteq E$ *for the set of events that belong to the spatio-temporal region described by* $\mathtt{r}$ *and originating from* ϵ*. We say such a set is* connected *if for every* $\epsilon' \in \mathtt{r}_\epsilon$ *with* $\epsilon' \neq \epsilon$*, there exists an* $\epsilon'' \in \mathtt{r}_\epsilon$ *such that* $\epsilon'' \rightsquigarrow \epsilon'$*. We say predicate* $\mathtt{r}$ *is connected if* $\mathtt{r}_\epsilon$ *is connected for every* $\epsilon \in E$*.*

In the definition above, ϵ can be understood as the event originating the region, and ϵ' as another event which is being checked for belonging to the region. Connected regions can be used to guide the local propagation of a spatial process, which can expand from the originating event to neighbours filtering out those outside of the region, and reach every event in the region this way. Propagation

[4] We do not give an explicit syntax for spatio-temporal regions, in order to cover applications with any such syntax. *Definable* corresponds to space-time computable [2], thus requiring the existence of a computational procedure deciding whether the predicate $\mathtt{r}(\epsilon, \epsilon')$ holds in some event ϵ' using only information in the past of ϵ'.

in non-connected regions $\mathtt{r}'$ needs to be handled by providing a connected region $\mathtt{r}$ to guide the propagation of the spatial process, such that $\mathtt{r}'_\epsilon \subset \mathtt{r}_\epsilon$, making the process inactive in events outside of $\mathtt{r}'$. As a paradigmatic example, consider the following two region predicates:

- $\mathtt{me}_k$, which holds on future events within k hops of the originating device: $\mathtt{me}_k(\epsilon, \epsilon') \Leftrightarrow \exists \epsilon_0 \ldots \epsilon_k \in E.\ d(\epsilon) = d(\epsilon_0) \ \wedge\ \epsilon \leq \epsilon_0 \rightsquigarrow \ldots \rightsquigarrow \epsilon_k = \epsilon'$;
- $\mathtt{here}_k$, which holds on future events within k hops of the originating *location*: $\mathtt{here}_k(\epsilon, \epsilon') \Leftrightarrow \exists \epsilon_0 \ldots \epsilon_k \in E.\ \ell(\epsilon) = \ell(\epsilon_0) \ \wedge\ \epsilon \leq \epsilon_0 \rightsquigarrow \ldots \rightsquigarrow \epsilon_k = \epsilon'$, where $\ell : E \to \mathcal{L}$ is a given map associating a location (from a finite set of locations $\mathcal{L}$) to each event. Notice that multiple devices may be simultaneously in the same location (unlike the `me` region).

We remark that the given theory can be applied to any other definable regions.

3.3 Specifications for spatio-temporal tuple operators

To provide a formalisation of spatio-temporal tuple operations, we specify what an acceptable behaviour is for them, by mirroring the traditional non-deterministic semantics of Linda in a distributed "event structures" setting. This effectively constitutes a declarative semantics of the spatio-temporal tuples language: a semantics at a denotational level is outlined in Section 3.4, while a concrete operational implementation is given in Section 4.2. To state the specification, we first need to define a notion of *tuple space evolution*, which is ultimately built from the following grammar of processes.

Definition 3 (Extended Spatio-temporal Process). *We define spatio-temporal processes P and extended spatio-temporal processes Q according to the following grammar:*

$$P ::= \mathtt{out}^\tau(\mathtt{t}\ \ @\ \mathtt{r}) \mid \mathtt{in}^\tau(\mathtt{tt}\ \ @\ \mathtt{r}).P \mid \mathtt{rd}^\tau(\mathtt{tt}\ \ @\ \mathtt{r}).P$$
$$Q ::= P \mid \mathtt{got}^{\tau_o,\tau_i}$$

where `t` *are tuples,* `tt` *tuple templates,* `r` *regions, τ unique identifiers.*

In this grammar, we avoided an explicit mention to classic process operators (parallel and non-deterministic composition, replication, etc.), as their treatment is orthogonal to the scope of this paper. This grammar follows closely that in Section 2.2, with few notable differences. First, every `out`, `in` and `rd` construct is marked with a unique identifier τ, discriminating every process from every other. Second, inert `got` processes are introduced in Q to mark the accesses of tuples from the distributed space, issued in the events when agreement is first reached about the matching of the `out` tuple with a corresponding `in` or `rd` process. As we shall see in the following definition, these agreement events must necessarily follow matching events (belonging to the intersection of the regions of the corresponding `out` and `in`/`rd` processes) and precede the continuation of the `in`/`rd` processes. These inert processes are of the form $\mathtt{got}^{\tau_o,\tau_i}$ where τ_o is the

unique identifier of the `out` process introducing the tuple, and τ_i is the unique identifier of the `in` or `rd` process accessing (and possibly removing) that tuple. Notice that the `got` processes are necessary for the formal definition of which matches are occurring in a computation. This information cannot be uniquely reconstructed from the continuations alone: indeed, the same continuation P may arise from reading/accessing (possibly) different tuples by different parent processes.

