Self-organising Coordination Regions: A Pattern for Edge Computing

Roberto Casadei, Danilo Pianini(✉), Mirko Viroli, and Antonio Natali

Alma Mater Studiorum–Università di Bologna, Cesena, Italy
{roby.casadei,danilo.pianini,mirko.viroli,antonio.natali}@unibo.it

Abstract. Design patterns are key in software engineering, for they capture the knowledge of recurrent problems and associated solutions in specific design contexts. Emerging distributed computing scenarios, such as the Internet of Things, Cyber-Physical Systems, and Edge Computing, define a novel and still largely unexplored application context, where identifying recurrent patterns can be extremely valuable to mainstream development of language mechanisms, algorithms, architectures and supporting platforms—keeping a balanced trade-off between generality, applicability, and guidance. In this work, we present a design pattern, named *Self-organising Coordination Regions* (SCR), which aims to support scalable monitoring and control in distributed systems. Specifically, it is a decentralised coordination pattern for partitioned orchestration of devices (typically on a spatial basis), which provides adaptivity, resilience, and distributed decision-making in large-scale situated systems. It works through a self-organising construction of regions of space, where internal coordination activities are regulated via feedback/control flows among leaders and worker nodes. We present the pattern, provide a template implementation in the Aggregate Computing framework, and evaluate it through simulation of a case study in Edge Computing.

Keywords: Coordination · Distributed systems · Design patterns · Decentralised orchestration · Self-organisation · Edge computing

1 Introduction

Design Patterns are paramount in software engineering. They capture expert knowledge by describing reasoned solution schemas for a well-defined class of repeatedly occurring problems in specific contexts [8]. Patterns help harnessing complexity by characterising systems of forces arising in a context, and strategies to resolve them [1], while abstracting from implementation details, denoting intents and properties of solutions, providing motivated guidance towards desired configurations, and supporting documentation and team communication through a common vocabulary [8]. Over time, several classes of patterns have been discovered to assist designers and implementors of software-based systems, resulting in *catalogues* of patterns, e.g., for object-oriented software [21], concurrency [40],

Published by Springer Nature Switzerland AG 2019
H. Riis Nielson and E. Tuosto (Eds.): COORDINATION 2019, LNCS 11533, pp. 182–199, 2019.
https://doi.org/10.1007/978-3-030-22397-7_11

messaging [25], reactive systems [44], fault-tolerant software [23] etc. Moreover, patterns can be classified into multiple taxonomies (e.g., by level of abstraction into architectural, design patterns, and idioms [8]), can be related to each other (e.g., by refinement, variance, and combination [8]), and can be presented using different formats (e.g., *Alexandrian* [1], *GoF* [21], and *POSA* [8]).

In this paper, we consider the context of coordination in large-scale distributed systems. Specifically, we focus on scenarios – e.g., pervasive computing, Collective Adaptive Systems (CAS), Internet of Things (IoT), Cyber-Physical Systems (CPS), and Edge Computing – characterised by the following forces:

- *Distribution.* Having distributed components leads to concurrency, lack of global clock, and independent (and often frequent) failure or unavailability of components [14]—with corresponding implications.
- *Situatedness.* Components may be logically or physically immersed into an environment such that their location and context are relevant, since their inputs and outputs may be limited to the surroundings.
- *Heterogeneity.* Components may differ by their computational capabilities, energy requirements, and general dependability.
- *Large scale.* Systems may be too large to be centrally orchestrated or manually operated.

Given the rather intense research ongoing in these contexts, their broad scope, complexity of the challenges, and proliferation of paradigms, some catalogues of design patterns have emerged. Relevant examples include pattern catalogues for multi-agent architectures [24] and ensemble structures [26], bio-inspired computing [20], and decentralised control [49] and coordination [17] in self-adaptive systems. They typically work at different levels of abstractions, from principles and high-level behaviour components to mathematically-defined evolution rules, and do not generally provide complete solutions for the complex problem of scalable coordination of large-scale situated systems.

Accordingly, in this paper we provide three original contributions, namely, we: (*i*) present a general, decentralised coordination design pattern for partitioned orchestration that aims to provide adaptivity and resilience in large-scale situated systems; (*ii*) improve over the existing instances by proposing a feedback loop dynamically resizing partitions, to be used e.g. for load balancing; (*iii*) propose a possible implementation of the pattern in the Aggregate Computing framework [4]; and (*iv*) show an application of the pattern in the context of *edge computing*, through a case study.

The pattern we describe finds application in several scenarios where a sparse set of leaders is expected to collect feedback from and enact decisions for a subset of other participants—examples include distributed sensing [12], target counting [37], group management for target tracking [32], decentralised service orchestration [29], self-adaptative software [49], Wireless Sensor Networks (WSN) [19,31], robot swarm control [48], crowd tracking and steering [4,10], peer-to-peer clouds [13], and coordination in hierarchical thing/edge/fog/cloud environments (as explored in this paper). We call this pattern ***Self-organising Coordination Regions (SCR)***, since it works through an internally-regulated,

adaptive construction of regions where activity is coordinated via feedback/control flows among master and worker nodes. In other words, it leverages asymmetry in complex coordination scenarios and accordingly proposes a tunable trade-off between centralised and decentralised decision-making.

The rest of this paper is structured as follows, with content following roughly the GoF pattern template form [21]. Section 2 provides context, a motivating example and discusses related work and patterns. Section 3 presents the pattern by providing its intent, synonyms, structure, dynamics as well as known uses, consequences and methodological guidelines of its application. Section 4 shows an implementation in the Aggregate Computing framework, and discusses variants. Section 5 provides empirical evaluation. Finally, Sect. 6 concludes the paper.

