Collective Adaptive Systems as Coordination Media: The Case of Tuples in Space-Time

Roberto Casadei, Mirko Viroli, Alessandro Ricci

ALMA MATER STUDIORUM—Università di Bologna, Cesena, Italy
Email: {roby.casadei, mirko.viroli, a.ricci}@unibo.it

Abstract—**Coordination is a fundamental problem in the engineering of collective adaptive systems (CAS). Prominent approaches in this context promote adaptivity and collective behaviour by founding coordination on local, decentralised interaction. This is usually enabled through abstractions such as collective interfaces, neighbour-based interaction, and attribute-based communication. Application designers, then, use such coordination mechanisms to enact collective adaptive behaviour in order to solve specific problems or provide specific services while coping with dynamic environments. In this paper, we consider the other way round: we argue that a CAS model can be used to provide support for high-level coordination models, simplifying their implementation and transferring to them the self-* properties it emergently fosters. As a motivating example, we consider the idea of supporting tuple-based coordination by Linda primitives such that tuples and operations have a position and extension in space and time. Then, we adopt an aggregate perspective, by which space-time is logically represented by a mobile ad-hoc network of devices, and show that coordination primitives can be implemented as true collective adaptive processes. We describe this model and a prototype implementation in the ScaFi aggregate programming framework, which is rooted in the so-called computational field paradigm.**

Index Terms—**collective adaptive systems, self-* coordination, spatial tuples, aggregate processes, space-time programming**

I. Introduction

Interaction is a fundamental aspect in software systems in general and in Collective Adaptive Systems (CAS) in particular. Indeed, by interacting, the entities of a collective can coordinate their activity to promote emergence of robust, global-level behaviour. Therefore, when it comes to engineering CAS, *coordination* (i.e., the ruling of interaction) [1] is a prominent problem to be tackled. Interaction in CASs, as observed in several natural systems (such as ant colonies [2]), is typically local, decentralised, and repeated in time in order to provide reactivity to environmental change (cf. self-adaptation and self-organisation). Inspired by these features, engineering approaches propose coordination mechanisms that leverage abstractions such as collective interfaces [3], neighbour-based interaction [4], and attribute-based communication [5]. Together with these mechanisms, other operators are generally given to application designers to develop collective adaptive behaviour in order to solve specific problems (e.g., edge resource management [6]–[8]) or provide specific services (e.g., smart mobility [4], [9]).

In this position paper, we argue and show that a CAS model, by fostering emergence of self-* properties, can be used to provide support for high-level coordination models, e.g., simplifying their implementation, transferring self-* properties and/or relaxing model assumptions. In other words, the idea is to shift from the view of "coordination for CAS" to the view of "CAS for coordination". In particular, we draw inspiration from three related works:

- the *Spatial Tuples* model, where agents interact by emitting and retrieving tuples situated *in space* (hence extending over plain tuple-based coordination);
- *self-organising coordination* [10], whereby coordination media are spread over the network to locally regulate interaction and foster emergence of global properties;
- aggregate computing approaches [4], [11], and especially the notion of *aggregate processes* [12], which model concurrent collective adaptive processes in a functional framework of computational fields.

Specifically, we adopt an aggregate perspective to self-organising coordination, by which space-time is logically represented by a mobile ad-hoc network of devices, and show how Spatial Tuples coordination primitives can be implemented as aggregate processes (i.e., concurrent field computations). As a result, we provide a collective adaptive support to the ability of injecting tuples in a space-time region covered by mobile devices, and retrieve them by pattern matching through intersection with the space-time of search.

The paper is organised as follows. Section II provides background and discusses related work in the area of tuple-based and space-based coordination. Section III describes a model for tuple-based coordination in space-time. Section IV drafts an implementation of the model in terms of aggregate processes. Section V provides some discussion. Finally, Section VI provides conclusion and future work.

II. Background and Related Work

In this section, we briefly introduce tuple-based coordination and survey variants and extensions of the basic model providing dedicated support for systems situated in space-time.