In the remainder of this paper, we write $name(P)$ for the τ first occurring in P and write $name(\texttt{got}^{\tau_o,\tau_i}) = (\tau_o, \tau_i)$. We also write $kind(P)$ for the construct first occurring in P (`out`, `in` or `rd`).

Definition 4 (Tuple Space Evolution). *Let* $\mathbf{E} = \langle E, \rightsquigarrow, d, s\rangle$ *be an augmented event structure, and let* $\mathcal{Q}$ *be the set of extended spatio-temporal processes according to Definition 3. A* tuple space evolution *is a function* $TS : E \to \mathcal{Q}^*$ *associating a finite set of processes* $TS(\epsilon) = \{Q_1, \ldots, Q_n\}$ *to each event* ϵ.

The following definition of a consistent tuple space evolution thus provides a specification of acceptable behaviours for spatio-temporal tuple processes.

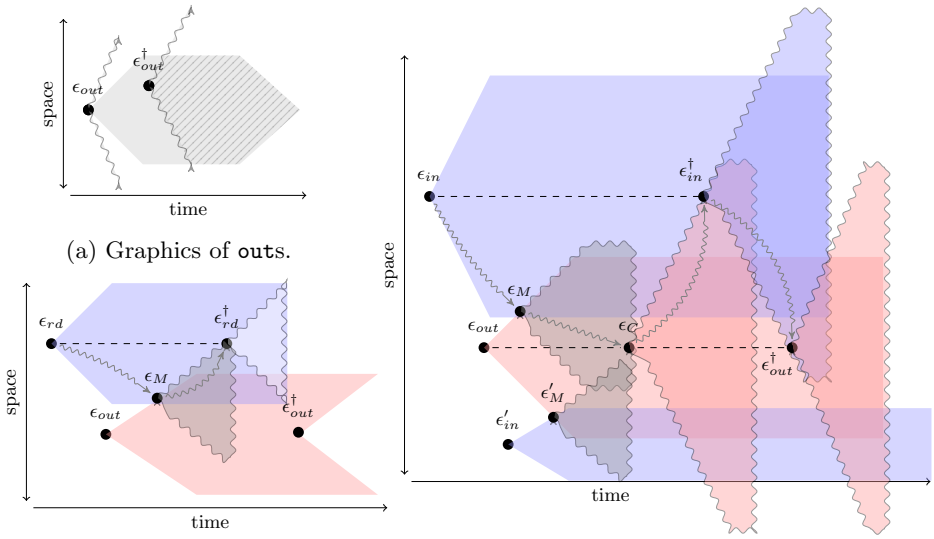
Definition 5 (Coherent Tuple Space Evolution). *Let* $TS : E \to \mathcal{Q}^*$ *be a tuple space evolution on* $\mathbf{E} = \langle E, \rightsquigarrow, d, s\rangle$ *and* $\mathcal{Q}$. *We say that* TS *is coherent if it respects the following properties for any* $\epsilon_x \in E$ *where* $x = 1, 2, 3, g, i, o$.

1. ***Identifier uniqueness:*** *given* $Q_1 \in TS(\epsilon_1)$ *and* $Q_2 \in TS(\epsilon_2)$, *if* $name(Q_1) = name(Q_2)$, *then* $\epsilon_1 = \epsilon_2$ *and* $Q_1 = Q_2$. *Given an identifier* τ, *we write* $proc(\tau)$ *for the unique* P *appearing in* TS *such that* $name(P) = \tau$.
2. ***Continuation markers:*** *given* $Q_1 = \texttt{op}^{\tau_i}(\texttt{tt @ r}).P \in TS(\epsilon_1)$ *and* $Q_2 \in TS(\epsilon_3)$ *such that* $name(P) = name(Q_2)$, *then there is* τ_o *such that* $\texttt{got}^{\tau_o,\tau_i} \in TS(\epsilon_2)$ *where* $\epsilon_1 \leq \epsilon_2 \leq \epsilon_3$.
3. ***Consistency:*** *if* $\texttt{got}^{\tau_o,\tau_i} \in TS(\epsilon_g)$, *then:*
   - *there exist* $P_x = proc(\tau_x) \in TS(\epsilon_x)$ *for* $x \in i, o$;
   - $P_i = \texttt{op}^{\tau_i}(\texttt{tt @ r}_i).P'$ *with* $\texttt{op} \in \{\texttt{in}, \texttt{rd}\}$ *and* $P_o = \texttt{out}^{\tau_o}(\texttt{t @ r}_o)$;
   - *there is* $\epsilon' \leq \epsilon_g$ *in* E *such that* $\texttt{r}_i(\epsilon_i, \epsilon')$ *and* $\texttt{r}_o(\epsilon_o, \epsilon')$ *both hold;*
   - *there exists a substitution* σ *such that* $\texttt{tt}[\sigma] = \texttt{t}$;
   - *for every* $\texttt{got}^{\tau_o,\tau'} \in TS(\epsilon')$ *with any* $\epsilon' \leq \epsilon_g$ *in* E, $kind(proc(\tau')) = \texttt{rd}$.
4. ***Atomicity:*** *if* $\texttt{got}^{\tau_o,\tau_i}$ *and* $\texttt{got}^{\tau_o,\tau_i'}$ *both appear in* TS *and* $kind(proc(\tau_i)) = kind(proc(\tau_i')) = \texttt{in}$, *then* $\tau_i = \tau_i'$.