2 Motivation

2.1 Motivating Scenario

Background: Edge Computing. Fog and edge computing [7,41,43] are emerging paradigms with the goal of bringing cloud-like functionality at the edge of the network, i.e., close to end users and to where data is generated and used (or, generally, to where computational intelligence is most needed—cf., IoT and CPS). There are at least three cases in which this is highly desirable: (i) *when the cloud is not available*, e.g., because of lack of Internet connectivity; (ii) *when the cloud is available but it cannot satisfy application requirements*, because of data privacy issues or lack of real-time guarantees due to large round-trip time to remote data centres; (iii) *when the cloud is available and suitable but it is costly*, e.g., in terms of subscription or network bandwidth. That is, edge computing is in some cases a necessity, but in general it represents a complementary model to cloud computing which enables a whole new set of possibilities ranging from infrastructure-level optimisations (like exploiting idle edge devices or filtering data before sending it to the cloud) to flexibility in service-level agreements and resilience through decentralisation.

Case Study. As paradigmatic case study, consider a multimedia application that requires computation over user-generated video stream and low-latency communication. Example applications are, e.g., metropolitan collaborative surveillance [16] and multiplayer gaming. For the latter, pervasive usage of multi-view and 360-degree-view video streams is currently limited by delay intolerance and excessive bandwidth usage [5]. Moreover, relevance of low-latency video processing will likely increase in the future with advancements in mobile augmented reality technology [39]. One wants such multimedia application to execute on a smart urban environment, where users, equipped with mobile devices (smartphones, or even augmented-reality equipment) can move. The smart city is populated with a network of static (non-mobile) edge servers, with which mobile devices can communicate. The goal is to adaptively select a subset of edge nodes (enough to sustain the computation) to work as local leaders, gather and redirect the video streams from user devices to one leader edge device, process the data gathered, and finally spread the computation result back to the users.

2.2 Problem and Forces

The SCR pattern addresses the problem of coordination in situations where:

1. heterogeneity creates asymmetry in individual capabilities, or tasks are so complex that *collaboration* is essential;
2. a *locality principle* holds, as context is key and cost is typically proportional to the distance between sources, processes, and users;
3. neither *full centralisation* nor *full decentralisation* in control and decision making is possible or desirable; and
4. the environment and system structure are *dynamic* (e.g., due to mobility or failure).

2.3 Related Work and Patterns

The SCR pattern recurs in a number of scientific works and proposed solutions, and is implemented variously.

Related Patterns and Abstractions. Related catalogues of design pattern and abstractions include [17], addressing decentralised coordination in self-organising emergent systems; [20], covering bio-inspired patterns; [49], focussing on decentralised control in self-adaptive systems; and [45], providing a library of reusable components of distributed behaviour. Some patterns there presented constitute the foundations of the current work. Indeed, the SCR pattern is a combination of three fundamental coordination (sub-)patterns:

- *Multi-leader election.* In distributed systems, it is sometimes useful to break symmetry or introduce multiple local centralisation points to simplify decision making or coordination. This pattern consists in the election of multiple leaders to uniformly cover a logical or physical space.
- *Information propagation.* Communication patterns that abstract from low-level implementation or networking details are essential in distributed systems. This pattern consists of propagating information from one or more sources outward, independently of the underlying system structure.
- *Information collection.* This pattern consists of collecting information from a set of sources into one or more sinks, still abstracting from low-level details.

In order to account for situations where devices can fail or change, coherently to the self-organisation principle, we should consider the above patterns as *continuous processes* (or, at least, as processes that are *reactive* [34] to failure or change). This means that information (updates) must move continuously, as a stream (logically, and despite potential optimisations), as captured by the *information flow* abstraction, defined in [18] as follows:

> *An information flow is a stream of information from source localities towards destination localities and this stream is maintained and regularly updated to reflect changes in the system. Between sources and destinations, a flow can pass other localities where new information can be aggregated and combined into the information flow.*

A common way to implement information flows is by activating processes that create and maintain structures for the communication paths. One such example is the *gradient* [2,15,17,33], a self-healing distributed data structure mapping any node of the system to its hop-by-hop estimated distance from source points: it provides an underlying carrier for controlling effective directions of propagation/collection of data flows. Information flows can be naturally expressed in the library of [45], which fosters the definition of collective behaviour of an ensemble of devices through a composition of self-organising patterns, drawing inspiration from biology [20]. The aforementioned sub-patterns are "building blocks" in [4], where are respectively called S (for ***S****parse-choice*—i.e., a scattered selection from the set of participating devices), G (for ***G****radient-cast*—i.e., a multicast diffusing information along a gradient), and C (for ***C****onverge-cast*—i.e., a multicast aggregating information to a sink device).

A well-known organisational meta-pattern for self-adaptive systems is MAPE [30]: it suggests structuring the system feedback control loop into four components: Monitor, Analyse, Plan, and Execute. In [49], several MAPE patterns are provided for organising the adaptation logic in decentralised self-adaptive systems. These are related and operate in a similar design context, but their focus is on internal organisation of system adaptivity rather than on external, application design. In particular, the *Regional Planning* pattern [49] consists in distributing *Planning* components to different "software regions" (i.e., loosely coupled software subsystems); there, they collect data from *Analyse* components (which are fed by *Monitoring* components) and command *Execute* components for enaction of planned adaptations. SCR subsumes Regional Planning: it enables the design of self-adaptation control loops but goes beyond that, by covering various assignments of responsibilities to the participants and being directly usable for application logic as well; e.g., leaders in SCR may gather regional data, resolve contention, or propagate events.