A. Tuple-based coordination

In *tuple-based coordination*, a set of *processes* coordinate indirectly by writing and reading tuples on a *shared tuple space* [1]. A *tuple* is an ordered collection of (possibly heterogeneous) values. Tuples are contained within a *tuple space*, and are accessed *associatively*, i.e., by matching on their content or structure. This coordination approach leverages so-called *generative communication*, whereby the "life" of generated data is independent of the "life" of the generator, and, crucially, enables *space decoupling* (two processes do not need to be spatially co-located to interact), *time decoupling* (two processes do not need to be temporally co-located to interact), and *name decoupling* (two processes do not need to know each other to interact). *Linda* is the progenitor tuple-based coordination language, and is often extended to provide specific features. When centralised tuple space approaches fall short, especially in distributed settings, an idea is to break the global tuple space into multiple local tuple spaces [13].

B. Tuples in pervasive networks and space(-time)

Tuple-based coordination approaches have been also used in the context of peer-to-peer (P2P) and mobile ad-hoc networks (MANETs), where there is no pre-existing infrastructure and devices interact opportunistically via short-range wireless technology. These scenarios share various characteristics – mobility, dynamicity, locality, openness – that challenge (the implementations of) the basic Linda model. Systems for tuple-based coordination in these contexts usually come with a middleware, dealing with certain issues related to distribution and mobility, and extensions to the basic tuple-space model and Linda language to support specific aspects, e.g., related to location management. Moreover, since networks of devices can be physically situated (cf., MANETs), it comes natural to extend models with first-class space and time abstractions.

In *$\sigma\tau$-Linda* [14] agents interact by injecting processes where Linda operations can be combined with network constructs like $next$ (to delay an operation to the next computation round in a device) $neigh$ (to propagate an operation to all the neighbour agents), and $finally$ (to execute an operation once other operations have completed). Then, the authors show an extension to standard Linda operations where these are spatio-temporally situated: each operation yields a waveform-like activity with a limited spatiotemporal extension. *LIME* [15] deals with both *physical* mobility of *hosts* (changing the actual network topology) and *logical* mobility of *agents* across hosts. Each agent holds a local tuple space and can see the tuples of agents connected to it. In *TOTA* [16], tuples are associated with *propagation rules* that describe how tuples should be propagated (hop-by-hop) in a network and how the content of tuples should change during propagation. *GeoLinda* [17] provides geometry-aware distributed tuple spaces where both tuples and reading operations have a spatial extension, or *volume*, called *tuple shape* or *addressing shape*, respectively.

C. Spatial Tuples

The *Spatial Tuples* coordination model [18] provides a synthesis of tuple-based and space-based coordination. In this approach, a tuple is decorated with spatial information, effectively situating the tuple to some *point-wise location* or *spatial region*—in the latter case also intending that the tuple has a *spatial extension*. The Spatial Tuples *language* provides linguistic constructs to work with spatial tuples: it consists of a *communication language* for expressing tuples and tuple templates (for matching), a *space description language* for expressing spatial information, and a *coordination language* for expressing interaction. Some of the key spatial primitives defined in [18] are the following ones:

- `out(t @ r)` — emits a spatial tuple `t` to a region or location `r`.
- `rd(tt @ rt)` — reads, non-deterministically and in a blocking fashion, a spatial tuple `t` matching template `tt` in any location `r` matching region template `rt`.
- `in(tt @ rt)` — fetches, non-deterministically and in a blocking fashion, a spatial tuple `t` matching template `tt` in any location `r` matching region template `rt`.

The space description language is an orthogonal aspect that depends on the application. However, the location of a tuple can also be specified *indirectly* by referring to the location of a situated component identified by `id`: `t @ id`. This creates a *binding* between the locations that is particularly useful for *mobility*. Implicit forms also exist: `t @ here` takes the current location of the executing component; and `t @ me` binds the location of `t` to the location of the executing component.

Using this approach, space-oriented coordination can be implemented, e.g., in terms of *situated communication*, where communicating parties leave and perceive messages at specific locations, and *spatial synchronisation*, where actions of multiple agents are ordered based on their spatial situation.

III. A Model for Spatiotemporal Tuple-based Coordination

The Spatial Tuples model bridges the digital and physical worlds by situating data and supporting space-oriented coordination. However, there are three open issues. First, the model abstracts from the actual notion of space, which is typically assumed to be global and serves only as a labelling/matching mechanism. Second, it neglects time and dynamic aspects such as evolving tuples à la TOTA [16]. Third, it is not straightforward to come up with a fully distributed (scalable and resilient) implementation suitable for infrastructures that include an important peer-to-peer component (MANETs, edge computing, wireless sensor networks, etc.). So, in the following, we propose a variant coordination model, which we call the *Spatiotemporal Tuples* model, that is based on a notion of "computational space-time" whose evolution and coordination primitives are enacted by a collective adaptive system.