The above properties state that (1) operation identifiers are globally unique; (2) a `got` process always exists between a `rd`/`in` request and it is unblocking; (3) a `rd`/`in` unblocks if there is a properly intersecting (in space-time and by tuple match) tuple (region); and finally (4) no pair of `in` can consume the same tuple.

3.4 Spatiotemporal tuple-based coordination

Note that the declarative specification of coherent tuple space evolutions just introduced does not hint at which underlying protocol may be used to respect



(a) Graphics of `out`s.

(b) Graphics of `rd`s.

(c) Graphics of interacting `in`s and `out`s.

Fig. 2: Graphics illustrating the spatiotemporal tuple operations.

it. Here, we propose a sample such protocol, via a high-level denotational description (i.e., at a global event structure level instead of a local interaction level) of the spatial processes that are created and propagated by each P, and how they interact together. This protocol assumes that every region $\mathtt{r}$ is equipped with *leader* predicate $lead^{\mathtt{r}}(\epsilon, \epsilon')$ which is satisfied for a set of events which is a subset of the region $lead^{\mathtt{r}}_{\epsilon} \subseteq \mathtt{r}_{\epsilon}$ and consists of a single chain of events: $lead^{\mathtt{r}}_{\epsilon} = \{\epsilon, \epsilon_1, \ldots, \epsilon_n\}$ where $\epsilon \rightsquigarrow \epsilon_1 \rightsquigarrow \ldots \rightsquigarrow \epsilon_n$.

Write (Figure 2a). Operation `out(t @ r)` in an event ϵ_{out} emits tuple `t` to spatio-temporal region `r`, corresponding to the set of events $\mathtt{r}_{\epsilon_{out}}$. The spatio-temporal extension of the emitted tuple is bounded to region `r` (light and dark areas together), which is a subset of the future event cone of ϵ_{out} (marked by the wiggled lines coming out from it). The actual region where a tuple is available may be smaller if a matching `in` operation occurs: tuple removal in event $\epsilon^{\dagger}_{out}$ results in removing the future event cone of $\epsilon^{\dagger}_{out}$ (darker area) from the availability region (lighter area). This behaviour can be implemented through a simple broadcasting process bounded to region `r`.

Read (Figure 2b). Operation `rd(tt @ r).`P reads, non-deterministically and in a blocking fashion, a tuple `t` matching template `tt`, situated in some spatio-temporal region $\mathtt{r}'$ intersecting with `r`. The operation is issued at event ϵ_{rd}, and propagates through a process within region `r` (blue). When the process enters the region $\mathtt{r}'$ (red) of an `out` operation with a tuple `t` matching template `tt`, notifications of the match(es) are propagated through broadcast in `r` (from

events such as ϵ_M). The first notification to reach the leader chain of `r` (horizontal dashed line) is accepted (in event $\epsilon^{\dagger}_{rd}$), leading to the termination of the *read* process, and starting the computation of the continuation process P in $\epsilon^{\dagger}_{rd}$.

Removal (Figure 2c). Operation `in(tt @ r)`.P fetches, non-deterministically and in a blocking fashion, a tuple `t` matching template `tt`, situated in some spatio-temporal region `r'` intersecting with `r`. This fetching leads to the termination of the `out` region, and no two different `in` operations are allowed to receive the same `t`. The operation is issued at event ϵ_{in}, and propagates through a process within region `r` (blue). Assuming the absence of partitions, atomicity and consistency can be guaranteed through a protocol involving a joint acknowledgement of the match between the two leaders of the `in` and `out` regions. This acknowledgement process is embodied by a chain of events of the form $\epsilon_{in} \leq \epsilon_M \leq \epsilon_C \leq \epsilon^{\dagger}_{in} \leq \epsilon^{\dagger}_{out}$. In ϵ_M a match is found: this is notified within region `r'` to reach for the leader chain (dashed lines) of the `out` operation in ϵ_C, which commits to the first arriving request (if any). The commitment is then broadcast through both regions `r'` and `r` until it reaches the leader chain of the `in` operation. This leader chain also commits to the first arriving request (in event $\epsilon^{\dagger}_{in}$) leading to the immediate termination of the `in` region and notifying the `out` region, which is reached in event $\epsilon^{\dagger}_{out}$ and terminates afterwards. It is possible that after the leader chain of an `out` operation commits to an incoming request (event ϵ_C), the leader chain of the accepted request does not accept the commitment, since it has already received another one. This can be detected by the `out`, given absence of partitions (cf. Section 6), in this case, the leader chain of the `out` operation erases its commitment, opening to new incoming requests. Notice that as multiple acknowledgements are necessary, concurrency of multiple `in` and `out` operations may lead to none of them being served.