Known Uses. Various forms and uses of the SCR pattern can be found in literature. In [29], SCR is used to design a decentralised service orchestration system; there, a workflow specification is split for scalability and performance into sub-workflows executed by multiple collaborating engines that are migrated to different network regions based on placement analysis. In [19], SCR is applied in the design of a WSN middleware, *TCMote*, where the system is organised in (possibly hierarchical) *sensor regions* governed by *leaders* with higher capabilities than the other region nodes (called *motes*); TCMote uses tuple channels for one-to-many and many-to-one communication between region sensors and the region leader in a single-hop. In the WSN middleware *TS-Mid* [31], tuple space-based logical regions are used for power saving; there, regional leaders dispatch operations to normal nodes and transmit results to sink nodes. In [48], the authors leverage dynamically selected, human-controlled leaders to guide robot swarms towards goal regions. Other known uses of the pattern include distributed sensing [12], target counting [37], group management for target tracking [32], design of self-adaptation control loops [49] (as discussed above), crowd tracking and steering [4,10] in opportunistic IoT, as well as peer-to-peer clouds [13].

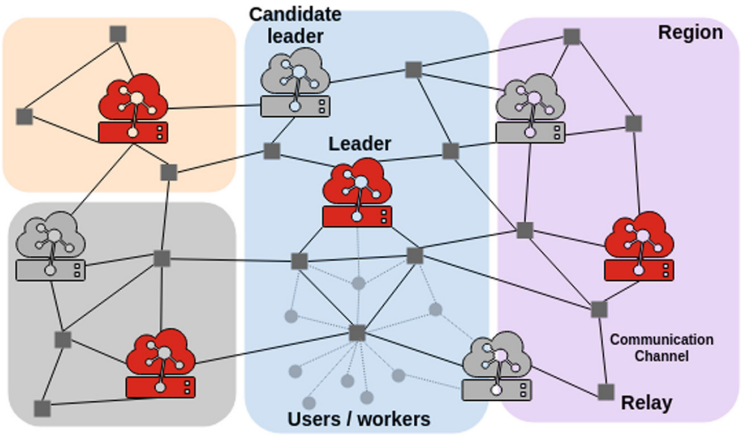

Fig. 1. SCR from a structural perspective—see description in Sect. 3.1. Notation: "gateway-like" nodes denote candidate leaders (red for active ones, grey for unelected ones); small grey squares denote relays; small grey circles denote users/workers (Color figure online)

3 Pattern Description

Intent. Support scalable control and monitoring of a distributed system, with resiliency to failures and dynamicity, and balancing centralisation and decentralisation in decision making.

Name and Synonyms

- *Self-organising Coordination Regions.* This reflects the decentralised nature of this pattern, as well as its support for coordination through scoped, endogenous, emergent structures and dynamics.
- *Decentralised Multi-Orchestration.* This is also a suitable name, as the pattern defines a decentralised coordination strategy for injecting multiple orchestration points into a system, creating corresponding system partitions regulated through feedback loops.
- *SGCG.* This name denotes the chain of aggregate programming blocks that provides a possible implementation schema of the pattern (see Sect. 4).

3.1 Structure and Participants

Structurally, the pattern is organised as of Fig. 1. The system is a network of *nodes* on which spatially extended and dynamic structures, called *regions*, emerge, each "containing" a subset of devices. These components can assume at any time one or more of the following roles:

- *Candidate leader*: a device eligible for leader election—even though the pattern itself makes no assumption on the network structure, on an edge deployment usually candidate leaders correspond to edge servers;

– *Leader*: the device responsible for obtaining information from and propagating decisions within a *region*;
– *User* (or *worker*): device which sends/receives information to/from the leader of the region it is part of;
– *Relay*: non-user and non-candidate device participating in the computation.

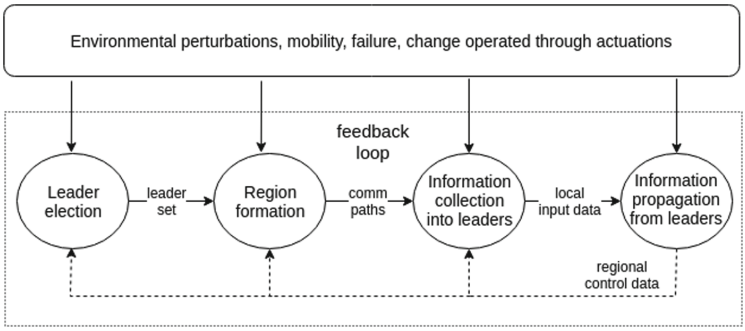

Fig. 2. SCR from a dynamical perspective—see description in Sect. 3.2. Notation: solid arrows represent required inputs or unavoidable perturbations; dashed lines denote possible feedback loops

3.2 Dynamics and Collaborations

The pattern induces a computational behaviour organised in four phases:

1. *Election of leaders*. Leaders are elected from the set of candidates.
2. *Formation of regions*. Structures are created such that each user is assigned to a single leader, and information can flow in both directions through proper communication paths.
3. *Information flow from users to leaders*. User nodes stream data or updates needed by leaders to achieve the system goals, and some processing can occur *en-route*—examples include sensor data, local events, service requests, or feedback information for the assigned tasks.
4. *Information flow from leaders to users*. Leaders stream computation results to all members of their managed region—it may be a decision to be enacted, a collective view to be propagated, instructions to be assigned, and so on.

Note that these phases are only conceptually sequential: they are rather dynamical processes that happen concurrently, are continuously revised, and are related by input/output dependencies (see Fig. 2). Specifically, the leader election phase can be thought as an active process black box that can react to various perturbations to automatically revise the selection of leaders and shape of regions; then, as regions change, the corresponding collection and propagation processes need to adapt. Moreover, the system can be configured with feedback

loops: information propagated by leaders may produce an effect on workers that can subsequently get perceived by leaders through collected data.