A. Design Concerns

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

- **Space** — The model should capture spatial situation, while providing a suitable abstraction over the notion of space, to possibly capture diverse situations. Most specifically, we mean to provide a *computational* notion of space, where space locations are associated with (logical or physical) computational devices, and proximity of locations matches the ability of a device to directly perceive its context, there including reception of messages.

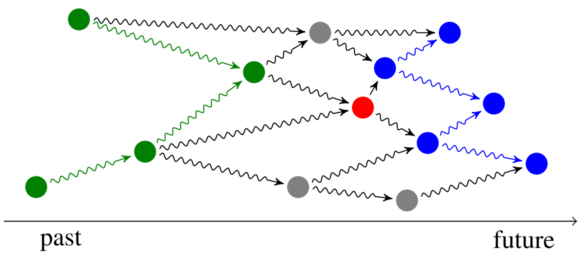
- **Time** — The model should dually capture temporal situation, while abstracting over the notion of time and hence of system evolution. Also, as we specifically target fully distributed systems, in which there is generally no notion of global time [19], [20], the model should provide the expressiveness to specify what and how notions of local time can be used and can be propagated.
- **Safety and CAP** — The model should adhere to the general Linda semantics [21], though extension with space-time can provide additional features. Moreover, when implementing the model in a distributed system, the *CAP theorem* [22] enters the picture, asserting that you may pick only two among the three properties: consistency, availability, and partition tolerance.
- **Heterogeneous deployments** — The model should foster implementation for different kinds of infrastructures, such as MANETs, P2P networks, edge and cloud-based architectures. That is, it should be sufficiently *general* to capture diverse settings, considering the specific relationships and constraints of modern distributed systems.

B. Computational space-time model

To define a model for Linda primitives in a spatiotemporal context, namely, where there is need of tracking propagation of information in space and time, we need to adopt a suitable underlying notion of computability. Accordingly, we based our model on the notion of space-time computability by Audrito et al. as of [23], which itself is based on the *event structure* framework. We here recall, and then extend this framework.

Definition 1 (Event Structure (taken from [23]). *An event structure* $\mathbf{E} = \langle E, \rightsquigarrow, < \rangle$ *is a countable set of events* E *together with a neighbouring relation* $\rightsquigarrow \subseteq E \times E$ *and a causality relation* $< \subseteq E \times E$*, such that the transitive closure of* $\rightsquigarrow$ *forms the irreflexive partial order* $<$ *and the "causal past" set* $\{\epsilon' \in E \mid \epsilon' < \epsilon\}$ *is finite for each* ϵ *(i.e.,* $<$ *is locally finite). We call* **ES** *the set of all such event structures.*

Consider the following example of event structure.



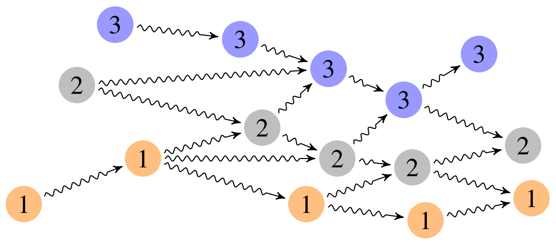
The red node is a *reference event* ϵ. The green nodes are the events ϵ' belonging to the *causal past* of ϵ (i.e., $\epsilon' < \epsilon$); the gray nodes are the events belonging to the *present* of ϵ (i.e., ϵ' and ϵ are unordered, or *concurrent*); the blue nodes are the events belonging to the *causal future* of ϵ (i.e., $\epsilon < \epsilon'$).

To specialise this model towards computational systems, we introduce a concept of *identity* across events and annotations to quantify local space and local time.

