Towards Adaptive Flow Programming for the IoT: the Fluidware Approach

Franco Zambonelli[1] and Mirko Viroli[2] and Giancarlo Fortino[3] and Barbara Re[4]

***Abstract*—The objective of this position paper is to present Fluidware, a proposal towards an innovative programming model for the IoT, conceived, to ease the development of flexible and robust large-scale IoT services and applications. The key innovative idea of Fluidware is to abstract collectives of devices of the IoT fabric as sources, digesters, and targets of distributed "flows" of contextualized events, carrying information about data produced and actuating commands. Accordingly, programming services and applications implies declaratively specifying "funnel processes" to channel, elaborate, and re-direct such flows in a fully-distributed way, as a mean to coordinate the activities of devices and realize services and applications. The potential applicability of Fluidware and its expected advantages are exemplified via a case study scenario in the area of ambient assisted living.**

I. Introduction

Programming services and applications for the IoT may involve the composition and coordination of a multitude of heterogeneous devices, possibly dispersed over a wide (e.g., urban) area. This scenario embeds many sources of complexity: the potential high number and density of deployed devices; the high frequency (possibly of an almost continuous nature) of computational events to occur and be managed; the need to self-adapt to short-/medium-/long-term changes and faults; the need to operate on top of a dynamically evolving infrastructure comprising IoT/edge/cloud devices and resources; the need of promptly (often in quasi real-time) reacting to sophisticated space-time situation recognition; the potential of running complex, goal-oriented orchestrations of distributed activities across devices of heterogeneous computational power (from tiny devices to cloud servers).

The above issue can make it very hard to develop and deploy dependable services and applications exhibiting predictable behavior, especially when using traditional service composition approaches that would require explicit handling of a multitude of possible exceptions and alternate scenarios [17]. Accordingly, there is compulsory need of novel programming approaches that make it possible to avoid dealing with mundane complexity details and rather rely on high-level constructs enabling to express in full the potentials of the IoT fabric.

In this position paper, we intend to show how a novel programming model for IoT services and applications, along with the associated supporting platform, engineering methodology and tools, can be conceived to ease the development of flexible and robust large-scale IoT services and applications. Starting from previous findings in the areas of field-based coordination [22], [12], collective adaptive systems [1], stream computing and aggregate computing [2], this project will address the complexity of building modern, large-scale IoT systems, by a full-fledged engineering approach revolving around a new notion of distributed programming.

Our proposal, which we call "Fluidware", relies on the innovative idea to abstract collectives of devices of the IoT fabric (both sensors and actuators) as sources, digesters, and targets of distributed flows of contextualized events, carrying information about data produced and manipulated over time. Accordingly, programming services and applications implies specifying "funnel processes" to channel, elaborate, and re-direct such flows in a fully-distributed way, as a means to coordinate the activities of devices and realize services and applications.

Funnel processes can be specified in a declarative way, in terms of how they consume and produce events over space and time. They can be associated to distributed and contextualized streams (i.e., flows) of events in terms of advanced pattern-matching mechanisms based on semantics of data and space-time conditions on their production. Thus, the specification of funnel processes totally abstracts from the actual number and type of devices to which events belong, enabling to define scale-independent computational activities inherently addressing self-adaptation to contextual conditions, and smoothly fitting various computing/network infrastructures. Indeed, a proper supporting platform (i.e., a middleware [14]) will be put in place to take care of the actual deployment of funnel processes and their transparent allocation and replication.

Overall, the Fluidware approach, once fully developed, will support the bottom-up construction of complex IoT applications through a correctness-guaranteed stack of software components, incrementally encompassing basic building blocks of stream/event manipulation, libraries for distributed coordination, and reusable IoT services on top.

II. Motivations and Case Study Scenario

The key motivations for Fluidware are to attack a number of challenges that currently hinder the possibility to easily develop complex and large-scale IoT systems and applications.

Device independence. Current IoT approaches are highly device-dependent, assuming the existence of specific types of devices in specific locations. However, the heterogeneity of devices, their ephemerality, and the impossibility to individually control all devices in an environment, requires programming approaches that enable services to be programmed and expressed in device-independent terms.