Variants and Extensions

- *Leader election with pre-established regions.* In some cases, the regions must be decided before the corresponding leaders are elected.
- *Connected leaders.* In some scenario, communication between leaders is desired to allow for global, system-wide coordination that goes beyond the needs of individual regions.
- *Hierarchical organisation.* The pattern can be applied recursively: a region can be split into sub-regions governed by sub-leaders, and so on.
- *Overlapping regions.* Multiple instances of the pattern may be concurrently spawned with different regions, in order to provide in each device a superimposed view of its various "localities". This requires the capability to execute some parts of the distributed coordination algorithm concurrently.

3.3 Applicability

When to Apply. Use of the SCR pattern is encouraged in any of the following:

- A large-scale situated system needs to self-organise in such a way that its components can be monitored and coordinated according to a view larger than local, such as in complex situation recognition.
- A balance between centralisation and decentralisation is required to support effective decision making in large-scale, dynamic contexts.
- All or part of the information should be processed nearby the users, because of resource constraints like bandwidth, storage, energy, and so on.
- The underlying network structure is unknown, the system is open (new relays, leader candidates and users can join and leave the system dynamically), failures are possible, or other events can dynamically change the network structure.

When Not to Apply. Adoption of the SCR pattern is discouraged (or would lead to degenerate cases) in the following circumstances:

- Decision making can be carried out in a fully local way.
- Decision making must be entirely centralised (actually, this could be tackled by electing a single leader, but more efficient solutions may exist for less dynamic scenarios).
- The network structure is statically defined.

3.4 Consequences

The SCR pattern has the following consequences:

- *Hybrid decision making.* Decisions are taken considering a tunable subset of the whole system, de-facto creating a hybrid between centralised and decentralised decision making.
- *Sub-network isolation.* Unless an extended version of the pattern is deployed, users belonging to different regions do not participate in the same sub-system (i.e., they do not exchange data).
- *Reduced dependence from deployment and network structure.* SCR creates a sort of dynamic, adaptive network overlay structure on top of the existing communication infrastructure. By merely organising application logic on that overlay, the specific shape of the underlying network can be abstracted away, allowing for easier porting to diverse setups (e.g. cloud, edge, purely P2P).
- *Eventual consistency.* Temporal mobility, loss of messages, and device failures, only temporarily affect the values collected in leaders, and hence, deviation from the actual global view.

4 Implementation

In this section, we describe some possible variants in the implementation strategy of the four phases described in previous section (Sect. 4.1), and then provide an example specification in the framework of Aggregate Computing (Sect. 4.2).

4.1 Implementation Issues

Election of Leaders and Formation of Regions

- *Consensus strategy.* Consensus on leadership may involve centralised algorithms, or resort to (more challenging) algorithms for distributed and asynchronous systems [42].
- *Candidate leaders.* In general, there could be constraints or preferences concerning which nodes can be selected as leaders: coordinators are usually preferably static, dependable nodes with significant computational and network resources, and little or no power saving concern—such as edge gateways or fog nodes. Trust could also be used to rate and therefore include/exclude nodes from the candidate set based on observed activity [9].
- *Time of election.* Leaders can be elected statically (i.e., before system execution) or be dynamically reconsidered, continuously or after a delay.

- *Objectives.* The goal is usually a configuration of leaders that must be valid or optimised with respect to a particular property—e.g., uniformity in spatial coverage (as of a smart city environment) or balancing of load (tasks, workers).
- *Adaptivity and resilience.* A new leader election process must be activated when the current leader configuration gets invalidated. E.g., this could happen due to mobility, change of load, or failure of some leader.

Information Spreading

- *Gossip.* One way to implement spreading of information is through gossip protocols [6], which are suitable for letting information flow from leaders to users under the condition that the generated information is monotonic (namely, it can only change in a single direction). Whenever such an assumption does not hold, gossip algorithms should get periodically reset (or overlapping replicates of the algorithm should execute in parallel [36]).
- *Gradient-based information cast.* A class of algorithms for distributed information spreading is rooted on the idea of carrying information along with a monotonically-increasing (logical or physical) distance from the information source. This is suitable both for generating regions once leaders are elected (by selecting the closest leader) and for propagating information from leaders to users. Several implementations of the algorithm exist, ranging from distributed adaptive Bellman-Ford [15] to advanced versions and compound algorithms taking into account aspects like time, speed, and acceleration of devices [2].

Information Accumulation

- *Gossip.* Information accumulation is generally a tougher task than information spreading. As for spreading, accumulation can be realised by gossiping information such that the leader is reached with messages from all nodes in the region: however, this effectively works only in the case of small regions.
- *Spanning tree techniques.* A more scalable technique is based on building a spanning tree over the network (locally selecting as parent the closest neighbour to the source), then accumulating along such tree towards the leader. Spanning trees, however, are highly fragile to changes in the network: disruption and creation of links may lead to different configurations, making naive versions of this algorithm unsuitable for mobile scenarios.
- *Multi-path techniques.* Multi-path techniques aggregate information along the source using multiple spanning trees rather than a single one. They are usually more robust to changes in the network structure, but take more time to converge in case of stable networks [45].

4.2 Sample Code

We propose an implementation draft for the pattern in the paradigm of aggregate computing [4,46]—used in next section as a basis for evaluating a smart city case study. The reason for this choice is rooted in the rather straightforward mapping between the sub-patterns of SCR and the building blocks available in existing aggregate computing languages, which allow for a concise implementation.