4 Spatiotemporal Tuples as Aggregate Processes

The model in Section 3 founds on the idea that tuple operations are tasks that are *collective*, *adaptive* and *situated*, namely on-going, collaborative computations run by devices interacting in some spatial environment. In field calculi [32], the notion of *aggregate process* [15] has been recently proposed to capture dynamic, concurrent field computations and hence providing a programming abstraction for collective adaptive processes. In this section, we provide a brief recap of aggregate processes and their implementation in ScaFi (Section 4.1), then we describe an implementation of Spatiotemporal Tuples (Section 4.2) conforming to the model of Section 3—whereas its correctness and usability will be empirically evaluated in Section 5.

4.1 Aggregate processes

In the field calculus (FC) [6], the formal model backing *aggregate computing* [8,32], dynamic collective behaviour is modelled as a functional manipulation

of *computational fields* (i.e., maps from devices to values). A FC program encodes both computation and data exchange, and must be repeatedly evaluated by each device against its local context (sensor values, state, and received messages). The output of evaluation is a message to be broadcast to neighbours for coordination. So, when a device locally evaluates a FC expression, it can use data from its neighbours that also evaluated that very expression—a notion known as *alignment* [5]. In order to evaluate a dynamic number of expression, a mechanism is needed to properly deal with alignment. Also, whereas a FC program is generally run by every device in the system, and branching mechanisms exist to scope an expression on a partition of the system, dynamically controlling the evolution of the scope of an expression tends to be tricky. A recent proposal is to use a dedicated construct, called `spawn`, for generating and running dynamic field computations [15]. Also called *aggregate processes*, their idea is to align on a *process identifier* (like a *pid* in operating systems) and to let devices opt in/out their execution and control the spreading of the process to neighbours.

Aggregate processes can be programmed in ScaFi as follows.

```
// 1. Define an aggregate process function, fixing types for pids, arguments, return values
val process: Pid => Args => (Return, Boolean) = ???
// 2. Define a field of pids for processes to be locally instantiated
val pids: Set[Pid] = ??? // e.g., reading a sensor for user commands, or via a FC expression
// 3. A field of arguments for the active process instances
val args: A = ???
// 3. A spawn expression is like a VM for processes of some kind
val map: Map[Pid,Return] = spawn[Pid,Args,Return](process _, keys, args)
```

The above program is evaluated by every device of the system repeatedly in *rounds* of execution intervalled by sleeping periods where *coordination messages* are also exchanged between neighbours, asynchronously. The execution and interaction protocol is "fixed", and dynamically gives rise to an augmented event structure (cf. Figure 1); what changes is the payload of messages, which is a result of a local evaluation of the program—namely, the program itself defines both local behaviour and the data that needs to be exchanged for coordination. As the evaluation proceeds, aggregate processes will be generated and managed through `spawn`, and variable `map` will contain, at any device, the map of the locally active processes (their IDs and local outputs).

Suppose the goal is to let the devices of the system emit messages within some distance d from the emitter. Distances can be estimated by *gradients* [4], i.e., algorithms mapping a Boolean field of sources to the floating-point field of minimum distances from those sources. One gradient computation is not sufficient, because any device would compute the minimum distance from its nearest source. As dynamic field computations are required, aggregate processes can be used. If, at some round, `pids` is locally non-empty, corresponding processes are generated. Each time `spawn` is evaluated, function `process` is called for every aggregate process that is locally active—newly generated processes, those run (and preserved) in the previous round, or those acquired by neighbours. In this example, the `pids` can be tuples of the emitter ID and the message; the argument can be a field d of a distance threshold; and `process` can be defined as a field expression returning a tuple of (i) the message, and (ii) a Boolean stating

whether the gradient value from the emitter is lower or equal d. When a process is generated at the emitter device, it spreads as follows:

- the emitter evaluates the process expression; the gradient from itself is 0, so it yields the result and propagates the process pid to all its neighbours;
- an emitter's neighbour evaluates the process expression; it computes the gradient from the emitter; if its distance is lower or equal d, it yields the expression result and propagates the process pid to all its neighbours in turn, otherwise it drops the process (which could still be re-evaluated in future if some neighbour keeps propagating the same pid).

By repeated application of these steps, a d-radius bubble from the emitter holding the message is set-up, with devices 1-hop beyond it evaluating but dropping the process. Such a continuous evaluation of the process border is essential for its expansion and retraction—more details, also regarding process shutdown, can be found in [15]. This example is very similar to what an `out` tuple operation must be like, except that its aggregate process must close when interaction with another aggregate process (of an `in` operation) leads to removal of the tuple.

4.2 Implementing Spatiotemporal Tuples via Aggregate Processes

We argue that aggregate processes are a suitable abstraction for implementing the Spatiotemporal Tuples model because: (i) they are based on and extend the FC, which is space-time universal [2] and a premier computational model for systems situated in space and time; (ii) they enable an aggregate to run a dynamic number of tuple operations concurrently; (iii) they define a tuple operation as a collective adaptive process that is carried out collaboratively, in a decentralised and self-organising way; and (iv) they enable each tuple operation to have a dynamic scope that depends on its intended spatiotemporal situation. In the following, we describe the essential elements of this implementation.

