2019 IEEE International Conference on Services Computing (SCC)

# Engineering Resilient Collaborative Edge-enabled IoT

Roberto Casadei, Mirko Viroli
*Department of Computer Science and Engineering*
*University of Bologna, Cesena, Italy*

Christos Tsigkanos, Schahram Dustdar
*Distributed Systems Group*
*TU Wien, Vienna, Austria*

***Abstract*—Novel scenarios like IoT and smart cities promote a vision of computational ecosystems whereby heterogeneous collectives of humans, devices and computing infrastructure interact to provide various services. There, autonomous agents with different capabilities are expected to cooperate towards global goals in dependable ways. This is challenging, as deployments are within unknown, changing and loosely connected environments characterized by lack of centralized control, where components may come and go, or disruption may be caused by failures. Key issues include (i) how to leverage, functionally and non-functionally, forms of opportunistic computing and locality that often underlie IoT scenarios; (ii) how to design and operate large-scale, resilient ecosystems through suitable assumptions, decentralized control, and adaptive mechanisms; and (iii) how to capture and enact "global" behaviors and properties, when the system consists of heterogeneous, autonomous entities. In this paper, we propose a model for resilient, collaborative edge-enabled IoT that leverages spatial locality, opportunistic agents, and coordinator nodes at the edge. The engineering approach is declarative and configurable, and works by dynamically dividing the environment into collaboration areas coordinated by edge devices. We provide an implementation as a collective, self-organizing workflow based on Aggregate Computing, provide evaluation by means of simulation, and finally discuss properties and general applicability of the approach.**

***Keywords*-self-organization; situated problem solving; decentralized coordination; collective intelligence; edge computing.**

## I. Introduction

The recent evolution towards an increasingly integrated world has at its basis novel types of pervasive and distributed systems fostered by technologies and paradigms such as the Internet of Things (IoT), inducing collectives composed of humans, heterogeneous devices and computing infrastructure. The complex software-intensive systems that emerge are dynamic and composed of autonomous elements with different capabilities, addressing societal challenges within smart cities, health care, energy, and industry. As computational systems permeate more and more critical aspects of human activity, their dependability is a highly-sought quality. Consider, e.g., scenarios like surveillance or urban monitoring/maintenance. A wide array of challenges arise when deployments are within unknown, changing and loosely connected environments characterized by an absence of centralized coordination and global system state. There, one needs system requirements to be satisfied reliably, in spite of different components (humans or devices) coming and going, or disruption caused by failures. Namely, the system should exhibit *resilience*, intended as "persistence and the ability to absorb change and disturbance" [19]. Key to achieving system resilience is decentralization of control facilities, so that operation towards design goals continues unhindered in the face of disruption. We argue that computing entities architecturally located near the network *edge*—hence, close to the IoT end-devices [30] that make up the heterogeneous collective system—should have a prominent role in coordinating system behavior and adaptive distribution. The edge is a first-class entity in our approach, supporting control agents in *(i)* observing and evaluating context and *(ii)* inducing the appropriate course of action for system entities under their control.

Our approach lies within engineering of collaborative systems that operate in a spatial environment, which may be composed of heterogeneous components typically deployed in diverse devices cooperating towards a global design goal. We shall present a methodology and technical framework for engineering resilient collaborative systems by leveraging *Aggregate Computing* (AC [6]), which we instantiate within the edge-enabled IoT. The paper's contributions are three-fold:

1) we address the broad application domain of situated problem solving within systematic engineering of heterogeneous human-edge-IoT systems, which can be instantiated for a variety of scenarios;
2) we expose a set of abstractions and devise a methodology for engineering collaborative systems based on separation of concerns, which the system designer can readily utilize by injecting domain-specific know-how;
3) we define a core technical implementation of our model based on Aggregate Computing, a formal programming paradigm able to compositionally capture adaptive behavior of agent collectives in a global way, with minimal assumptions on reliability of devices and connectivity.

The rest of the paper is structured as follows. Section II gives a high level overview of the engineering aspects of our approach. Methodology and key design concerns are described in Section III, while our problem-solving coordination solution based on Aggregate Computing is presented in Section IV. Section V provides an assessment of resilience aspects achieved, with a case study showing applicability.

2474-2473/19/$31.00 ©2019 IEEE
DOI 10.1109/SCC.2019.00019

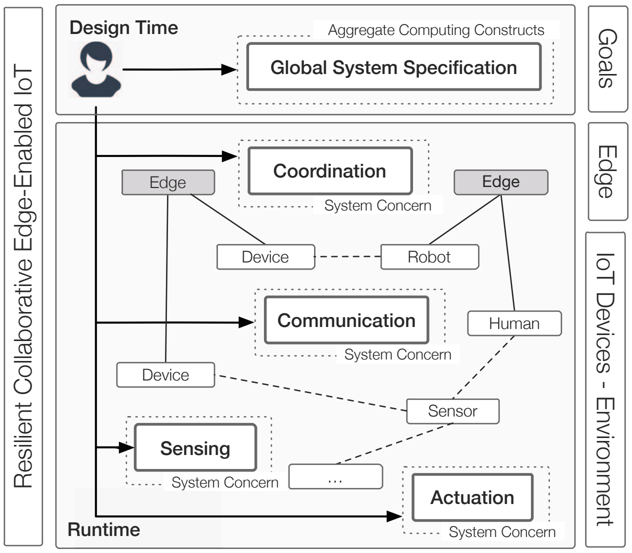

Figure 1: The proposed engineering approach for collaborative edge-enabled systems is configurable w.r.t. a set of design and runtime concerns (as discussed in Section III-B).

Related work is considered in Section VI, and Section VII concludes the paper.

## II. Overview: Engineering Collaborative Problem Solving in Edge-enabled Systems

The approach we propose for engineering resilient collaborative edge-enabled systems is shown in Figure 1 and outlined below.

Modern IoT applications are of varying types and complexities, with multiple software components deployed in diverse domains and contexts. IoT systems supporting distributed applications consist of heterogeneous components (light-border boxes in the figure) —robots, low-powered devices, or humans make up the collective system, which is deployed in constantly changing, unpredictable, and often unknown environments. However, those do not live in isolation, but must interact with each other to collaboratively solve problems as dictated by the requirements of the collective system at hand (such components are depicted as interconnected with dash lines). To facilitate the runtime operation of the resulting system, then, IoT components, situated in spatial localities solving problems, rely on edge nodes for coordination (continuous lines to edge devices).