Background: Computational Fields and Aggregate Computing [4,46]. Aggregate computing is founded on the idea of programming systems from a global perspective, declaratively [47], by functional manipulation of *(computational) fields* data structures—time-evolving maps from devices to values. The *field calculus* [3,46] is the formal, universal, minimal language for functionally composing and manipulating fields, based on which domain-specific languages (DSL) like ScaFi [12] and Protelis [38] have been introduced to specify, simulate and run self-organising behaviours and collective coordination logic.

In the field calculus, a program describes a collective behaviour by neglecting the single-device viewpoint. However, the operational semantics [3] defines how the single device can "continuously" process the program and sustain the overall system behaviour, by cyclic steps encompassing: *(i)* assessment of a local context (previous state, environment perception, collection of input messages received so far); *(ii)* interpretation of the aggregate program against such a context (producing a new state, messages to be sent, and actions to be executed); *(iii)* execution of actions and spread of messages to neighbours.

Pattern Implementation Schema. In ScaFi, a Scala-internal DSL for aggregate programming, the pattern can be encoded as follows[1] (for the implementation of the sub-patterns in ScaFi and details on the syntax, refer to [11]) (Fig. 3):

```
class SCR extends AggregateProgram with BlockG with BlockC with BlockS {
  def main = {
    // selects a field of leaders, with at least grain distance
    val leader = branch(isCandidate) { S(grain) } { false }
    // creates a gradient from leaders based on a given metric
    val potential = distanceTo(leader, metric)
    // gathers localInput values towards leaders by aggregation
    val convergeCast = C(potential, localInput, aggregationFun)
    // on leaders, takes a local decision based on received data
    val decision = decisionMaking(leader, convergeCast)
    // broadcast decisions and take action
    val divergeCast = G(leader, metric, decision)
    localAction(divergeCast)
  }
}
```

[1] Purple symbols are non-primitive aggregate building blocks, grey symbols are configuration parameters, and bold symbols denote methods for local activity to be tailored to the application.

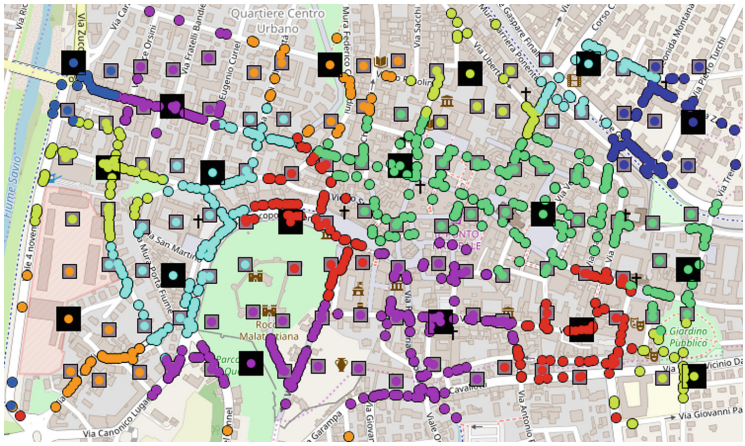

Fig. 3. A snapshot of the simulation in execution. Edge servers are depicted as square nodes, users as circular nodes. Leaders are black, big squares; unelected leaders (working as relays) are smaller, greyed squares. The colour of the circular dot identifies the id of the region assigned to that node (Color figure online)

5 Evaluation

In this section, we present an example implementation of the pattern in the context of smart cities and edge computing (as introduced in Sect. 2.1) and evaluate it by simulation to reveal its intrinsic self-organisation character.

5.1 Scenario Description

We consider a scenario of multiple edge servers (specifically, 126) in the centre of the Italian city of Cesena, all participating in the system as leader candidates. Their positions form an irregular grid, and vary on different simulation runs. We dynamically select a subset of these leader candidates to work as leaders, and let the others participate in the system as relays. More precisely, the edge servers elect a leader for every region of 200 m in radius, competing using the `S` building block (namely, breaking symmetry using a device local id, and favouring already established leaders if in a proper range).

The goal of the system is to collect data streams generated by users, aggregate it, and diffuse to the whole region the number of streams being processed. Users are modelled as devices moving along roads open to pedestrian traffic (data obtained from OpenStreetMap [22]) at a constant speed of $1.4\frac{m}{s}$. Bidirectional communication is considered established between users and edge servers, and among edge servers, if physical distance is within WiFi range (100 m). Users do not directly communicate with each other. In our experiment, we let the system run for 10 simulated minutes, then we simulate a disruptive event: elected leaders fail with probability ρ—e.g. as would happen due to a city-wise power shortage. After this event, we simulate 10 further minutes of system evolution.

Table 1. Free variables for the scenario in exam

Name	Description	Values
u	Active user devices count	[50, 100, 200, 500, 1000]
α	Backoff algorithm parameter	[0, 10^{-3}, 10^{-2}, 10^{-1}, 1]
ρ	Probability for a leader to shut down after 10 min	[0, 0.25, 0.5, 0.75, 1]
fb	Determines whether the feedback loop is enabled	[true, false]

Table 2. Measures for the case study

Name	Description	Unit
E of feedback adjustment	Mean of the feedback adjustment for every leader. It measures how much the radius of the coordinated region is extended. Lower values indicate bigger regions	m
σ of feedback adjustment	Standard deviation of the feedback adjustment for every leader. It is an indication of how much the radius of the coordinated region varies among leaders. Higher values indicate higher disparity in such values, meaning that the feedback system is altering the region sizes more intensively	m
$\sum$ of clients per edge server	Overall number of users served. The value should ideally match the number of users in the system. Higher values indicate streams being processed by multiple leaders (due to users changing region), lower values indicate non-served users	users
σ of clients per edge server	Standard deviation of number of users served by each leader. Indication of load balancing. Higher values indicate that more computational capacity is required for some leaders w.r.t. others. The lower, the better balanced is the load	users

We compare two implementations of the SCR pattern, a classic one (as described in Sect. 4) and a version with a feedback loop. In the latter, leaders try to coordinate and resize their regions in the attempt to cover approximately the same number of users, so as to reduce disparities in elaboration load that would cause slowdowns on overloaded edge servers. We implement self-organising adaptation of region size by feeding the information on the number of served users back to the leader, and using it to dynamically change the region size (the more users, the smaller the region), competing with other leaders. In order to prevent sharp oscillations of the region sizes, with possible resonance phenomena, we don't feed the served user count back to the algorithm input directly, but we filter it using an exponential backoff (a low pass filter), namely, the feedback value is $\alpha u_t + (1 - \alpha)u_{t-1}$, where u_t is the count of served users at time t.