Definition 2 (Event Structure with Identities). *Let* $\mathbf{E}$ *be an event structure,* $\mathcal{L}$ *be a set of labels (identifiers),* $\mathcal{I} : E \to \mathcal{L}$ *be a function from events to labels, and* $E_i = \{\epsilon \mid \mathcal{I}(\epsilon) = i\}$ *denote the subsets of events* $\epsilon^{[i]}$ *with identity* i*. An* Event Structure with Identities *is a triple* $\mathbf{E_I} = \langle \mathbf{E}, \mathcal{L}, \mathcal{I} \rangle$ *such that, for any identity* i*, the event structure restricted to* E_i *consists of a single chain, i.e., a path* $\epsilon_1 \rightsquigarrow_i^* \epsilon_n$*. As a consequence, the set* E *of events is split by labels* ℓ *into contiguous partitions* E_ℓ*, induced by* $\mathcal{I}$*, denoting* processes*. The neighbouring relation* $\rightsquigarrow \subseteq E \times E$ *is also partitioned into* $\rightsquigarrow_i$: $E_i \to E_i, \forall i \in \Im(\mathcal{I})$ *(in the image of* $\mathcal{I}$*) and* $\rightsquigarrow_{ij}$: $E_i \to E_j, \forall i, j \in \Im(\mathcal{I}), i \neq j$*.*

The intuition of this definition is that identities represent devices, a neighbouring relation between events with same identity represents one-step passage of time, while one between events with different identity represents proximity in space (and ability to directly interact).

Consider the following example of event structure with identities $\mathcal{L} = \{1, 2, 3\}$, where different colours are used to better visualise different processes.



Definition 3 (Computational Space-Time Structure). *Let* M_T *be a (partially defined) time metric with default* $\bot_t$ *and* M_S *be a (partially defined) spatial metric with default* $\bot_s$*. A* Computational Space-Time Structure **ST** *is an event structure with identities* $\mathbf{E_I}$*, where edges* $\rightsquigarrow (\epsilon, \epsilon')$ *are annotated with a tuple* $\langle M_T(\epsilon, \epsilon'), M_S(\epsilon, \epsilon') \rangle$*, further restricted so that:*

- *edges* $\rightsquigarrow_i (\epsilon, \epsilon')$ *are annotated with a temporal distance* $M_T(\epsilon, \epsilon')$ *and* $\bot_s$*;*
- *edges* $\rightsquigarrow_{ij} (\epsilon, \epsilon')$ *are annotated with a spatial distance* $M_S(\epsilon, \epsilon')$ *and* $\bot_t$*.*

Therefore, in this structure, a label $\ell \in \mathcal{L}$ denotes both a different *spatial locality* and a different process; an event ϵ represents both a particular *(space-time) locality* and a computational step; a chain of localities $\epsilon_1 \rightsquigarrow \ldots \rightsquigarrow \epsilon_n$ with the same label ℓ (explicitly written $\epsilon_1^{[\ell]} \rightsquigarrow \ldots \rightsquigarrow \epsilon_n^{[\ell]}$) represents both a *(space-time) evolution* and a process execution.

In this model, tuples are spatiotemporally situated in some spatiotemporal region.

Definition 4 (Computational Spatiotemporal Region). *Let* **ST** *be a computational space-time structure. We define a* Computational Spatiotemporal Region *in* **ST** *as a subset* $R \subseteq E$ *of the space-time localities.*

Definition 5 (Spatiotemporal Causality Structure). *A* Spatiotemporal Causality Structure *is a computational spatiotem-*

poral structure **ST** *with a distinguished unique element* $\epsilon_\perp$ *which is the* $<$*-minimum in* **E**, *i.e., such that* $\forall \epsilon \in E, \epsilon_\perp \leq \epsilon$.

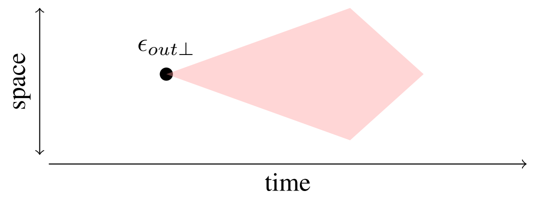
Definition 6 (Spatiotemporal Tuple Region)**.** *A* Spatiotemporal Tuple Region *is a computational spatiotemporal region of a spatiotemporal causality structure where* $\epsilon_\top$ *is the locality in which the tuple is issued.*

A spatiotemporal tuple region describes the maximum region in which a tuple may exist (since removal operations may reduce its lifetime). As we will see in Section IV, a spatiotemporal region can also be represented as a Boolean *space-time value* computed by a *space-time function* [23].