The complexity that arises in designing effective systems on top of this architecture is addressed in our approach by isolating major IoT system concerns (dark-border boxes at the runtime level): *(i) sensing*, responsible for perception of the environment, *(ii) actuation*, responsible for control actions operating upon the environment, *(iii) communication*, whereby system components interact and exchange information among one another, and *(iv) coordination*, responsible for decentralized organization and control of components in order to ensure progression towards the system goal. Such concerns define a boundary between runtime and design time levels: at design time, they become constructs of a global system specification language, that allows the designers to abstract away low-level details of system components and their interactions, focusing on describing declarative domain-specific behavior instead. Overall, our approach is grounded on formal foundations providing various behavioral guarantees, e.g., ensuring the IoT system can autonomously react to changes in various environment contexts and resiliently cope with unforeseen failures.

**Motivating Scenario.** As a running example of a hybrid problem-solving IoT system, consider a wide smart manufacturing facility populated with machinery, mobile robots and humans. Due to the facility's operation, toxic waste may be spilled in unknown places within the floor. Sensors—or roaming human workers—may detect waste, which due to health hazards must be cleaned by specialized robots. Cleaning robots move to the toxic waste spill area and clean it, upon instruction of various edge-level system control entities responsible for decision-making. Since the system goal is critical—toxic waste is dangerous—the system must be resilient in fulfilling its goal and failure of components must not lead to violation of the system goal. Since the toxic spillage *problem* typically emerges in unknown places, it is to be tackled dynamically by cooperation between different entities (i.e., detected through specialized sensors or human workers, and solved by dispatching cleaning robots) while overall control and coordination must take place in a way that is resilient to failure (e.g., faults in single devices or in the control infrastructure must not lead to global failure).

## III. Model and Methodology

Here, we present the model of our solution and describe how it can be employed from a designer's perspective.

### A. Situated Problem Solving System Model

Our goal is to build a distributed coordination system for large-scale, situated, collaborative problem detection and problem solving. The generalized model we consider is as follows. A system is situated within an *environment*, where *problems* (or *issues*) arise. The environment is inhabited by a (large) set of heterogeneous *agents* (a.k.a. *workers*) making up the IoT system, which roam inside it and interact opportunistically. Workers have *sensors* and *actuators*, to perceive the environment for potential issues and perform repairing actions, as well as specific *skills* (a.k.a. *capabilities*)—i.e., an ontology of pragmatic or epistemic actions potentially useful for the considered problems. Specifically, the key entities we consider for collaborative problem solving are:

- **Environment.** An IoT system's spatial operational environment that needs to be monitored—e.g., the physical area of the smart manufacturing facility.

- **Problems.** Within the system's environment, *problems* (or *issues*) may arise that need to be solved. Those are *situated* (i.e., localized in space and time); e.g., a toxic waste spill occurs in a specific area within the manufacturing facility at some specific time point.
- **Workers.** These are active, situated agents (e.g., IoT devices, humans, robots) that inhabit the environment and are part of the system. They which opportunistically wander or profitably visit a set of loci of interest to perform tasks. Workers may be heterogeneous, exposing different capabilities w.r.t. detecting or solving problems. For the toxic waste scenario, things or humans enjoy problem detection capabilities (i.e., sensing toxins), while specialized robots are responsible for solving problems (i.e., actuating—cleaning toxic waste). When a worker detects a problem, it cannot autonomously decide how to deal with it, and must report the issue to another entity responsible for decision-making.
- **Coordinators.** Resource-rich computational entities, deployed on the edge, are responsible for coordinating workers. A coordinator takes decisions about issues it has been notified about, which result in assignment of *tasks* to workers under its supervision. Workers which detect problems notify their coordinator, who subsequently allocates appropriate tasks to worker(s) with the necessary capabilities. Coordinators themselves may differ in their decision-making ability or computational power.

The system entities above are not static: they interact towards the global system goal. When a worker detects a problem, either the worker is allowed to directly handle it, or not. The former case is of course rather simplistic and assumes no need for coordination for solving problems; the worker may solve the problem by itself if it owns needed capabilities, or may delegate tasks to other workers. However, in the latter case, which is the one we focus on in this paper, the worker cannot autonomously take decisions about how to deal with the problem, and must report the problem to another entity responsible for decision-making, the *coordinator*. Coordinators are responsible for determining the *assignment* (a.k.a. *allocation*) of problems/tasks to workers. The level of sophistication of such decision-making carried out by the coordinator is of course left to the system designer to implement in a domain-specific manner. The coordinator might elaborate a (partial) *plan*, hence assigning (partial) sequences of actions to workers, or it might just *delegate* the issue to the workers, assuming they have the knowledge to do the planning themselves—in this paper, we mainly consider the latter option. Additionally, a coordinator is expected to play a role throughout the problem solving process, i.e., by also supervising or monitoring the activity of workers and providing any needed help. Accordingly, workers engaged in a task can provide *feedback* to the coordinator in order to report progress, request further resources, or provide any information useful for the specific and overall workflow.

A smart choice of coordinators is generally desired, where "smart" depends on various factors; typically, this means choosing nodes that both guarantee *(i)* good and uniform spatial coverage of the environment, and *(ii)* balanced coverage of workers—which might be unevenly distributed across space. We support *multiple coordinators*, each of which is responsible for a certain portion of the environment (called an *area)*. Decision-making is decentralized, as coordinators control sets of workers independently.

However, components in a system might fail, as is especially the case in the highly distributed, volatile IoT-based systems we target. Assuming workers are generally available, failure of coordinators must be tackled, as workers cannot solve problems by themselves and their coordination is critical for achieving the system goal. To this end, we support dynamic selection of coordinators in case of failure: *candidate coordinators* are system components which (in varying degrees) are able to perform coordination duties (e.g., as being resourceful or trusted enough). Out of candidate coordinators, some *leaders* are elected (active coordinators), responsible for a set of workers; those not elected (i.e., inactive) are considered "backup coordinators". Failure of an elected, active coordinator leads to dynamic, automatic selection of a backup one; thus, the system is resilient to their failure.