We first evaluate good values for α in our scenario, by looking at how different values affect the size of regions and their stability. We then measure performance and resilience for both the base and the optimal-α versions of the SCR pattern varying the number of users and ρ, and observe the number of users served in total and by each edge server. A summary of the free variables for the case study is given in Table 1; measures are instead summarised and explained in Table 2.

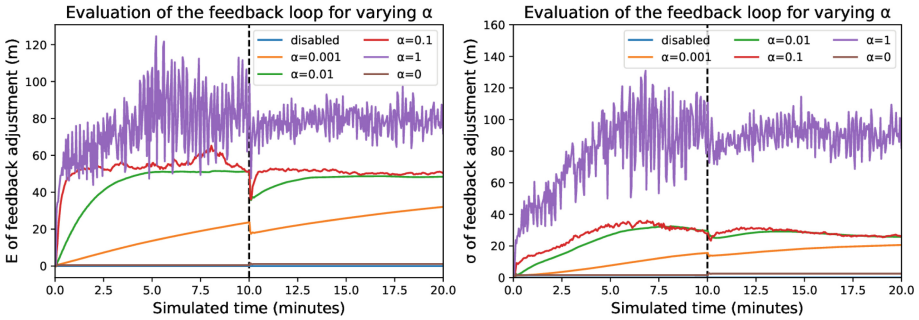

Fig. 4. Evaluation of the backoff parameter. Values are averaged along all values of u and ρ. Not considering new values ($\alpha = 0$) has a similar effect to disabling feedback entirely. Plugging the feedback directly, without filtering, makes the system oscillate. Other values show how α tunes the trade-off between reactivity and stability, with $\alpha = 0.01$ both smooth and with an impact on the system comparable to $\alpha = 0.1$

The pattern has been implemented in Protelis [38], and simulations have been performed using Alchemist [35]. We executed 100 replicas of the experiment for each configuration in the cartesian product of the parameters values, varying displacement of edge devices, initial position of users and their waypoints, and execution times of devices. Data has been processed using Python xarray [27] and matplotlib [28]. The experiments include a reference implementation of the SCR pattern, they are entirely open-sourced, automated, and reproducible using the instructions provided in a publicly accessible repository[2]. Confidence intervals are not pictured in charts reported on this paper, but can be obtained by using the full data and processing tools available in the aforementioned repository.

5.2 Results

We initially measure the benefits of using the feedback system and the impact of different values for α. Results are depicted and described in Fig. 4, and show how $\alpha = 10^{-2}$ is the best choice among the analysed values.

We then evaluate correctness and performance of the algorithm both without and with feedback enabled ($\alpha = 10^{-2}$). Results presented in Fig. 5 show that the system is able to serve all the users, actually serving some users twice at the moment they cross the boundary between neighbouring regions.

Finally, we study resilience of the system to failures by analysing its behaviour with different sudden disruptions hitting the leaders. Figure 6 shows the pattern reaches stability in few seconds even when disruption is large, and regardless of the feedback system. At disruption time, several nodes are not served and several others get instead apparently overserved, as they are in an inconsistent state and participating in multiple, quickly changing regions, with their streams

[2] https://bitbucket.org/danysk/experiment-2019-coordination-dynamic-orchestration.

getting lost because of the time required to recover both regions and spanning trees for data accumulation. The feedback system has a negligible impact on resilience, but improves load balancing both before and after disruption.

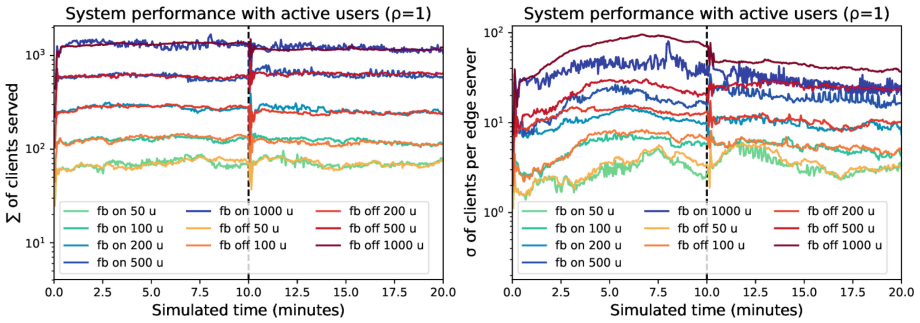

Fig. 5. System correctness. Warm colours are results with feedback system disabled, cold colours are results with feedback system enabled and $\alpha = 10^{-2}$. Both configurations serve all the users, and actually slightly "overserve" them. This is due to the fact that users joining a different region, have, for some time, their streams counted also in the region they left due to network propagation and elaboration times. The feedback system provides benefits in terms of load balancing, as depicted in the right chart: the lower σ means lower disparity among leaders in the number of served users (Color figure online)