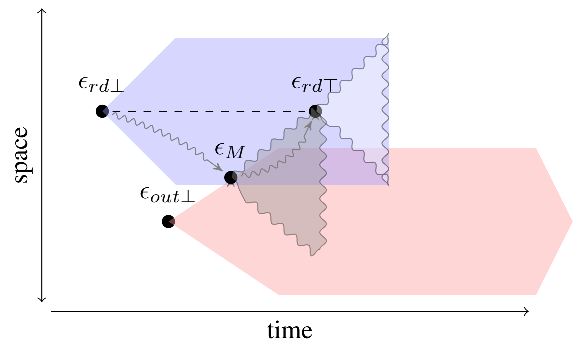
C. Spatiotemporal tuple-based coordination

The primitives are similar to those in Spatial Tuples as described in Section II-C. However, the space description language becomes a *space-time description language*, and the semantics is adjusted to operate on a computational space-time structure. The semantics is informally described as follows, and implemented as covered in Section IV.

a) Write: Operation `out(t @ r)` emits tuple `t` to spatiotemporal tuple region `r` from a starting event $\epsilon_{out\perp}$. Note that this region may or may not be finite.



b) Read: Operation `rd(tt @ rt)` reads, non-deterministically and in a blocking fashion, a tuple `t`, situated in a spatiotemporal tuple region `r`, matching template `tt` which is also situated in region `rt`. The operation is issued by a process i (owner) at locality $\epsilon_\perp^{[i]}$. The process will be able to read the tuple at some event $\epsilon_\top^{[i]}$, following evolution $\epsilon_\perp^{[i]} \rightsquigarrow^* \epsilon_\top^{[i]}$, iff there is a path $\epsilon_\perp^{[i]} \rightsquigarrow^* \epsilon_M \rightsquigarrow^* \epsilon_\top^{[i]}$ where ϵ_M is a "matching event" belonging to both regions `r` and `rt`.



In the figure, the convention is to use "causal cone"-like shapes to indicate the spatiotemporal scope of activities originating at particular space-time locations (denoted by named black dots); sharp cones denote tuple operation processes, whereas curly-bordered ones denote truncated causal future cones originating at some bottom event.

c) Removal: Operation `in(tt @ rt)` fetches, non-deterministically and in a blocking fashion, a tuple `t`, situated in a spatiotemporal tuple region `r`, matching template `tt` which is also situated in region `rt`. Assuming the absence of partitions, atomicity and consistency can be guaranteed through a protocol involving a path $\epsilon_\perp \rightsquigarrow^* \epsilon_M \rightsquigarrow^* \epsilon_C \rightsquigarrow^* \epsilon_R \rightsquigarrow^* \epsilon_A \rightsquigarrow^* \epsilon_\top$ where $\epsilon_\perp$ is the initiator locality where the `in` is emitted, ϵ_M is the locality where the tuple and tuple template regions match, ϵ_C is the locality where consensus is reached on the tuple to remove, ϵ_R is the locality where the initiator receives the tuple, ϵ_A is the locality where the reception acknowledgment is received (defining a spatiotemporal causality structure where the tuple is absent everywhere), and $\epsilon_\top$ is the locality where the initiator unblocks.

In case of multiple `out`s, event ϵ_R is responsible to initiate a situated activity to choose and acknowledge only one of them (i.e., leading to a single ϵ_A). The suggested semantics is meant to ensure consistency; the problem with partitions and liveness is discussed in Section V.

IV. Spatiotemporal Tuples as CAS Processes

In this section, we propose a design and implementation of Spatiotemporal Tuples that straightforwardly follows the model provided in Section III. Specifically, we represent tuple operations as *live collective processes* on a computational space-time structure given by a (logical or physical) network of devices. In particular, we leverage the recently proposed *aggregate process* abstraction [12] to directly program dynamic processes (emitting, reading, or removing tuples) concurrently run by a networked set of devices in a scoped way through dynamic ensembles. Therefore, in the following we first review aggregate processes (Section IV-A) and then present an API in the ScaFi aggregate programming framework (Section IV-C).