The basic idea is to map a tuple operation to a corresponding aggregate process, and define how these should behave and interact. Most specifically:

- the problem of scoping a tuple operation to a certain spatiotemporal region is mapped to the problem of scoping its aggregate process (i.e., specifying a Boolean field to control what devices must opt in the aggregate process);
- the problem of matching tuple operations and non-deterministically selecting tuples is mapped to the problem of letting aggregate processes interact and reaching internal consensus;
- the problem of unblocking operations is mapped the problem of controlling the *lifetime* of aggregate processes.

An implementation sketch is given in Figure 3 (for the full sources, refer to the repository provided in Section 5). For details on the FC/ScaFi language, refer to [14,16]. The idea is to let the tuple operation processes evolve in collective behaviour phases (like in a state-machine), commanded by the leader, ensuring "transactional semantics", and to use messages (appended via `<<` to the phase

```
def tupleOp(op: O)(arg: Map[O,R]): (R, Status) = op match {
  case Out(...) => outLogic(...); case In(...) => inLogic(...); ...
}
rep[Map[O,R]](Map.empty())(ops => spawn(tupleOp _, newLocalOps(), ops))
```

(a) ScaFi field-calculus expression for spawning and executing tuple operations.

```
def outLogic(pid,t,r,ops,owner) = {
  val inside = computeRegion(r)
  val p = workflow(Available){
    case curr @ Available => {
      val requests = C(owner, ∪, request())
      val choice = requests.headOption()
      if(owner && choice.isPresent)
        Serving(choice.get) else curr
    }
    case curr @ Serving(inP) => {
      val ack = gossip(inOwnerAck())
      (if(owner && ack) Done(inP) else curr)
        << Msg(ReservedFor(pid,inP))
    }
    case curr @ Done(inP) =>
      { curr << OutAck(pid,inP)  }
  }
  (R(t,p.msgs), runOrNot(inside,p))
}
```

(b) Excerpt of ScaFi code for `outs`.

```
def inLogic(pid,tt,r,ops,owner) = {
  val inside = computeRegion(r)
  val p = workflow(Waiting){
    case curr @ Waiting => {
      val offers = C(owner, ∪, outOffers(ops))
      val choice = offers.headOption()
      (if(owner && choice.isPresent)
        Removing(choice.get) else curr)
        << Msg(Request(pid,choice))
    }
    case curr @ Removing(outP) => {
      val ack = gossip(outOwnerAck())
      (if(owner && ack) Done(outP) else curr)
        << Msg(InAck(pid,outP))
    }
    case Done(outP) => { /* no-op */ }
  }
  (R(p.tupleIfAny(),p.msgs),runOrNot(inside,p))
}
```

(c) Excerpt of ScaFi code for `ins`.

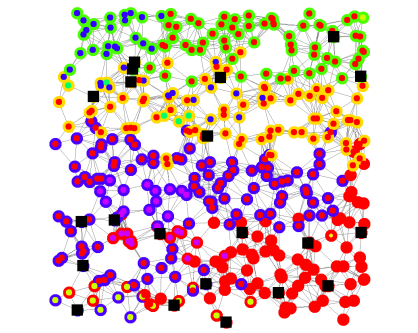
Fig. 3: (Pseudo-)Implementation of tuple operations as ScaFi processes.

descriptor) to let *aggregate processes within an individual device* to interact. As shown in Figure 3a, each aggregate process receives the overall map of processes and corresponding results (`ops`, remembered from round to round via `rep`) as an argument. Results `R` include a tuple (if any) and the messages. Function `computeRegion` uses region description `r` to call a proper ScaFi function yielding a Boolean field which is `true` only for the nodes that should belong to the region. E.g., if `r` denotes the region within a range ρ from the leader, then field expression `gradient(mid()==leader)<=`ρ is computed (where `mid()` returns the node ID); or, if `R` is a geographic area, then `GPScoordinates()` $\in$ `R` would do the job. Note that devices opt for the process (`runOrNot`) based on `inside` and phase `p`.

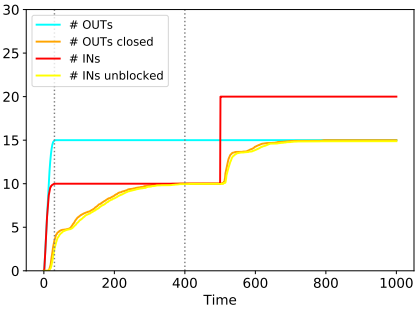
5 Evaluation

With the goal of checking correctness in dynamic environments, and to discuss applicability, in this section we evaluate the presented model and implementation by means of simulation, through synthetic experiments (Section 5.1) and a rescue case study (Section 5.2). For the simulations, we leverage the ScaFi incarnation [13] of the Alchemist simulator [29]. Source code, tools, and instructions for reproducing the experiments can be found in the attached public repository[5].