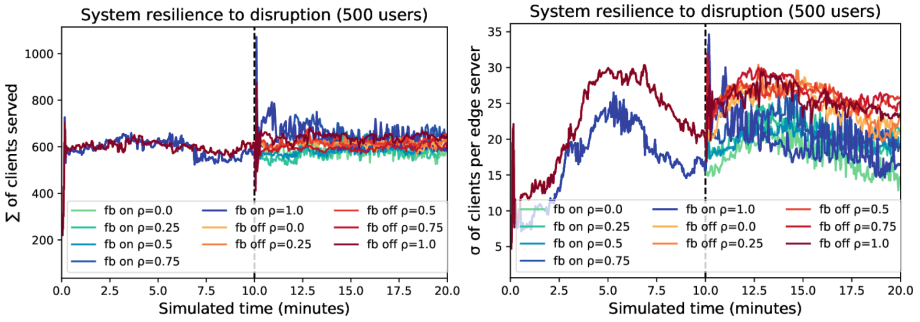

Fig. 6. System resilience to disruption. Both the pattern configurations provide resilience to disruptions. The system is able to find new leaders in few seconds even if the whole set of previously selected leaders is shut off. The feedbacked system seems to achieve slightly better performance for smaller disruptions, but takes more time to stabilise in the worst case. As seen in Fig. 5, the feedbacked system achieves visible better performance in terms of load balancing, both before and after the disruptive event, regardless of its entity

6 Conclusion

In this paper, we introduce *Self-organising Coordination Regions*, an adaptive coordination pattern for dynamic, opportunistic scenarios where neither complete centralisation nor full decentralisation of control and decision making are possible or desirable. The pattern fits a problem of potentially growing relevance, and it is particularly suitable for edge systems and for deploying a coordination stance that covers more than pure locality yet without requiring any global coordinator. To show applicability and benefits, we also present a case study in edge computing, showing that the pattern is able to create semi independent coordination regions, aggregate information, and propagate results to region members. The pattern is also easily extensible: we show, e.g., how a simple feedback mechanism could be devised to improve the load balancing across different leaders. We believe the presented pattern, along with easy implementation on the Aggregate Computing framework and its library of reusable blocks, can streamline prototype and development of a wide class of advanced coordination mechanisms, especially in the context of edge computing.

References

1. Alexander, C.: A Pattern Language: Towns, Buildings, Construction. OUP, Oxford (1977)
2. Audrito, G., Casadei, R., Damiani, F., Viroli, M.: Compositional blocks for optimal self-healing gradients. In: IEEE SASO (2017)
3. Audrito, G., Viroli, M., Damiani, F., Pianini, D., Beal, J.: A higher-order calculus of computational fields. ACM Trans. Comput. Log. **20**(1), 5:1–5:55 (2019)
4. Beal, J., Pianini, D., Viroli, M.: Aggregate programming for the internet of things. IEEE Comput. **48**(9), 22–30 (2015)
5. Bilal, K., Erbad, A.: Edge computing for interactive media and video streaming. In: 2nd International Conference on Fog and Mobile Edge Computing (FMEC). IEEE, May 2017
6. Birman, K.: The promise, and limitations, of gossip protocols. ACM SIGOPS Oper. Syst. Rev. **41**(5), 8 (2007)
7. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: 1st Workshop on MCC, pp. 13–16. ACM (2012)
8. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture, Volume 1: A System of Patterns. Wiley, Hoboken (1996)
9. Casadei, R., Aldini, A., Viroli, M.: Towards attack-resistant aggregate computing using trust mechanisms. Sci. Comput. Program. **167**, 114–137 (2018)
10. Casadei, R., Fortino, G., Pianini, D., Russo, W., Savaglio, C., Viroli, M.: Modelling and simulation of opportunistic IoT services with aggregate computing. Futur. Gener. Comput. Syst. **91**, 252–262 (2019)
11. Casadei, R., Pianini, D., Viroli, M.: Simulating large-scale aggregate MASs with Alchemist and Scala. In: FedCSIS Proceedings, pp. 1495–1504. IEEE (2016)
12. Casadei, R., Viroli, M.: Programming actor-based collective adaptive systems. In: Ricci, A., Haller, P. (eds.) Programming with Actors. LNCS, vol. 10789, pp. 94–122. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00302-9_4