### B. Capturing Problem-Solving System Concerns

The model of Section III-A defines key abstractions, and relationships among them, essential to the problem solving conception. According to Figure 1, the system designer can take the problem solving workflow as a functional black box: she just needs to provide inputs/configuration and refine the abstractions with domain-specific details. Methodologically, the following needs to be defined.

**Problems Model.** A taxonomy of the problems has to be defined, together with associated properties and metadata for use, e.g., in allocation decision-making. In the toxic waste removal scenario, a spillage problem can be modeled, e.g., by specifying the location in the environment, the kind of substance, and the rough amount of material to be disposed.

**Agents Model.** The agents as well form a taxonomy. In the toxic waste removal scenario, we may have human or robot detectors, and three (possibly overlapping) solver roles: waste collector, disposer, and cleaner. So, sensors and actuators have to be defined and provided: e.g., the waste collector may be equipped with a camera, a pump, and mechanical arm. Agents have capabilities—crucial for problem allocation; e.g., waste collectors and disposers may advertise their ability to carry on light or heavy loads, or their resistance to hot or acid substances. Finally, we only assume that an agent is able to communicate , at the

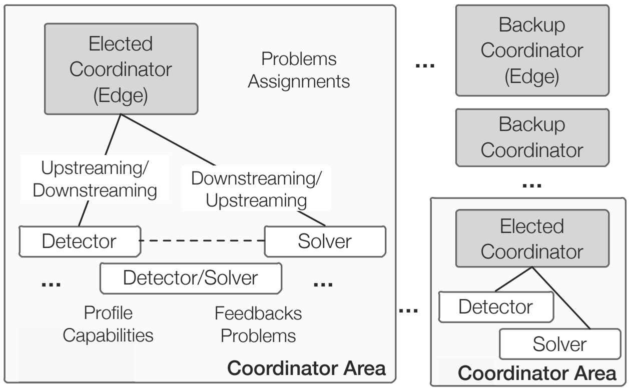

Figure 2: Problem Solving Ecosystem.

minimum, with other nearby agents—the concrete modality being a design decision.

**Solving Processes.** The allocation strategy used by the coordinator to assign tasks/problems to solvers has to be designed, weighing various variables (e.g., required, preferred, and optional skills, solver-to-problem distance, urgency etc.) in an ad-hoc manner, and possibly leveraging heuristics and ML techniques for optimization purposes. Also, the solving process for each problem type has to be designed in terms of a micro-level workflow, expressed e.g., via finite-state automata. This includes identifying phases and states of the activity, pre- and post-conditions preventing or enabling progress, and corresponding feedback messages for the coordinator. E.g., the spillage problem solving workflow can be captured as going through states $\{start, collected, disposed, cleaned, disposed\&cleaned\}$; through feedback, the coordinator can mobilize cleaners and disposers once the $collected$ stage is reached.

**System and Environment Design.** From the definition of sensors and actuators follows the model of the environment as perceived by an agent. Moreover, other elements of the environment and overall system can be specified, including number of agents; number of edge servers (candidate coordinators); number and dimension of areas; and infrastructural elements such as wireless access points for communication.

## IV. Coordinating Problem Solving at the Edge

In this section, we introduce the engineering paradigm we adopt, and describe the core of our solution, i.e., the aggregate specification of a heterogeneous "collective system" carrying out the collaborative problem solving process in edge-centric IoT—as modeled in Section III.

### A. Background: Aggregate Computing

Aggregate Computing (AC [6]) is a paradigm for engineering collective adaptive systems. Its key idea is to describe the system-level behavior of an entire *aggregate* of agents (a.k.a. *collective*, or *ensemble*) through a single *global specification* (i.e., an *aggregate program*) that is locally interpreted by each constituent agent according to the aggregate semantics and the agent's *local context*, which is given by the portion of environment that it can observe (through *sensors*) and messages received from *neighbor agents*. Collective behavior and properties are obtained through patterns of computation and interaction whereby individual activity affects the corresponding neighborhood and in turn global portions of the system, through a sort of "controlled emergence". Then, adaptivity is achieved by "continuously" interpreting the aggregate program in order to re-assess individual contexts (by sampling the environment and observing activity in the neighborhood) and provide up-to-date responses seeking global coherence, in turn affecting the rest of the system.

The AC model roots in few ideas. Structurally, an aggregate system is a set of agents organized into a (dynamic) logical network based on a *neighboring relationship*. Behaviorally, devices repeatedly compute an aggregate program at a certain configurable, possibly non-homogeneous frequency; i.e., computation evolves at asynchronous rounds of execution spaced out by "sleeping" periods. Interactionally, communication is only possible with neighbors.

By a programming viewpoint, AC is *compositional* and *declarative*. Aggregate programs can be defined by *functionally composing* aggregate-level building blocks that abstract from various low-level details, such as the actual topology of the system or the concrete unfolding of execution. Critically, this abstraction allows the *AC platform/middleware* to drive aggregate systems in a flexible and optimized way [39]. AC is supported on the JVM by ScaFi [12], a Scala library that includes a runtime, internal-DSL and actor-based middleware [13].

The formal framework that actualizes AC is based on the *(computational) field* notion and the corresponding *Field Calculus (FC)* [4]. A field is a distributed data structure that maps agents to computational values. A field computation is a function from input fields to output fields. For instance, a *gradient* computation that measures the minimum hop-by-hop distance from any agent to "source" agents can be modeled at the aggregate-level as a function from a $boolean$ field ($true$ for the source agents and $false$ for others) to a $double$ field of distances. Such functional orientation, "preserved" by the semantics of field primitives, is instrumental for the *compositionality* of the approach. The field primitives cover the natural lifting of local values and operations to fields, application of a field of functions (also splitting the domain of computation into *branches*), stateful evolution of fields, and interaction with neighbors. Note that field compositions, within some formalized conditions, have been proved to enjoy interesting properties, and to preserve such properties across composition; a notable example is *self-stabilization* [37], whereby a system, in the absence of further perturbations, is guaranteed to eventually reach a stable output (or fixpoint). A full technical presentation of AC and FC is beyond the scope of this paper; a complete account on the matter can be found in [4].