*This work was not supported by any organization

[1]F. Zambonelli is with the University of Modena and Reggio Emilia, Italy franco.zambonelli at unimore.it

[2]M. Viroli is with the University of Bologna, Italy mirko.viroli at unibo.it

[2]G. Fortino is with the University of Calabria, Italy giancarlo.fortino at unical.it

[2]B. Re is with the University of Camerino, Italy barbara.re at unicam.it

Scalability. IoT services may involve individual sensors and actuators, or compose and coordinate a limited number of local devices, but also exploit a multitude of cooperating devices distributed over a wide area. An approach for programming and deploying IoT services should adopt the same basic model for small and for large-scale services, and should not lose in effectiveness when scaling.

Adaptivity. IoT systems are called to operate in very dynamic environments. Devices can be ephemeral, mobile, or become unreachable. Yet, IoT services and applications must adaptively react to dynamics and be highly available to serve reliably despite contingencies.

Service-orientedness. The current perspective on the IoT is strictly service-oriented: an IoT is conceived as a provider of one or more services, and composite services are created by exploiting traditional service composition approaches. This does not acknowledge that most IoT devices are instead producer of almost continuous flows of events and data.

Seamless integration of devices, edge, and cloud levels. The overall IoT fabric will include multitude of distributed devices, edge computers that can act as "local clouds" for the devices in a locality, and general cloud resources. To exploit the IoT fabric at the best, there is thus need of supporting both direct device-to-device interactions, edge computation, and cloud resources, all within the same design and programming abstraction.

Interoperability and security. Enabling interoperability among heterogeneous devices and ensuring security of IoT systems and services are key challenges to enable the widespread diffusion of IoT services.

To clarify the above issues, let us a specific case study in the healthcare area, yet representative of a larger class of IoT scenarios such as smart homes and smart cities [20]: an IoT enriched rehabilitation center. In such center, inhabited by patients in needs of assisted rehabilitation, we assume that all the ambients of the rehabilitation center are densely enriched with connected sensors and actuators: light and heat controllers, gas and smoke detectors, presence and motion sensors, doors (main doors, internal doors, fridge, kitchen furniture) sensors, electric consume sensors, shutter/curtain controller, as well as sensorized everyday objects (e.g, cup, fork, cane). Moreover, also medical devices (e.g., pulse-oximetry, smart scale) may be provided to patients in order to automatically send health status information and measures.

In such a scenario, IoT devices can be exploited to realize a variety of different services to support both medical doctors in the monitoring and care activities of individuals, to help individuals and their family members in their everyday self-managed healthcare activities, and to control the overall conditions of the rehabilitation center. However, If the rehabilitation center is a very big one, the number of individual devices embedded in it can be very large and heterogeneous, thus programming services by having to account for the individual characteristics of each devices can be very hard. One should also consider that devices can easily become unreachable, or their proper functioning could be undermined by patients, calling for an approach that adaptively tolerates devices unavailability. Also, given the potentially very high-number of devices scattered over a possibly large area, an approach promoting scalability is necessary.

Given the safety-critical nature of the scenario, many services related to e.g., the health condition of patients or the ambient conditions of the rehabilitation center may be required to continuously send information to be analyzed to different actors (from doctors to the hotel maintenance center). Thus, any approach for developing services and application should more properly conceive sensors and actuators as producers and consumers of continuous flows of events, rather than as loci of services to be invoked.

At the architectural level, it is clearly possibly to realize services and application in a centralized way, by having all data from sensors by re-directed to some central cloud, where it can be analyzed and – depending on the services to be realized – commands for the actuators can be eventually issues. However, given the high-number of sensors and actuators, and the inherent real-time nature of services for patients (where potentially dangerous health conditions or simply dangerous behaviors by patients needs immediate actions) also call for decentralized approaches where the needed sensorial information can immediately flow by the proper actuators.

Finally, as in most other scenarios, interoperability and security are necessary towards ease of development and towards trustworthiness, respectively, the latter being particularly critical given the involvement of personal clinical data.

III. The Fluidware Approach

The Fluidware approach considers IoT-enriched environments densely populated with a variety of ICT devices (overall forming the"IoT fabric"), acting as sensors or actuators or both. Sensing devices generate "contextualized streams" of data representing events about something that is happening somewhere in the environment. Actuating devices receive "contextualized commands streams" related to what they should do over time.

A. Funnel Processes

The starting innovative idea of the project is that IoT services and applications can be realized by means of "funnel processes", acting as digesters and producers of widely-distributed streams of events, involving collectives of devices, and which we shall also call "event flows" (or simply "flows"). Namely, funnel processes are able to capture event flows, elaborate them, send/(re)distribute them over the network of distributed IoT devices or over the edge/cloud (see Figure 1).