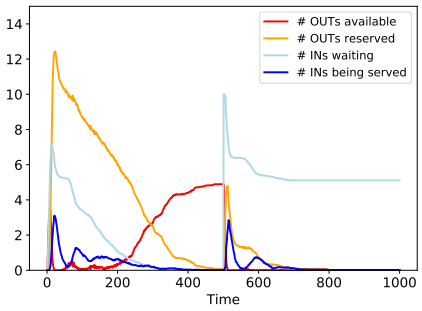
[5] `https://github.com/metaphori/experiment-2021-spatiotemporaltuples`



(a) A graphical view of the scenario as simulated in the ScaFi-Alchemist framework. The colours are used to denote different tuple operations (aggregate processes), though actually a single node may run several of them concurrently. The smaller coloured dots denote `out` processes, while the larger halos denote `in` processes. The black square symbols denote the devices that generated any tuple operation.



(b) Evolution in time of the number of `out`s and `in`s spawned and closed.



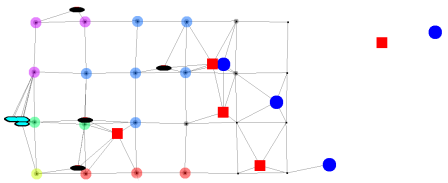
(c) Evolution in time of the number of `out`s and `in`s in the different phases.

Fig. 4: Evaluation

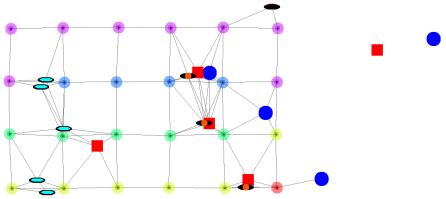
5.1 Simulation-based Evaluation

Setup The scenario is shown in Figure 4a. We configure a square arena with 400 mobile devices displaced in a $1km^2$ grid: they interact with neighbours within a 100-metre connectivity range, and compute the ScaFi program asynchronously about once per second. We let the devices generate random tuple operations, either `out` or `in` operations, such that the system is engaged in multiple, concurrent operations that need to be carried out. Moreover, we generate tasks so that for $t < 400$ there are more `out`s than `in`s in the system, and for $t > 400$ there are more `in`s than `out`s. These operations have an extension of 450 metres to promote high contention and are generated so that they intersect (otherwise they could not be matched). Then, we monitor the evolution of the system. We keep track of the number of spawned operations and terminated operations, as well as the phases that these transit on. We expect that a single `in` operation pairs up with a single `out` operation, and vice versa. We perform 100 runs of the system, with different random seeds affecting the actual positions of nodes, the relative scheduling of the devices, and which devices generate the operations.

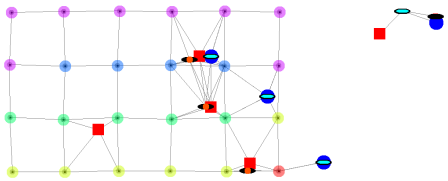
Results The results, averaging the data produced in the 100 runs, are shown in Figure 4. In Figure 4b, we observe that the number of `out`s that get closed



(a) Explorers start exploring and depositing breadcrumbs.



(b) Some explorers hit obstacles and become impaired.



(c) The followers managed to reach the victims.

Fig. 5: Snapshots of the simulated case study. Notation: black discs (explorers), cyan discs (followers), large blue dots (victims), large red squares (obstacles), small black dots (tactical network nodes), coloured halos (breadcrumbs), orange dots on the discs (impairment). (Larger pictures are available at the provided repository)

and the number of `in`s that are satisfied grow equally. All the operations are satisfied, provided there is still a matching one in the system, a condition that does not hold approximately for $t \in [200, 400]$ and $t > 550$ when no more `in` and `out` processes are alive, respectively. In Figure 4c, we observe how the processes transit from their "waiting a match" phase to their "match found" phase. In particular, notice how the `out` processes immediately offer their tuple to a matching `in`, but as the `in`s are exhausted they become available again (approximately for $t = [150, 400]$).

5.2 Case Study: Rescue Scenario with Breadcrumbs

Like for Spatial Tuples, the Spatiotemporal Tuple model can be adopted in applications requiring various forms of spatial coordination. Example application scenarios providing motivation for this coordination approach can be found in [30]. A benefit of Spatiotemporal Tuples is that it streamlines decentralised implementation of such coordination patterns in logically ad-hoc or peer-to-peer networks.

Here, we consider a simple rescue scenario: a set of rescuers have to explore a territory to find victims needing assistance; in the area, however, there are some hazardous elements (e.g., mines or blocks) that may impair the rescuers. The area is partially covered by a tactical mesh network. We consider two teams: explorers and followers. According to the *breadcrumb pattern* [30], the explorers navigate the area and, from time to time, leave a spatiotemporally-tagged tuple (a "breadcrumb") at their location to keep track of their paths. Some time later, the followers begin their expedition: they move by following the breadcrumbs left by the explorers; however, if the breadcrumbs-path interrupts, they

take a random detour and, after that, start exploring in turn. Screenshots of the different phases of the simulation are in Figure 5, showing how the spatial coordination pattern, backed by spatiotemporal tuples, allows the rescuers to succeed in reaching the victims (assuming no further obstacles impair them).

6 Conclusion and Future Work

In this paper, we propose a model for spatiotemporal tuples where tuple operations run on a *computational space-time structure* that *logically* bridges the situation domain with the computation domain. This choice has a twofold benefit: it enables locality and scalability of the tuple-based system and promotes straightforward implementation in the aggregate computing paradigm.