In this paper, we explore the fitness of AC for IoT and Edge Computing. We argue the AC paradigm helps to address prominent issues in such scenarios – e.g., complex decentralized, situated coordination, self-organizing behavior, and flexible deployment – as it provides a straightforward model to represent situated entities interacting in a spatial locality, and to horizontally scale, from few up to large numbers of agents, while keeping the flexibility of mapping the logical system to diverse physical architectures in the edge-fog-cloud continuum [39].

### *B. Aggregate Specification of the Collaborative Workflow*

We refine the model described in Section III as per Figure 2 and accordingly implement the situated, collaborative problem solving workflow through the specification of Figure 3[1], expressed in the SCAFI DSL. Such a specification should be intended as a script to be fully executed asynchronously and in a repeated fashion by each agent, and whose execution carries on in a "coordinated" way by means of the SCAFI platform. This program describes the workflow by a global perspective through a continuous process mixing coordination and computation. Concretely, an aggregate program is an `AggregateProgram` subclass implementing the `main` method with the desired field expression meant to be collectively executed in an aggregate fashion, i.e., continuously in asynchronous rounds of computations intertwined with neighboring communication acts.

The logic of `ProblemSolvingEcosystem` is straightforward. First, the system determines the coordinators. Operator **`S`**`(grain)` is an well-known aggregate building block that computes a boolean field holding **`true`** for elected leaders, ensuring they are at a mean distance `grain` between each other; this has the effect of partitioning the network into "areas" controlled a single leader. Function **`priorityS`** is a variant of **`S`** which takes into account a numeric `priorityField` to prioritize leader election—in general, there might be some preferences or constraints for eligibility of agents for coordination (e.g., based on computational capabilities or trust metrics).

Based on the leader set, a gradient field `potential` is maintained to support data propagation from coordinators to agents in the corresponding area (*downstreaming*), and vice versa (*upstreaming*). Such a potential field is built only considering (by partitioning the computation domain through **`branch`**) agents available for this data distribution process, as indicated by boolean field `infoPropagationNet`; note that "leaf" agents would still be able to send and receive data (but won't participate in the dissemination itself).

As the spatial structure for hop-by-hop data transmissions is set up (`potential` field), it is used to *collect*, from workers to coordinators, three pieces of data: *(i)* new issues that have been identified (`problems`); *(ii)* profiles of workers (`solvers`), e.g., as descriptions in terms of availability or advertised skills; and *(iii)* data about ongoing problem solving by workers (`feedbacks`), which may include resource requests or event occurrences such as confirmation of task acceptance or completion. Operation **`collectSets`** is for merging locally emitted data sets along a spanning tree from leaves (workers) to roots (coordinators); it derives from the aggregate collection operator **`C`**`(potential,acc,v,none)`, where merging function `acc` is the set union, `v` is the set to be collected, and `none` (emitted from nodes with no parent) is the empty set.

So, the coordinators use such an upstream data flow to decide new *assignments* (a.k.a. *allocations*) of problem instances to solvers, which are then propagated downstream via a collective **`broadcast`** activity. Such function leverages the generic aggregate operator **`G`**`(potential,acc,field)`, using for `acc`umulation the identity function—the value to be broadcast is taken from `field` where `potential` is zero (i.e., in source points), and preserved along the transmission path.

Finally, each device takes the downstream field `tasks` of allocations for execution. This data structure contains all the allocations for an area, so each device needs to consult the map for its own problem assignments. Note that devices "outside" `infoPropagationNet` will receive the allocation map, but will not contribute to the gossip process.

### *C. Resilience through Consensus and Coordination*

From a dynamical perspective, the system is based on three self-healing, ongoing processes: *(1)* adaptive decentralized consensus for election of active coordinators; *(2)* adaptive creation of areas and corresponding data pathways between leaders and workers; and *(3)* adaptive upstreaming and downstreaming of data between leaders and workers. Such aggregate processes take time to build and take time to adjust to change, depending on the entity of perturbations. There is a direct, I/O dependency among them: as the output of *(1)* is the input of *(2)*, and the output of *(2)* is the input of *(3)*. A change in the set of active coordinators will result in a different shape of areas and, consequentially, different data pathways. Mobility of workers also creates perturbations to the gradient computation producing the potential field, which in turn affects the spanning tree used for data collection.

Computational fields are not snapshots: they are continuously evolving device-to-value mappings that might be somewhat incoherent during their transient phase, before they stabilize or reach the desired property. For instance, in spatial leader election, leaders are progressively elected to

[1] For space reasons, we do not introduce the Scala syntax and every bound name in the program. The full source is available at the repository provided in Section V. Refer to [12] for further details on the field-based functions included in the SCAFI standard library or straightforward extensions thereof.

```
class ProblemSolvingEcosystem extends AggregateProgram with ProblemAPI {
  override def main = {
    val coordinators = priorityS(grain, priorityField)
    val potential = branch(infoPropagationNet){gradient(coordinators)}{+∞}
    val problems  = collectSets(downTo=potential, problemOccurrences)
    val solvers   = collectSets(downTo=potential, solverProfile)
    val feedbacks = collectSets(downTo=potential, feedbackField).groupBy(_.problem)
    val assignments = branch(coordinators){
      allocate(coordinators,solvers,problems,feedbacks) }{ Set() }
    val tasks       = broadcast(potential, assignments)
    branch(workers){ execute(tasks) }{ () }
} }
```

Figure 3: Excerpt of the aggregate program modeling situated problem solving as a decentralized workflow. Gray, underlined symbols denote fields of parameters (e.g., `grain`) or built-in/sensor values (e.g., `solverProfile`). Black, bold symbols denote application-specific functionality. Red and purple symbols denote core and library constructs, respectively.

ensure a uniform coverage of space, but it takes time before such property is guaranteed. Different aggregate computing algorithms and techniques can be used to promote particular dynamical properties of system processes: for instance, for collecting information, the potential field should be built with a gradient algorithm that focuses more on the shape of the output field rather than on a precise estimation of distances. An in-depth discussion of these algorithms, which is subject of intense research [38], is beyond our scope; also, the reader should be aware that there is there is ongoing research on analysis and synthesis of effective implementations for aggregate operators [38], [20], [3], [21].