Funnel processes will be specified in a declarative way independently of their actual allocation and distribution (managed by the Fluidware middleware). Their specification will be such to define which flow to connect to, depending on contextual/spatio-temporal and semantic matching (e.g., "in room X now", "where temperature is greater than 25 degrees"). Thus, specification will totally abstract from the actual devices that events belong to: it can be such to include a limited number of local events from a limited number of devices, but also a large-number of devices spread over a large-scale. This enables to define scale-independent computational activities, inherently independent of external conditions and smoothly fitting various computing/network infrastructures. Most specifically, an event

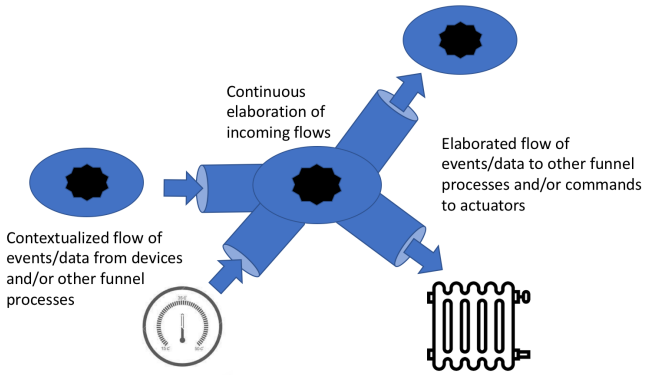

Fig. 1. A general abstract representation of funnel processes exploited locally.

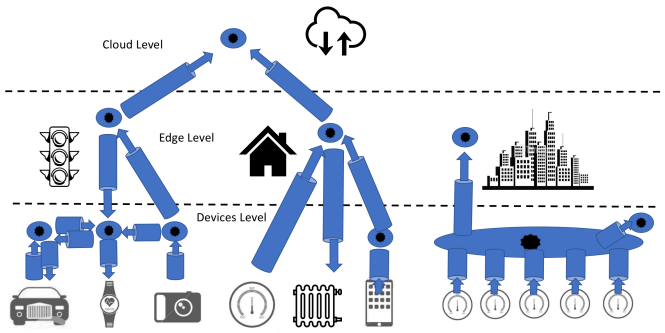

Fig. 2. Overview of the Fluidware approach.

flow is manipulated computationally to produce a new flow, by a combination of mechanisms typical of stream processing frameworks and of self-adaptive and self-organizing systems, such as aggregation, spreading, persistence, and so on.

B. Deployment Scenarios

At the implementation and deployment level, the Fluidware platform/middleware will take care of creating actual processes and connecting them to the flows of actual devices, to realize funnel process specifications. Such processes can be deployed on any IoT device with enough resources to support their execution. Opportunistically, funnel processes can be deployed at the level of edge computers associated to some specific location and having access to all devices in that location [16], or even at the level of some centralized cloud.

Clearly, when allocated to individual devices, a process can pre-digest and elaborate the data sensed by that device before forwarding it to some other devices or to some collection point. However, it can also be seamlessly used to spread and re-distribute events to nearby devices, and itself contribute absorbing and possibly aggregating events coming from nearby devices. This can be used to realize composite services with a direct device-to-device coordination. This scenario is represented in the left part of Figure 2.

In addition, a funnel process to be connected to an event flow spread from a multitude of devices, and/or a wide area, can be realized by replicating and distributing multiple actual processes, all logically part of the same distributed funnel process. Such aggregations can be used to realize services and applications in the large-scale by reasoning at the level of collectives of devices and, in the end, to promote scale-independent and device-independent computations, along the lines of field-based coordination and aggregate computing approaches [2], [4]. The right part of Figure 2 represents such a scenario.

On the other hand, when acting at the edge level, funnel processes can be used to collect the events provided by devices at a specific location (e.g., a room, a building, or a plaza), and elaborate them to produce flows of commands to the local actuators, and possibly forwarding streams of data to the cloud. Finally, nothing prevents from associating funnel processes to some cloud servers to realize some sorts of centralized services, either explicitly within funnel process deployment specification [13], or implicitly as an opportunistic choice at the platform level. This scenario is represented in the central part of Figure 2.