Finally, we discuss the following aspects, to be fully investigated in the future.

- *Properties and guarantees of the model and its implementations.* Basic properties of the model are given in Definition 5: these ensure safety and liveness of spatiotemporal tuple operations. A benefit of the proposed model with respect to Spatial Tuples [30] is that it provides a convenient basis for decentralised implementations where the tuple space is fragmented in a collection of local tuple spaces owned by the individual devices. It also promotes *scalability* through locality of tuples and operations: only the devices situated in the spatial region of a tuple operation would execute the aggregate process sustaining that operation. So, what about sparse networks or positioning tuples in areas not covered by any device? The idea is that a device should be aware of what tuple operations are where: it is sufficient that it knows the aggregate process IDs and it will play them once it belongs to their spatial region. So, for scalability, a decentralised middleware solution could propagate those IDs to a larger spatial region (still smaller than the entire application space), hence exploiting locality while ensuring operations are not lost. Moreover, a distributed implementation of the Spatiotemporal Tuple model has to decide how to deal with the *CAP theorem* [10], i.e., what kind of consistency and availability guarantees to provide when facing failure and network partitions. For instance, the relative level of consistency and availability (e.g., by introducing time-outs or priorities) might affect scalability [9]. Design decisions should be taken according to the levels of contention, variability (as induced by mobility, failure), and operation rates.
- *Spatiotemporal property verification and monitoring.* Potential for combining the coordination language with spatial and temporal logics for verification and monitoring, along the lines of [3], could also be investigated. Beside distributed runtime verification, statistical spatio-temporal model checking [18] may be adopted for verifying implementations in simulated settings.
- *Generality of the model w.r.t. deployments.* A major merit of the approach is that it supports both centralised, infrastructure-based deployments (cf. cloud- or server-based systems) and decentralised, infrastructureless deployments (cf. MANETs). Indeed, since aggregate computing systems can be

partitioned into different deployment units (a notion also known as *pulverisation* [12]), applications can exploit available infrastructure and hosts to promote different levels of performance and CAP guarantees.

References

1. Alrahman, Y.A., Nicola, R.D., Loreti, M.: Programming interactions in collective adaptive systems by relying on attribute-based communication. Sci. Comput. Program. **192**, 102428 (2020). https://doi.org/10.1016/j.scico.2020.102428
2. Audrito, G., Beal, J., Damiani, F., Viroli, M.: Space-time universality of field calculus. In: COORDINATION 2018. pp. 1–20 (2018). https://doi.org/10.1007/978-3-319-92408-3_1
3. Audrito, G., Casadei, R., Damiani, F., Stolz, V., Viroli, M.: Adaptive distributed monitors of spatial properties for cyber-physical systems. J. Syst. Softw. **175**, 110908 (2021). https://doi.org/10.1016/j.jss.2021.110908
4. Audrito, G., Casadei, R., Damiani, F., Viroli, M.: Compositional blocks for optimal self-healing gradients. In: Self-Adaptive and Self-Organizing Systems (SASO), 2017 IEEE 11th International Conference on. pp. 91–100. IEEE (2017)
5. Audrito, G., Damiani, F., Viroli, M., Casadei, R.: Run-time management of computation domains in field calculus. In: Foundations and Applications of Self* Systems, IEEE International Workshops on. pp. 192–197. IEEE (2016)
6. Audrito, G., Viroli, M., Damiani, F., Pianini, D., Beal, J.: A higher-order calculus of computational fields. ACM Trans. Comput. Logic **20**(1), 5:1–5:55 (Jan 2019). https://doi.org/10.1145/3285956
7. Beal, J., Dulman, S., Usbeck, K., Viroli, M., Correll, N.: Organizing the aggregate: Languages for spatial computing. In: Formal and Practical Aspects of DSLs: Recent Developments, chap. 16, pp. 436–501. IGI Global (2013), a longer version at: `http://arxiv.org/abs/1202.5509`
8. Beal, J., Pianini, D., Viroli, M.: Aggregate programming for the Internet of Things. IEEE Computer **48**(9), 22–30 (2015). https://doi.org/10.1109/MC.2015.261
9. Boix, E.G., Scholliers, C., De Meuter, W., D'Hondt, T.: Programming mobile context-aware applications with TOTAM. J. Syst. Softw. **92**, 3–19 (2014)
10. Brewer, E.: Cap twelve years later: How the "rules" have changed. Computer **45**(2), 23–29 (2012)
11. Busi, N., Gorrieri, R., Zavattaro, G.: On the expressiveness of linda coordination primitives. Inf. Comput. **156**(1-2), 90–121 (2000)
12. Casadei, R., Pianini, D., Placuzzi, A., Viroli, M., Weyns, D.: Pulverization in cyber-physical systems: Engineering the self-organizing logic separated from deployment. Future Internet **12**(11), 203 (2020)
13. Casadei, R., Pianini, D., Viroli, M.: Simulating large-scale aggregate MASs with alchemist and scala. In: Computer Science and Information Systems (FedCSIS), 2016 Federated Conference on. pp. 1495–1504. IEEE (2016)
14. Casadei, R., Viroli, M., Audrito, G., Damiani, F.: FScaFi: A core calculus for collective adaptive systems programming. In: 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA, Proceedings. LNCS, vol. 12477, pp. 344–360. Springer (2020). https://doi.org/10.1007/978-3-030-61470-6_21
15. Casadei, R., Viroli, M., Audrito, G., Pianini, D., Damiani, F.: Aggregate processes in field calculus. In: COORDINATION 2019. pp. 200–217 (2019). https://doi.org/10.1007/978-3-030-22397-7_12