Resilience naturally emerges from the adaptiveness of the three aforementioned processes: if a leader fails, another will be elected; if the leader formation changes (i.e., if the potential field loses a source and gains a new one), it will self-heal (depending on the gradient implementation, this might be more or less fast [3]); if the potential field changes, the streaming of data will flow along a different path. However, the point is not only eventual consistency, but also what properties are preserved during the transient phase. These are usually application-specific and might include, e.g., consistency guarantees. For instance, if the formation of areas is changing, it might happen that a certain problem reaches more than one coordinator. If that is a problem, precautions have to be taken, e.g., by explicitly denoting the target coordinator in the messages. The failure of workers, or a worker losing ability to communicate with its neighbors (a kind of local failure) is not a problem as long as this does not result in network partitioning (which would prevent the non-local propagation of information). Indeed, if a worker fails, then it will stop streaming its status to the coordinator, which will be aware of that and adjust its assignments.

## V. Evaluation

To evaluate the proposed approach, we illustrate a case study of urban infrastructural maintenance in smart cities and set up an experimental framework with simulations based on the ScaFi-Alchemist platform [12][2]. Our evaluation's focus is on functional correctness, resilience, and on the actual automatic triggering of adaptivity mechanisms[3].

### *A. Case Study: Infrastructural Maintenance in a Smart City*

As a case study, consider a scenario where autonomous agents (e.g., robots) and human workers are collectively employed for maintaining a city's infrastructure. As parts of the city's common facilities may break or degrade, issues must be quickly identified and dealt with appropriate actions. This entails the notification and resolution of issues by the active agents operating within it. Non-autonomous agents might be useful as well: cameras and diffused plain sensors may provide data to smart software components which are capable of inferring semantics and contributing to the system. The issues arising in the city's facilities are *situated*, i.e., they have an identity and location in space-time. Agents may use electromagnetic sensors, smoke/gas sensors, cameras, or even accept inputs by citizens to detect potential problems. Naturally, agents who identify issues might not be able to solve those by themselves: they may not have required skills or enough resources to deal with the problem, and hence they have to report it to a "control center". Note that issues might be dealt with, in principle, in a completely decentralized way: the agent who finds an issue may locally broadcast requests for specialists or resources, without involving any central entity, and the closest matching agents would respond.

Such a problem setting fits our approach particularly well. We advocate keeping the system quite decentralized, by splitting it into areas of space of reasonable size, while

[2] ScaFi is available at: https://github.com/scafi/scafi. The source code of simulations as well as instructions for running the experiments and generating the plots are available at the repository https://github.com/metaphori/engineering-collaborative-edge-iot.

[3] Detailed quantitative analyses, such as how delays in work allocation vary with the grain of areas, will be considered in extensions of this work.

also introducing centralization points (the coordinators) to provide more sophisticated/optimized coordination and decision-making. Coordinators should be placed in strategic/central places of the city, and as they may have to optimize decisions, they should be resourceful machines—e.g., a cloudlet [31] or an edge computer. We assume security countermeasures are taken for the system to be safe, as well as that potential coordinators outnumber required coordinators (e.g., for redundancy) and have legal ability to carry on their tasks. A smart coordinator might choose to allocate problems to workers based on elements like problem severity, skills of workers, or distance from workers to problems; especially when heterogeneous teams are needed to deal with complex issues, such an allocation decision is not an easy one to be left to a self-organizing team of workers.

### B. Experimental Setup and Simulation Framework

For our experiments, we employ the aggregate specification of Figure 3, enriched with simulation-specific code for parametrization and data gathering. The experimental setting and simulation scenario (depicted in Figure 4) are as follows. A number of devices are supposed to be deployed in the city center of Vienna: $300$ lightweight devices and $10$ edge servers. Two devices can communicate if they are within $50$ meters range. All these devices, including edge servers, are assumed to run the AC middleware as a service and the aggregate application described by the program in Figure 3 on top. They are assumed to "fire" (i.e., to run computation rounds and send corresponding data to neighbors) asynchronously but at similar frequencies.

We run simulations considering either "smart" coordinators (which use an advanced allocation strategy of problems to workers—abstracting from the concrete one) or "naive" coordinators, as indicated by a $smartness$ boolean parameter. We measure, along time: *(i)* the total number of problems detected by all workers, *(ii)* the problems streamed to the coordinator but still unhandled, *(iii)* the problems both allocated to *and* accepted by at least one worker (i.e., those successfully assigned), and *(iv)*, the total number of problems handled to completion. We observe the system response by injecting problem occurrences and failure as described in Figure 5. Our experiments are implemented as ScaFi simulations [12] and available online.

### C. Experimental Results

The experimental results are reported in Figure 5. Our evaluation goals concern a qualitative assessment of correctness, adaptivity and resilience of the system. Assumptions and explanations of *how* these are achieved are in Section IV-C.

**Correctness.** Evidence comes from the fact that all the problems found have been managed: this means that both the notification of problems, the task allocation, and the feedback process work well. Moreover, notice how the injection of a blackout, disconnecting edge servers from the network (between $t_2 = 300$ to $t_3 = 310$), provides a delay but does not affect the outcome. Finally, differences emerge between coordinators that use an advanced allocation strategy and naive ones: overall, one can observe the increase of performance when smart allocation decisions are taken.

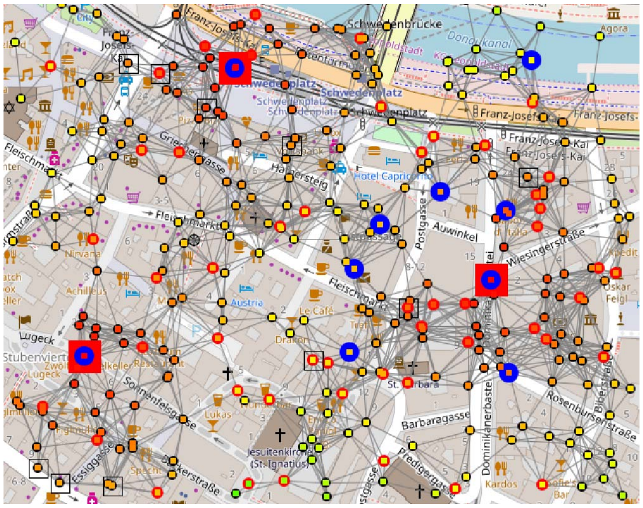

Figure 4: Snapshot of the simulation scenario. Large, blue nodes represent edge devices eligible for election as coordinators; large red-filled squares denote leaders. Small circles represent workers; their filling color reflects the potential field (warmer colors when closer to coordinators). Square contours (e.g., bottom-left corner) denote nodes currently working on a problem. Gray edges depict neighboring links.