A. Aggregate processes

Basically, *aggregate processes* [12] are dynamic aggregate computations. An *aggregate computation* [4] is a coordinated set of computation and communication acts performed by an *aggregate*, i.e., a collective of devices interacting on a neighbourhood basis. Such a computation is expressed by a *global perspective*, and takes the form of an *aggregate program* that is continuously re-interpreted by every device against its local context. The *field calculus* [24] is the core language for expressing aggregate computations, for which both local device and network operational models are defined. Denotationally, aggregate computations are defined in terms of whole manipulations of distributed data structures called *computational fields*, i.e., maps from a domain of devices to computational values. It is shown in [23] that space-time computation *universality* is achieved by the field calculus through only few mechanisms that provide: (i) stateful evolution of fields, to model dynamics, (ii) branching of fields, to model separation of domains for independent computations, and (iii) neighbour-based observation of fields, modelling interaction between nearby parts. For programs expressed in this fashion to actually produce the intended global results, operationally every device must adhere to a (program-independent) "behaviour protocol"; specifically, a device must "continuously" perform *computational rounds* where it:

1) gathers the local context (sensor data and messages from neighbours);
2) interprets the aggregate program against its local context;
3) uses the output of the aggregate program evaluation to act on its local context (through actuators) and inform neighbours of local changes.

Now, aggregate processes extend the field calculus with a construct, **spawn**, providing a way of expressing a dynamic number of concurrent aggregate computations with a dynamic scope [12]. To follow this section, it is sufficient for the reader to understand the general principles of aggregate computing and its execution model; full details are in [12], [24].

In order to *program* with aggregate processes, you generally follow the following steps.

```
// 1. Define the logic of a process in the aggregate approach
def process[K,A,R](key: K)(argument: A): (R,Boolean) = ???
// 2. Define a field of keysets for generating process instances
val keys: Set[K] = ???
// 3. Define a dynamic field of parameters
val args: A = ???
// 3. Call the spawn
val map: Map[K,R] = spawn[K,A,R](process _, keys, args)
```

Locally at each device, for each key `k:K`, an instance of `process` is generated. For each active process, its function is executed, returning a pair of type `(R,Boolean)` containing the process output and a status flag indicating whether the device participates in the process or not. All the participating devices propagate the process to their neighbours. In general, for a spatially limited process, a border of devices "external" to the process will form (*process boundary*); these devices will continuously re-evaluate the process to deal with its potential expansion. Conversely, the participating devices can call themselves out to deal with process shrinking through retraction of the border.

B. Spatiotemporal Tuples as Aggregate Processes

The idea behind an aggregate implementation of the Spatiotemporal Tuple model is to consider every Linda operation as an aggregate process, i.e., as a computation collectively executed by a dynamic set of situated devices. So:

- the problem of controlling the spatiotemporal region of a tuple becomes the problem of controlling the boundary of the corresponding process (which means defining a Boolean field for an evolving domain of devices);
- the problem of operation unblocking becomes the problem of controlling the lifecycle of executing and continuation processes;
- the problem of consistency becomes the problem of managing state and interaction between reading and writing aggregate processes.

The implementation straightforwardly reflects the abstract semantics described in Section III-C, where the corresponding events can be denoted by specific points of Boolean fields.

C. ScaFi Spatiotemporal Tuples API

ScaFi is an aggregate programming toolkit [25] for Scala, a strongly-typed multi-paradigm language running on the JVM. In ScaFi, the field calculus and the libraries on top are provided through *embedding* into Scala, i.e., as an *internal* domain-specific language (DSL) [26]. The ScaFi Spatiotemporal Tuples API is a library layer that exposes the usual Linda-like coordination primitives, extended to deal with situation of tuples, while hiding the management of aggregate processes through **spawn**. The code shown in this section is valid ScaFi/Scala code. A brief video of the tool is available online[1].

1) Communication language: Tuples and templates can be defined from strings.

```
val tuple: Tuple = "task(description)"
val tupleTemplate: TupleTemplate = "task(X)"
```

2) Spatiotemporal description language: Tuples and templates are, by default, situated at the executing device (`Me`). However, a situation can be explicitly indicated as follows,

```
tuple @@ spacetimeRegion1
tupleTemplate @@@ spacetimeRegion2
```

where, for instance:

```
val spacetimeRegion1 = Everywhere / Forever
val spacetimeRegion2 = AroundMe(ext = 30) // implicitly forever
```

These have straightforward mappings to field computations, leveraging *gradients* [27] to compute distances relatively to source localities. More complex (causally admissible) spatiotemporal regions can also be programmed.