13. Casadei, R., Viroli, M.: Coordinating computation at the edge: a decentralized, self-organizing, spatial approach. In: Proceedings of 4th IEEE Fog and Mobile Edge Computing Conference (2019, to appear)
14. Coulouris, G.F., Dollimore, J., Kindberg, T.: Distributed Systems: Concepts and Design. Pearson Education, London (2005)
15. Dasgupta, S., Beal, J.: A Lyapunov analysis for the robust stability of an adaptive bellman-ford algorithm. In: 55th Conference on Decision & Control (CDC). IEEE (2016)
16. Dautov, R., Distefano, S., Bruneo, D., Longo, F., Merlino, G., et al.: Metropolitan intelligent surveillance systems for urban areas by harnessing IoT and edge computing paradigms. Softw.: Pract. Exp. **48**(8), 1475–1492 (2018)
17. De Wolf, T., Holvoet, T.: Design patterns for decentralised coordination in self-organising emergent systems. In: Brueckner, S.A., Hassas, S., Jelasity, M., Yamins, D. (eds.) ESOA 2006. LNCS (LNAI), vol. 4335, pp. 28–49. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-69868-5_3
18. De Wolf, T., Holvoet, T.: Designing self-organising emergent systems based on information flows and feedback-loops. In: 1st SASO Conference, pp. 295–298. IEEE (2007)
19. Diaz, M., Rubio, B., Troya, J.M.: A coordination middleware for wireless sensor networks. In: Systems Communications, pp. 377–382. IEEE (2005)
20. Fernandez-Marquez, J.L., Serugendo, G.D.M., Montagna, S., Viroli, M., Arcos, J.L.: Description and composition of bio-inspired design patterns: a complete overview. Nat. Comput. **12**(1), 43–67 (2013)
21. Gamma, E., Helm, R., Johnson, R., Vlissides, J.M.: Design Patterns: Elements of Reusable Object-Oriented Software, 1st edn. Addison-Wesley Professional, Boston (1994)
22. Haklay, M., Weber, P.: OpenStreetMap: user-generated street maps. IEEE Pervasive Comput. **7**(4), 12–18 (2008)
23. Hanmer, R.S.: Patterns for Fault Tolerant Software. Wiley, Hoboken (2013)
24. Hayden, S., Carrick, C., Yang, Q., et al.: Architectural design patterns for multiagent coordination. In: International Conference on Agent Systems, vol. 99 (1999)
25. Hohpe, G., Woolf, B.: Enterprise Integration Patterns. Prentice Hall, Upper Saddle River (2004)
26. Horling, B., Lesser, V.: A survey of multi-agent organizational paradigms. Knowl. Eng. Rev. **19**(4), 281–316 (2004)
27. Hoyer, S., Hamman, J.: xarray: N-D labeled arrays and datasets in Python. J. Open Res. Softw. **5**(1), 1–6 (2017)
28. Hunter, J.D.: Matplotlib: a 2D graphics environment. J. Open Res. Softw. **9**(3), 90–95 (2007). https://doi.org/10.1109/MCSE.2007.55
29. Jaradat, W., Dearle, A., Barker, A.: Towards an autonomous decentralized orchestration system. Concurr. Computat. Pract. Exper. **28**(11), 3164–3179 (2016)
30. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **1**, 41–50 (2003)
31. Lima, R., Rosa, N., Marques, I.: Ts-Mid: middleware for wireless sensor networks based on tuple space. In: 22nd AINA Workshops, pp. 886–891. IEEE (2008)
32. Liu, J., Liu, J., Reich, J., Cheung, P., Zhao, F.: Distributed group management in sensor networks: algorithms and applications to localization and tracking. Telecommun. Syst. **26**(2–4), 235–251 (2004)
33. Lluch-Lafuente, A., Loreti, M., Montanari, U.: Asynchronous distributed execution of fixpoint-based computational fields. Log. Methods Comput. Sci. **13**(1) (2017)

34. Magnaudet, M., Chatty, S.: What should adaptivity mean to interactive software programmers? In: Symposium on Engineering Interactive Computing Systems, pp. 13–22. ACM (2014)
35. Pianini, D., Montagna, S., Viroli, M.: Chemical-oriented simulation of computational systems with ALCHEMIST. J. Simul. **7**(3), 202–215 (2013)
36. Pianini, D., Beal, J., Viroli, M.: Improving gossip dynamics through overlapping replicates. In: Lluch Lafuente, A., Proença, J. (eds.) COORDINATION 2016. LNCS, vol. 9686, pp. 192–207. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39519-7_12
37. Pianini, D., Dobson, S., Viroli, M.: Self-stabilising target counting in wireless sensor networks using Euler integration. In: 11th SASO Conference. IEEE (2017)
38. Pianini, D., Viroli, M., Beal, J.: Protelis: practical aggregate programming. In: ACM Symposium on Applied Computing (2015)
39. de Sá, M., Churchill, E.F.: Mobile augmented reality: a design perspective. In: Huang, W., Alem, L., Livingston, M. (eds.) Human Factors in Augmented Reality Environments, pp. 139–164. Springer, Heidelberg (2012). https://doi.org/10.1007/978-1-4614-4205-9_6
40. Schmidt, D.C., Stal, M., Rohnert, H., Buschmann, F.: Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects. Wiley, Hoboken (2000)
41. Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L.: Edge computing: vision and challenges. IEEE Internet Things J. **3**(5), 637–646 (2016)
42. Stoller, S.: Leader election in asynchronous distributed systems. IEEE Trans. Comput. **49**(3), 283–284 (2000)
43. Vaquero, L., Rodero-Merino, L.: Finding your way in the fog: towards a comprehensive definition of fog computing. ACM CCR **44**(5), 27–32 (2014)
44. Vernon, V.: Reactive Messaging Patterns with the Actor Model: Applications and Integration in Scala and Akka, 1st edn. Addison-Wesley Professional, Boston (2015)
45. Viroli, M., Audrito, G., Beal, J., Damiani, F., Pianini, D.: Engineering resilient collective adaptive systems by self-stabilisation. ACM Trans. Model. Comput. Simul. **28**(2), 1–28 (2018)
46. Viroli, M., Beal, J., Damiani, F., Audrito, G., Casadei, R., Pianini, D.: From field-based coordination to aggregate computing. In: Di Marzo Serugendo, G., Loreti, M. (eds.) COORDINATION 2018. LNCS, vol. 10852, pp. 252–279. Springer, Heidelberg (2018). https://doi.org/10.1007/978-3-319-92408-3_12
47. Viroli, M., Casadei, R., Pianini, D.: On execution platforms for large-scale aggregate computing. In: ACM Conference on Pervasive and Ubiquitous Computing, pp. 1321–1326 (2016)
48. Walker, P., Amraii, S.A., Chakraborty, N., et al.: Human control of robot swarms with dynamic leaders. In: Conference on Intelligent Robots and Systems, pp. 1108–1113. IEEE (2014)
49. Weyns, D., et al.: On patterns for decentralized control in self-adaptive systems. In: de Lemos, R., Giese, H., Müller, H.A., Shaw, M. (eds.) Software Engineering for Self-Adaptive Systems II. LNCS, vol. 7475, pp. 76–107. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35813-5_4