**Adaptivity.** When the active coordinators fail, the system self-organizes to elect new coordinators. This results into an adaptation of the structures supporting the data flows.

**Resilience.** Resilience naturally emerges: despite coordination failures, group formation changes or general faults in the control infrastructure, the system does not fail: it correctly responds to failures through appropriate coordination reactions. Notice the gentle degradation of performance caused by failure and the restoration of conventional efficiency.

### D. Discussion

**Functional Perspective.** Elements about the functional correctness of the solution are provided in Section IV-C and empirically verified in this section. The solution schema in Section IV is the core of the approach but may not satisfy all the functional properties needed by a real-world application. For instance, the designer needs to decide what happens when an area lacks resources for specific problems or coordinators do not receive timely feedback.

**Non-Functional Perspective.** *A) Bandwidth and storage.* The amount of data that needs to be propagated depends

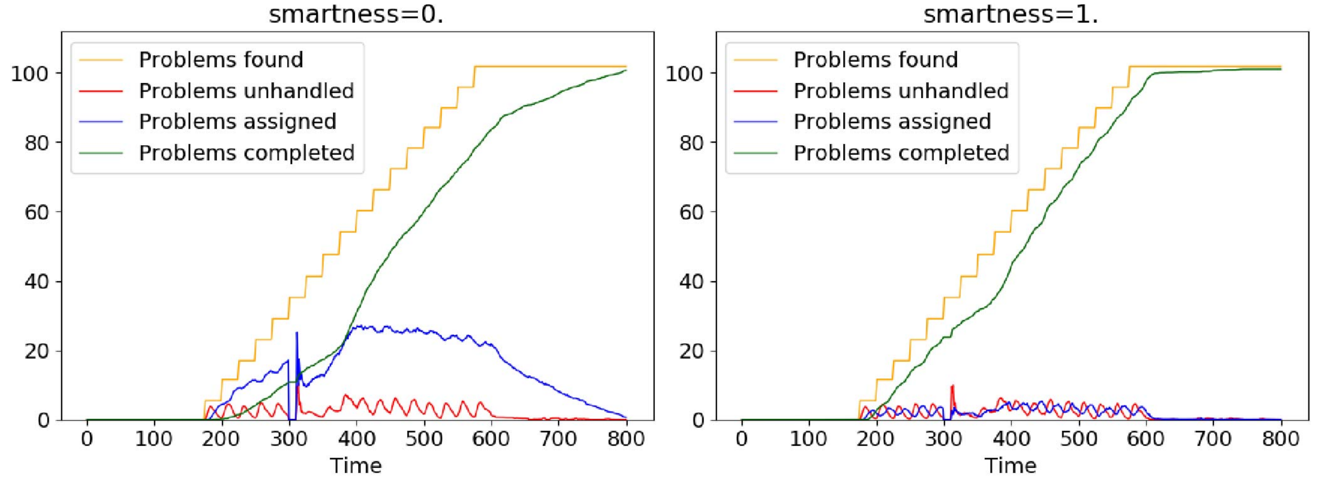

Figure 5: Aggregated results of multiple simulation runs, with “naive” (left) and “smart” (right) coordinators. For each case, 50 simulation instances are executed for different random seeds, and the mean values are taken for the measured quantities. From $t_0 = 150$ to $t_1 = 600$ time units, a significant number of “problems” are randomly generated, so that they can be detected by worker devices. Moreover, from $t_2 = 300$ to $t_3 = 310$, a blackout is injected, with the effect of temporarily detaching the edge servers from the network.

on the number of nodes in each area, the amount of problem solving activity, and amount of data required by the coordinators (concerning problem reporting, solver profiles, and feedbacks). Storage is needed because data propagation through aggregate operators `G` and `C` requires keeping state. The specification in Figure 3 partially deals with this by using a subset of nodes for the epidemic distribution of data in each area. *B) Latency.* In the simplest case, the time between problem identification and resolution is $T_C + T_A + T_G + T_S$ where $T_C$ is the time needed to collect problem data to the coordinator, $T_A$ is the time needed by the coordinator to make its allocation decision, $T_G$ is the time needed to transmit the allocation decision, and $T_S$ is the time needed by the worker to solve the problem. In particular, $T_C$ and $T_G$ are proportional to `grain` and depend on the firing frequency of nodes and the propagation delay, whereas $T_A$ and $T_S$ are application-specific. While it is useful to be aware of these performance aspects, it must be noticed that a variety of optimizations can be applied to the execution process globally sustaining an aggregate application [39]: messages can be compressed (or include only deltas), round frequency can be dynamically adjusted according to desired QoS, communications might be optimized through localisation in edge access points—what is possible ultimately depends on assumptions, configuration, and available infrastructure.

**Usability.** A description of what the designer must define and what is provided by our approach is given in Section II and Section III-B. For the large part of application design, the designer can focus on the business logic, filling in the gaps with specifications of problems, agents, solution workflows, coordination, and environment. However, knowledge about AC and its toolchain is required for deployment and implementation of extensions with respect to the basic workflow.

**On Real-World Deployment.** Generally, for a real-world deployment, the physical devices that are going to be part of the aggregate system need to host and run two main software artifacts: (i) the AC middleware [39], which provides an interface to physical capabilities (i.e., sensors, actuators, and communication interfaces) as well as logical services to applications and (ii) the particular aggregate application, which consists of an aggregate program (as the one in Figure 3) and configuration metadata, and leverages services provided by the middleware. Edge provisioning and code deployment patterns [28] might be adopted to distribute the aggregate logic as a containerized application. Security and privacy are also relevant concerns: Attacks such as the forgery of communication data can compromise aggregate algorithms [10] (think about the effect of a counterfeit potential field). So, public-key cryptography might be used to ensure that (i) original aggregate applications are executed, e.g., using signed containers, and (ii) aggregate-level coordination data is not manipulated. In [27], the authors propose to use blockchain technology to transparently support security at the level of the aggregate computing middleware.