3) Coordination and process description language: The coordination language consists of Linda primitives: **out**`(t)`, **in**`(tt)`, **rd**`(tt)`—shown in red. It is used together with a *process description language* (violet symbols) that helps to structure individual processes over an aggregate of devices.

```
process(taskGenerator){
  when(taskProposal){
    t => out(s"task($t)")
  }.andNext((tuple: Tuple) => { /* ... */ })
}
process(taskStealer){
  when(doReadTask) {
    in("task(X)" @@@ AroundMe(50))
  }.evolve((tuple: Tuple)  => {
    out(s"currentTask(${tuple})" @@ Me)
  }).evolve((tuple: Tuple) => {
    out(s"done(device(${mid}),${tuple})" @@ AroundMe(30))
  })
}
```

This snippet expresses a system specification from a global viewpoint, which is directly executable in a distributed fashion through the aggregate semantics. In `process(p)`, `p` is a Boolean field indicating the role(s) of each device, whereas construct `when` leverages Optional or Boolean fields for the activation of new operations (aggregate process instances).

[1]https://vimeo.com/394411022

V. Discussion

The proposed model of self-organising coordination reinterprets the Spatial Tuples model, where spatial information is just an attribute of tuples and an element for operations to match tuples on spatial basis, to actually consider tuples and tuple operations as spatially situated processes running on a collective adaptive system.

A first element of design concerns what properties and guarantees are to be provided by the model and whether (and to what extent) these have to or can be relaxed depending on the application scenario at hand. The basic requirements of the Linda model (i.e., atomicity of retrieval) have to be guaranteed. However, a distributed implementation might decide how to deal with the *CAP theorem* [22], i.e., what kind of consistency and availability guarantees to provide when facing failure and network partitions. In particular, relaxing consistency in favor of availability can support better scalability [28].

Another interesting aspect concerns the generality of the model w.r.t. deployments. The provided characterisation of a computational space-time structure captures both MANETs and cloud-based systems. In MANETs, each situated device has its own identity, mapping to a physical position, and each computation step of that device represents a space-time locality; i.e., the device carries out a (space-time) evolution. The model also straightforwardly maps to cloud-based deployments, whereby the cloud represents the only physical, non-situated device, and the space-time structure is rather given by a set of *logical* devices, e.g., assumed to be situated at some physical space-time location. Then, a physically situated user device can issue operations that will actually perform on the logical space overlay, in an "augmented reality fashion".

VI. Conclusion and Future Work

In this paper, we advance the idea of exploiting CAS approaches to support the implementation of advanced coordination models. To exemplify the idea, we propose a model for tuples in space-time whereby tuples and operations become collective adaptive processes on situated networks approximating a space-time structure. This enables straightforward implementation as concurrent aggregate processes and consequential trasfer of adaptivity features.

As future work, we would like to formally develop the model, proving the correctness of the mapping, as well as evolving the prototype into a full-fledged, tested API.

References

[1] D. Gelernter, "Generative communication in linda," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 7, no. 1, pp. 80–112, 1985.

[2] J.-L. Deneubourg, S. Aron, S. Goss, and J. M. Pasteels, "The self-organizing exploratory pattern of the argentine ant," *Journal of insect behavior*, vol. 3, no. 2, pp. 159–168, 1990.

[3] F. Baude, D. Caromel, L. Henrio, and M. Morel, "Collective interfaces for distributed components," in *7h IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*. IEEE, 2007, pp. 599–610.

[4] J. Beal, D. Pianini, and M. Viroli, "Aggregate programming for the Internet of Things," *IEEE Computer*, vol. 48, no. 9, pp. 22–30, 2015.

[5] Y. Abd Alrahman, R. De Nicola, and M. Loreti, "Programming interactions in collective adaptive systems by relying on attribute-based communication," *Science of Computer Programming*, p. 102428, 2020.

[6] A. Paulos, S. Dasgupta, J. Beal, Y. Mo, K. Hoang *et al.*, "A framework for self-adaptive dispersal of computing services," in *FAS*W@SASO/ICAC*. IEEE, 2019, pp. 98–103.

[7] R. Casadei and M. Viroli, "Coordinating computation at the edge: a decentralized, self-organizing, spatial approach," in *FMEC*. IEEE, 2019, pp. 60–67.

[8] R. Casadei, D. Pianini, M. Viroli, and A. Natali, "Self-organising coordination regions: A pattern for edge computing," in *COORDINATION*, ser. LNCS, vol. 11533. Springer, 2019, pp. 182–199.

[9] A. Bucchiarone and M. Mongiello, "Ten years of self-adaptive systems: From dynamic ensembles to collective adaptive systems," in *From Software Engineering to Formal Methods and Tools, and Back*. Springer, 2019, pp. 19–39.