We emphasize that, building on previous experience on formal models of distributed and adaptive business processes [7] and in distributed field-based coordination [2], [22], we aim at enriching the Fluidware approach with a formalized description of funnel processes supporting static and behavioral property verification, independently of where the approach acts at the level of individual devices or at the edge/cloud level, and independently of whether it is used to implement traditional composition of services or rather in-the-large collective distributed coordination.

C. Addressing the Challenges

Let us now analyze how Fluidware has the potential to effectively address the challenges identified in Section II, also by making examples related to the case study scenario.

Device independence. In Fluidware, whole collectives of IoT devices are abstracted as producers (or consumers, for actuators) of distributed streams of contextualized events, thus making system programming dependent on the availability of specific flows of events, not on the specific identity of the devices producing or consuming them.

In the case study, accelerometers on smart phones or wrist devices produces flow of data that can be used to detect and classify activities of patients. We envision in Fluidware to be possible to define a generic funnel process able to "digest" such streams, apply a classifier function to them, so as to produce a flow of classified data. The funnel process needs not to be tied to a specific accelerometer, but can associated to any flow of acceleration data associated to a specific patient, e.g.,[1]

```
def classifier
      [accelerometer(bind-to patientX)
       -> classifier_function
-> activity(patient X)]
```

Along with accelerometers, it is possible to think at capturing the flow of all cameras in the room of a patients, or at the level of the whole rehabilitation center, to detect activity via image processing and generate an additional "activity" flow.

[1]We emphasize the syntax and semantics of Fluidware are still under definition, thus the examples has to be considered pseudocode.

All the flows related to activity recognition can then feed a "alarm-generator" that can recognize dangerous situations for patients and can trigger alerts in the medical personnel.

To monitor environmental situations, we can for instance define a funnel process to monitor temperature in the rehabilitation center and signal potential unusual situation (e.g., a fire). Such funnel process can be associated to all the sensors in the center capable of detecting temperature to perform global monitoring, e.g,.

```
def detect-danger
      [temperature(bind_to any)
        ->detection_func
        -> situation(source sensorX)]
```

Another funnel process, can be in turn instructed to monitor the results of the "detect-danger" process to eventually trigger activation of fire extinguishers.

```
def triggger-extinguishers
      [if situation(bind_to any)->"fire"
        ->  identify_location
        ->  activate(extinguishers location)]
```

Scalability. In Fluidware, Funnel processes can be used to access individual streams, as well as – with the same model – to aggregate, compose and control distributed flows of events generated by myriads of devices, thus seamlessly enabling small-scale service composition and large-scale services based on collective behaviors.

As from the examples above, at the programming level, a single funnel process can handle the flows of data multiple devices. In addition, it can aggregate flows from multiple devices to generate distributed data structures to represent global level situations that can be used realize context-dependent flow computations. For instance, upon detection of fire in some rooms of the center, one can think at spreading an aggregating the related information across the whole building, so as to form a distributed "field" [22], [2] to represent, in each location, the direction of fire locations and the distance from it:

```
def fire-field
      [if situation(bind_to here)->"fire"
        ->  aggreggate(fire-fields)
        ->  spread(fire-fields)
```

Adaptivity. In Fluidware, Funnel processes are not statically tied to specific IoT devices, but are dynamically bound to any flows of events matching contextual and semantic characteristics. Thus, they can operate by dynamically reconnecting to different sets of devices, simply depending on their characteristics, whenever needed to react to contingencies.

In the example above, no matter if some of the temperature sensors crashes, or if new ones are added, there is not need to update the detect-danger processes, and not even to re-start them.

Service-orientedness. In the Fluidware vision, IoT devices are not loci of services, but producers and consumers of data/event flows. This enables a more dynamic and real-time handling of situations.

It should be clear from the example above that the definition of funnel processes that coordinate the activities of sensors and actuators conceives devices in a very different way than as simple loci of services.

Seamless integration of devices, edge, and cloud levels. Funnel processes may be playing this role of re-directing streams of data to the edge, where further processes may digest them and re-direct them to realize composite edge services, or they can be used to re-direct streams to the cloud.

For instance, depending on the capabilities of the devices hosting accelerometers, one can think at deploying the "classifier" process them directly on them or rather at some centralized server. Similarly, for the "detect-danger" process, one can think at having a single process in a centralized control center, or at having one process in each room of the center. We expect the Fluidware middleware, possibly guided by some "allocation" directive, to automatically handle the allocation and possible replication of Funnel processes.