**Generality and Extensions.** The specification of Figure 3 represents a general solution schema that can be specialized for different situated problem solving applications, whose key design dimensions are explained in Section III-B. Straightforward examples are those in smart cities, such as infrastructural maintenance (Section V-A), but include generally any scenario involving situated monitoring, decision-making, and action (e.g., firefighting, car crash management, etc.). Depending on the specific scenario, extensions may mix "centralized" and "localized" decision-making, mix "opportunistic" and "planned" monitoring, allow collaboration globally or among adjacent areas, or balance the distribution of skills/resources among areas according to certain metrics (e.g., occurrences of problems, or criticality).

## VI. Related Work

In this paper, we address large-scale, situated problem-solving coordination in Edge/IoT scenarios through a spatial, self-organizing approach with minimal assumptions on reliability of devices and connectivity (neighbor-based communication is sufficient for system operation). Other works addressing similar problems—e.g., in multi-agent systems (MAS) research [24]—typically adopt different focus and assumptions. We abstract individual planning of workers and any task scheduling optimization by orchestrators and rather focus on coordination in a dynamic edge environment. Hierarchical MAS frameworks with control loops such as [26] (which is tailored for smart cities) usually lack compositional programming abstractions and do not leverage a global or spatial stance.

In this paper, we address large-scale, situated problem-solving coordination in Edge/IoT scenarios through a spatial, self-organizing approach with minimal assumptions on reliability of devices and connectivity. It should be noticed that implementation in AC is natural but not essential for the proposed solution—the crucial part being its *dynamics*. Still, a novelty of this paper also lies in the adoption of AC techniques in this new scenario, differing from typical uses [38] such as rescue operations, distributed event recognition [13], swarm sensing [7], and crowd management in safety-critical scenarios [6] and opportunistic IoT [11]. Indeed, these works do not explicitly consider the edge layer, and address specific scenarios rather than classes of applications.

Resilience concepts have been largely investigated in fields ranging from critical and dependable systems [33], CPSs [16], [2], and networks [29] and cloud computing [18]. Engineering support similarly ranges from theoretical foundations [34], operations and process [8] and formally backed system validation [17], and fault management [5], [14]. Within robotics, resilience in the form of convergence in the presence of byzantine faults [9] has been investigated, however assuming no direct communication. Architectural aspects of WSNs—in particular interaction and connectivity—are treated in [25]. Cloud-IoT couplings have been investigated for automatic management, analysis and control of IoT systems [35], pertinent to resilience [15]. Such hierarchical IoT multi-tier architectural models are used for gathering data to perform analytics on the cloud, where techniques for resilience are deployed at different tiers [36]. In contrast, we assume systems that are edge-enabled and decentralized. Also, our model is generic and applicable to a wide array of edge-enabled systems within smart environments [1]. We note the current absence of usable approaches towards programming abstractions for resilience tailored for contemporary IoT systems.

On the platform side, an outline of architectural styles and deployment tactics for aggregate systems is given in [39], showing how AC can exploit IT infrastructures comprising edge/fog/cloud layers. This is related, e.g., to works on elastic [22] and osmotic computing [23], where ecosystems of people, processes, and things are dynamically managed considering both infrastructural and application requirements. Our point is that elasticity in edge/cloud environments has to be enabled by declarative programming models.

Works in the context of on collaborative systems also share some focus with this paper. E.g., in [32], a model for hybrid collaborative adaptive systems is proposed in which the designer specifies an environment where collectives—i.e., persistent or transient teams of peers (humans and machines)—are involved in collective tasks. W.r.t. our approach, such model is more articulated, human-centered and focuses on orchestration of complex collaborative tasks in small-scale social computing scenarios.

## VII. Conclusion and Future Work

We proposed resilient situated problem-solving coordination in edge-enabled IoT settings, providing a general model that defines key abstractions and workflow relationships which can be extended and specialized with application- and domain-specific aspects. We further provided an AC specification implementing the workflow in a resilient, self-organizing way, demonstrated its correctness and discussed its properties. We stress that adopting AC is convenient but not a requirement of the approach.

Regarding future work, we would like to extend the quantitative analysis of our approach as well as apply it to more realistic scenarios. Moreover, we would like to focus in more detail on the heterogeneity of teams, and better explore the role of humans therein, e.g., to study what is the impact of additional unpredictability on the dynamics of collective workflows, and how to seamlessly integrate humans in a self-organizing cyber-physical aggregate.

## References

[1] Abreu, D.P., Velasquez, K., Curado, M., Monteiro, E.: A resilient internet of things architecture for smart cities. Annals of Telecommunications **72**(1-2), 19–30 (2017)

[2] Arlat, J., Diaz, M., Kaâniche, M.: Towards resilient cyber-physical systems: The adream project. In: Design & Technology of Integrated Systems In Nanoscale Era, 2014 9th Int. Conf. On. pp. 1–5. IEEE (2014)

[3] Audrito, G., Casadei, R., Damiani, F., Viroli, M.: Compositional blocks for optimal self-healing gradients. In: 11th IEEE Int. Conf.on Self-Adaptive and Self-Organizing Systems, SASO. pp. 91–100. IEEE Computer Society (2017)

[4] Audrito, G., Viroli, M., Damiani, F., Pianini, D., Beal, J.: A higher-order calculus of computational fields. ACM Transactions on Computational Logic **20**(1), 5:1–5:55 (2019)

[5] Avresky, D., Arlat, J., Laprie, J.C., Crouzet, Y.: Fault injection for formal testing of fault tolerance. IEEE Transactions on Reliability **45**(3), 443–455 (1996)

[6] Beal, J., Pianini, D., Viroli, M.: Aggregate programming for the Internet of Things. IEEE Computer **48**(9) (2015)

[7] Beal, J., Usbeck, K., Loyall, J., Rowe, M., Metzler, J.: Adaptive opportunistic airborne sensor sharing. ACM Transactions on Autonomous and Adaptive Systems (TAAS) **13**(1), 6 (2018)

[8] Bernardi, S., Merseguer, J., Petriu, D.C.: Dependability modeling and analysis of software systems specified with uml. ACM Computing Surveys (CSUR) **45**(1), 2 (2012)

[9] Bouzid, Z., Potop-Butucaru, M.G., Tixeuil, S.: Optimal byzantine-resilient convergence in uni-dimensional robot networks. Theoretical Computer Science **411**(34-36), 3154–3168 (2010)