16. Casadei, R., Viroli, M., Audrito, G., Pianini, D., Damiani, F.: Engineering collective intelligence at the edge with aggregate processes. Eng. Appl. Artif. Intell. **97**, 104081 (2021)
17. Casadei, R., Viroli, M., Ricci, A.: Collective adaptive systems as coordination media: The case of tuples in space-time. In: 1st IEEE International Conference on Autonomic Computing and Self-Organizing Systems, ACSOS, Companion Volume. pp. 139–144. IEEE (2020). https://doi.org/10.1109/ACSOS-C51401.2020.00045
18. Ciancia, V., Latella, D., Massink, M., Paskauskas, R., Vandin, A.: A tool-chain for statistical spatio-temporal model checking of bike sharing systems. In: Margaria, T., Steffen, B. (eds.) 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10-14, 2016, Proceedings, Part I. LNCS, vol. 9952, pp. 657–673 (2016). https://doi.org/10.1007/978-3-319-47166-2_46
19. Coulouris, G., Dollimore, J., Kindberg, T.: Distributed systems - concepts and designs (3. ed.). Addison-Wesley-Longman (2002)
20. DeHon, A., Giavitto, J., Gruau, F. (eds.): Computing Media and Languages for Space-Oriented Computation, 03.09. - 08.09.2006, Dagstuhl Seminar Proceedings, vol. 06361 (2007), `http://drops.dagstuhl.de/portals/06361/`
21. Gelernter, D.: Generative communication in linda. ACM Transactions on Programming Languages and Systems (TOPLAS) **7**(1), 80–112 (1985)
22. Gelernter, D.: Multiple tuple spaces in linda. In: International Conference on Parallel Architectures and Languages Europe. pp. 20–27. Springer (1989)
23. Loo, J., Mauri, J.L., Ortiz, J.H.: Mobile ad hoc networks: current status and future trends. CRC Press (2016)
24. Mamei, M., Zambonelli, F.: Programming pervasive and mobile computing applications: The TOTA approach. ACM Trans. on Software Engineering Methodologies **18**(4), 1–56 (2009). https://doi.org/http://doi.acm.org/10.1145/1538942.1538945
25. Menezes, R., Wood, A.: The fading concept in tuple-space systems. In: Proceedings of the 2006 ACM symposium on Applied computing. pp. 440–444 (2006)
26. Merrick, I., Wood, A.: Scoped coordination in open distributed systems. In: COORDINATION 2000. pp. 311–316. Springer (2000)
27. Murphy, A.L., Picco, G.P., Roman, G.C.: LIME: A coordination model and middleware supporting mobility of hosts and agents. ACM Transactions on Software Engineering and Methodology **15**(3), 279–328 (Jul 2006). https://doi.org/10.1145/1151695.1151698
28. Pauty, J., Couderc, P., Banatre, M., Berbers, Y.: Geo-linda: a geometry aware distributed tuple space. In: 21st International Conference on Advanced Information Networking and Applications (AINA'07). pp. 370–377. IEEE (2007)
29. Pianini, D., Montagna, S., Viroli, M.: Chemical-oriented simulation of computational systems with ALCHEMIST. J. Simulation **7**(3), 202–215 (2013). https://doi.org/10.1057/jos.2012.27
30. Ricci, A., Viroli, M., Omicini, A., Mariani, S., Croatti, A., Pianini, D.: Spatial tuples: Augmenting reality with tuples. Expert Systems **35**(5), e12273 (2018)
31. Tolksdorf, R., Menezes, R.: Using swarm intelligence in Linda systems. In: Omicini, A., Petta, P., Pitt, J. (eds.) Engineering Societies in the Agents World IV. LNCS, vol. 3071, pp. 49–65. Springer (2004)
32. Viroli, M., Beal, J., Damiani, F., Audrito, G., Casadei, R., Pianini, D.: From distributed coordination to field calculus and aggregate computing. Journal of Logical and Algebraic Methods in Programming **109**, 100486 (2019). https://doi.org/10.1016/j.jlamp.2019.100486

33. Viroli, M., Pianini, D., Beal, J.: Linda in space-time: An adaptive coordination model for mobile ad-hoc environments. In: COORDINATION 2012. pp. 212–229 (2012). https://doi.org/10.1007/978-3-642-30829-1_15
34. Winskel, G.: An introduction to event structures. In: Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, School/-Workshop, Proceedings. LNCS, vol. 354, pp. 364–397. Springer (1988). https://doi.org/10.1007/BFb0013026