Interoperability and security. Concerning interoperability, the possibility in Fluidware to program services in a device-independent way promotes interoperability, and just requires devices to host a a Fluidware local proxy, or - in the case of very lightweight IoT devices - to directly communicate via standard IoT protocols to Fluidware edge devices. However, of course, funnel processes requires adhering to a common semantics to agree on the "meaning" of the flow they digest, e.g., in the above examples, they require sharing the concept of "activity" and the meaning of activity classes.

Concerning security, this is an open issue, and we are still in the process of understanding how to protect the integrity of applications from malicious funnel processes.

IV. The Path Towards Fluidware

Let us now present the different research activities that we are currently undertaking to make Fluidware a solid reality.

A. Programming Model

From the development and programming viewpoint, we expect that IoT services and IoT applications will be mostly up to expert system developers. Yet, we expect that the Fluidware declarative approach will also enable local managers of a location (and, to some limited extent, end users) to be able to directly personalize the behavior of such applications with simple forms of user-level programming of funnel processes, for instance, in a similar way to "if this then that" approaches [8] or by selecting specific funnel processes from libraries of reusable specifications.

To formalize the Fluidware operational model and implement its basic programming interface, we plan to undertake the following activities:*(i)* Developing the operational model of funnel processes, to serve as a blueprint for implementation of the platform, for defining composition techniques, and to check well-formedness of specifications and properties. It will include development of a core calculus, operational semantics, and by-construction proofs of self-stabilization and

safe encapsulation. *(ii)* Implementing a library to provide the core mechanisms devised in the model, to specify and compose processes, as an interface towards the platform and existing simulators. We envisage the adoption of modern techniques to smoothly integrate with mainstream programming and functional-oriented declarative approaches.

B. Middleware

We expect Fluidware be supported by a middleware capable of instantiating local proxies of funnel processes and launch them in execution into the proper location [14], and to relocating them as needed. To promote scalability and flexibility, the platform will support interactions and coordination at three levels (as from Figure 2): direct device-to-device interactions (e.g., for field-based coordination), with funnel processes directly instantiated on sets of devices, and with events flowing, aggregating and re-distributing from device to device; edge level, with funnel processes dynamically allocated on edge computers (i.e., cloudlets or fog computers), to digest streams of events, implement local coordinated services, and possibly to connect multiple edges to realize inter-edge coordination; cloud level, for centralized monitoring, coordination, and storage.

The activities to develop the Fluidware middleware platform will address three related objectives: *(i)* To define a mapping from Fluidware programming specifications into a set of distributed components to be deployed atop the Fluidware platform. Mapping will be based on a model-driven development and will distinguish abstract platform-independent specifications from deployable platform-dependent components. *(ii)*To implement a distributed engine for supporting the execution of Fluidware systems and services. It will be organized as a three-layered (devices, edges, and cloud) super-peer architecture. Platform-dependent components can be dynamically activated and re-configured in IoT devices, edge servers and/or cloud platforms. *(iii)* To analyze interoperability and security issues. Without the ambition of developing innovative solutions, the task will define guidelines for enabling devices to connect to the Fluidware platform, and will analyze how a trust-oriented distributed infrastructure for inter-component security can be integrated within it.

C. Engineering Methodology

The Fluidware approach will also require the definition of new conceptual abstractions to reason about complex IoT services and applications and their requirements. In addition, it will call for the identification of specific methodological guidelines to drive the design and development process, to be necessarily accompanied by Fluidware-specific tools to support the activities of the development process, and to provide correctness guarantees [11], [21]..

Accordingly, there is need of synthesizing from the previously identified activities in order to make available the basic engineering instruments supporting of the analysis, design and development of complex IoT systems and services with Fluidware. To this end, we plan to: *(i)* Identify the key conceptual abstractions around which the analysis and design of Fluidware systems and services should rely on, and on this basis it will define guidelines for analysis, design, development, testing, and adaptation of Fluidware services and systems. *(ii)* Identify and prototype a set of tools in support of the development of Fluidware systems and services. These will include support to verify service behavior against specifications (by model-checking or static analysis) and a large-scale simulator to verify overall system behaviors.

D. Application Studies

We expect Fluidware to be a general-purpose approach, suitable for developing IoT services and applications in a variety of emerging scenarios, such as smart homes, smart cities, traffic control systems, energy control systems, and smart production systems.