[10] Casadei, R., Aldini, A., Viroli, M.: Towards attack-resistant aggregate computing using trust mechanisms. Science of Computer Programming **167**, 114–137 (2018)

[11] Casadei, R., Fortino, G., Pianini, D., Russo, W., Savaglio, C., Viroli, M.: Modelling and simulation of opportunistic IoT services with aggregate computing. Future Generation Computer Systems **91**, 252–262 (2019)

[12] Casadei, R., Pianini, D., Viroli, M.: Simulating large-scale aggregate MASs with Alchemist and Scala. In: FedCSIS, Proceedings of. pp. 1495–1504. IEEE (2016)

[13] Casadei, R., Viroli, M.: Programming actor-based collective adaptive systems. In: Programming with Actors, LNCS, vol. 10789, pp. 94–122. Springer (2018)

[14] Cristian, F.: Understanding fault-tolerant distributed systems. Communications of the ACM **34**(2), 56–78 (1991)

[15] Delic, K.A.: On resilience of IoT systems: The internet of things (ubiquity symposium). Ubiquity **2016**(Feb), 1 (2016)

[16] Denker, G., Dutt, N., Mehrotra, S., Stehr, M.O., Talcott, C., Venkatasubramanian, N.: Resilient dependable cyber-physical systems: a middleware perspective. Journal of Internet Services and Applications **3**(1), 41–49 (2012)

[17] Ghosh, R., Kim, D., Trivedi, K.S.: System resiliency quantification using non-state-space and state-space analytic models. Reliability Engineering & System Safety **116**, 109–125 (2013)

[18] Ghosh, R., Longo, F., Naik, V.K., Trivedi, K.S.: Quantifying resiliency of iaas cloud. In: Reliable Distributed Systems, 29th Symposium on. pp. 343–347. IEEE (2010)

[19] Holling, C.S.: Resilience and stability of ecological systems. Annual review of ecology and systematics **4**(1), 1–23 (1973)

[20] Mo, Y., Beal, J., Dasgupta, S.: Error in self-stabilizing spanning-tree estimation of collective state. In: FAS*W, Proceedings of. pp. 1–6. IEEE (2017)

[21] Mo, Y., Beal, J., Dasgupta, S.: An aggregate computing approach to self-stabilizing leader election. In: FAS*W, Proceedings of. pp. 112–117. IEEE (2018)

[22] Moldovan, D., Copil, G., Dustdar, S.: Elastic systems: Towards cyber-physical ecosystems of people, processes, and things. Computer Standards & Interfaces **57**, 76–82 (2018)

[23] Nardelli, M., Nastic, S., Dustdar, S., Villari, M., Ranjan, R.: Osmotic flow: Osmotic computing + IoT workflow. IEEE Cloud Computing **4**(2), 68–75 (2017)

[24] Olfati-Saber, R., Fax, J.A., Murray, R.M.: Consensus and cooperation in networked multi-agent systems. Proceedings of the IEEE **95**(1), 215–233 (2007)

[25] Oteafy, S., Hassanein, H.: Resilient iot architectures over dynamic sensor networks with adaptive components. IEEE Internet of Things Journal **4**(2), 474–483 (2017)

[26] Patrascu, M., Dragoicea, M., Ion, A.: Emergent intelligence in agents: A scalable architecture for smart cities. In: System Theory, Control and Computing (ICSTCC), 2014 18th International Conference. pp. 181–186. IEEE (2014)

[27] Pianini, D., Ciatto, G., Casadei, R., Mariani, S., Viroli, M., Omicini, A.: Transparent protection of aggregate computations from byzantine behaviours via blockchain. In: Proceedings of the 4th EAI International Conference on Smart Objects and Technologies for Social Good. pp. 271–276. ACM (2018)

[28] Qanbari, S., Pezeshki, S., Raisi, R., Mahdizadeh, S., Rahimzadeh, R., et al.: IoT design patterns: computational constructs to design, build and engineer edge applications. In: 1st Int. Conf. on Internet-of-Things Design and Implementation (IoTDI). pp. 277–282. IEEE (2016)

[29] Rak, J.: Principles of communication networks resilience. In: Resilient Routing in Communication Networks, pp. 11–43. Springer (2015)

[30] Ren, J., Guo, H., Xu, C., Zhang, Y.: Serving at the edge: A scalable IoT architecture based on transparent computing. IEEE Network **31**(5), 96–105 (2017)

[31] Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N.: The case for vm-based cloudlets in mobile computing. IEEE pervasive Computing **8**(4) (2009)

[32] Scekic, O., Schiavinotto, T., Videnov, S., Rovatsos, M., Truong, H.L., Miorandi, D., Dustdar, S.: A programming model for hybrid collaborative adaptive systems. IEEE Transactions on Emerging Topics in Computing (2017)

[33] Siewiorek, D., Swarz, R.: Reliable Computer Systems: Design and Evaluatuion. Digital Press (2017)

[34] Stoicescu, M., Fabre, J.C., Roy, M.: Architecting resilient computing systems: overall approach and open issues. In: International Workshop on Software Engineering for Resilient Systems. pp. 48–62. Springer (2011)

[35] Suciu, G., Vulpe, A., Halunga, S., Fratu, O., Todoran, G., Suciu, V.: Smart cities built on resilient cloud computing and secure internet of things. In: Control Systems & Computer Science (CSCS), Int. Conf. on. pp. 513–518. IEEE (2013)

[36] Uddin, M.Y.S., Nelson, A., Benson, K., Wang, G., Zhu, Q., et al.: The scale2 multi-network architecture for IoT-based resilient communities. In: Smart Computing (SMARTCOMP), Int. Conf. on. pp. 1–8. IEEE (2016)

[37] Viroli, M., Audrito, G., Beal, J., Damiani, F., Pianini, D.: Engineering resilient collective adaptive systems by self-stabilisation. ACM Transactions on Modeling and Computer Simulation **28**(2), 16:1–16:28 (2018)

[38] Viroli, M., Beal, J., Damiani, F., Audrito, G., Casadei, R., Pianini, D.: From field-based coordination to aggregate computing. In: Int. Conf. on Coordination Languages and Models (COORDINATION), pp. 252–279. Springer (2018)

[39] Viroli, M., Casadei, R., Pianini, D.: On execution platforms for large-scale aggregate computing. In: Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct. pp. 1321–1326. ACM (2016)