[10] M. Viroli, M. Casadei, and A. Omicini, "A framework for modelling and implementing self-organising coordination," in *Proceedings of the 2009 ACM Symposium on Applied Computing (SAC)*, 2009, pp. 1353–1360. [Online]. Available: http://doi.acm.org/10.1145/1529282.1529585

[11] M. Viroli, J. Beal, F. Damiani, G. Audrito, R. Casadei, and D. Pianini, "From distributed coordination to field calculus and aggregate computing," *J. Log. Algebraic Methods Program.*, vol. 109, 2019.

[12] R. Casadei, M. Viroli, G. Audrito, D. Pianini, and F. Damiani, "Aggregate processes in field calculus," in *COORDINATION*, ser. Lecture Notes in Computer Science, vol. 11533. Springer, 2019, pp. 200–217.

[13] D. Gelernter, "Multiple tuple spaces in linda," in *International Conference on Parallel Architectures and Languages Europe*. Springer, 1989, pp. 20–27.

[14] M. Viroli, D. Pianini, and J. Beal, "Linda in space-time: An adaptive coordination model for mobile ad-hoc environments," in *COORDINATION*, ser. LNCS, vol. 7274. Springer, 2012, pp. 212–229.

[15] A. L. Murphy, G. P. Picco, and G.-C. Roman, "LIME: A coordination model and middleware supporting mobility of hosts and agents," *ACM Transactions on Software Engineering and Methodology*, vol. 15, no. 3, pp. 279–328, Jul. 2006.

[16] M. Mamei and F. Zambonelli, "Programming pervasive and mobile computing applications: The TOTA approach," *ACM Trans. Softw. Eng. Methodol.*, vol. 18, no. 4, pp. 15:1–15:56, 2009.

[17] J. Pauty, P. Couderc, M. Banâtre, and Y. Berbers, "Geo-linda: a geometry aware distributed tuple space," in *AINA*. IEEE Computer Society, 2007, pp. 370–377.

[18] A. Ricci, M. Viroli, A. Omicini, S. Mariani, A. Croatti, and D. Pianini, "Spatial tuples: Augmenting reality with tuples," *Expert Systems*, vol. 35, no. 5, p. e12273, 2018.

[19] G. Coulouris, J. Dollimore, and T. Kindberg, *Distributed systems - concepts and designs (3. ed.)*, ser. International computer science series. Addison-Wesley-Longman, 2002.

[20] R. Menezes and A. Wood, "The fading concept in tuple-space systems," in *Proceedings of the 2006 ACM symposium on Applied computing*, 2006, pp. 440–444.

[21] N. Busi, R. Gorrieri, and G. Zavattaro, "On the expressiveness of linda coordination primitives," *Inf. Comput.*, vol. 156, no. 1-2, pp. 90–121, 2000.

[22] E. Brewer, "CAP twelve years later: How the "rules" have changed," *Computer*, vol. 45, no. 2, pp. 23–29, 2012.

[23] G. Audrito, J. Beal, F. Damiani, and M. Viroli, "Space-time universality of field calculus," in *COORDINATION*, ser. Lecture Notes in Computer Science, vol. 10852. Springer, 2018, pp. 1–20.

[24] G. Audrito, M. Viroli, F. Damiani, D. Pianini, and J. Beal, "A higher-order calculus of computational fields," *ACM Trans. Comput. Logic*, vol. 20, no. 1, pp. 5:1–5:55, Jan. 2019. [Online]. Available: http://doi.acm.org/10.1145/3285956

[25] M. Viroli, R. Casadei, and D. Pianini, "Simulating large-scale aggregate mass with alchemist and scala," in *FedCSIS*, ser. Annals of Computer Science and Information Systems, vol. 8. IEEE, 2016, pp. 1495–1504.

[26] M. Voelter, *DSL Engineering - Designing, Implementing and Using Domain-Specific Languages*, 2013.

[27] G. Audrito, R. Casadei, F. Damiani, and M. Viroli, "Compositional blocks for optimal self-healing gradients," in *SASO*. IEEE Computer Society, 2017, pp. 91–100.

[28] E. G. Boix, C. Scholliers, W. De Meuter, and T. D'Hondt, "Programming mobile context-aware applications with TOTAM," *Journal of Systems and Software*, vol. 92, pp. 3–19, 2014.