To keep focus without losing generality, we intend to initially focus on a specific on scenarios of ambient assistant living similar to that of the case study scenario here adopted [8]. This will enable putting the Fluidware approach at work in real-world problems, to guide its development strategies, and to assess the effectiveness of the approach. In particular, we intend to test ambient assisted living application both at the scale of a real-life domestic indoor testbed and of a large-scale simulated urban scenario. The domestic testbed will be aimed at verifying ease and expressiveness of programmability of Fluidware, as well as the effectiveness of the platform and its adaptability properties. The large-scale simulated scenario will be useful to verify Fluidware scalability and flexibility.

V. Related Work

The Fluidware computational model definitely owns to methods proposing simplifying distributed programming by abstracting from individual networked devices, and working at the level of their collective behaviors, such as SCEL [9], SAPERE [22], or DECCO [4]. These also include space-time models for the universal manipulation of field data structures diffused in space and evolving with time, as in field-based [22] and aggregate computing [2]. Fluidware, with its concept of funnel processes, will advance such approaches by enabling the seamless modeling of small-scale composite services as well as large-scale collective services, and by supporting the integration of semi-decentralized approaches with fully distributed ones. Also, by exploiting the lessons of process algebraic [6] and formal approaches [3], Fluidware can add a process layer which *(i)* internally carries on stream computation, and *(ii)* externally defines life-cycle aspects such as funnel process generation, space-time extension, interaction with environment, and deallocation.

By taking inspiration from frameworks to automatically split data processing behavior for cloud- and cluster-style execution (e.g., Spark [19], and Flink [5]), Fluidware will enrich traditional collective and aggregate approaches to distributed programming by considering a transition from handling collective ”fields” of data to the notion of distributed stream of events, thus addressing also non-functional aspects concerning the control of dynamic aspects of event generation and diffusion.

Some recent approaches to programming IoT services propose new computational abstractions as building blocks of IoT services, such as the microservices of Osmotic computing [18], the core processors of EdgeIoT [16], or the deployment units

of the Elastic Computing [13]. These approaches share with Fluidware the idea of enabling the adaptive deployment of such building blocks at the level of both cloud or edge computers, and possibly at the level of IoT devices. However, Fluidware will enrich that with an operational semantics addressing dynamically multi-layered architectures, with the possibility of also acting in collective terms at the level of devices to enforce large-scale adaptive service composition.

Concerning middleware, a variety of platforms have been proposed to support the deployment and execution of IoT services and applications (see [14] for a survey) and including solutions to adaptively handle interoperability [10], context-dependency [15], and adaptivity [22]. Similarly to them, the Fluidware platform will promote interoperability (thanks to device-independence of the funnel process abstraction), adaptivity (thanks to the flexible deployment of funnel processes) and context-dependency (events digested by funnel processes are inherently contextual). However, it will also provide solutions for the collective execution and coordination of funnel processes on the large-scale.

VI. Conclusions

In this article, we have presented Fluidware, a proposal for an innovative programming model for IoT services and applications, conceived, to ease the development of flexible and robust large-scale IoT services and applications.

The key innovations that Fluidware promises to bring about include: *(i)* Its funnel process abstraction can go significantly beyond the state-of-the-art in computational models for distributed and collective systems, allowing to declaratively express distributed processes managing contextualized streams of events in a way that is effectively scale-independent, device-independent, and adaptive; *(ii)* The interplay between Fluidware model and middleware will fully provide utility-driven exploitation of infrastructure, enabling execution of funnel processes along the entire IoT/edge/cloud stack; *(iii)*The definition of novel software engineering abstractions, methodologies, and tools to support activities performed to develop Fluidware applications, can shed new lights into the general software engineering issues associated to the development of complex IoT systems and services in general, independently of Fluidware.

Currently, we are in the process of developing the presented ideas, in the hope to actually deliver the Fluidware identified potentials.

References

[1] Vasilios Andrikopoulos, Antonio Bucchiarone, Santiago Gómez Sáez, Dimka Karastoyanova, and Claudio Antares Mezzina. Towards modeling and execution of collective adaptive systems. In *International Conference on Service-Oriented Computing*, pages 69–81. Springer, 2013.

[2] Jacob Beal, Danilo Pianini, and Mirko Viroli. Aggregate programming for the internet of things. *IEEE Computer*, 48(9):22–30, 2015.

[3] Lenz Belzner, Matthias Hölzl, Nora Koch, and Martin Wirsing. Collective autonomic systems: Towards engineering principles and their foundations. In *Transactions on Foundations for Mastering Change I*, pages 180–200. Springer, 2016.

[4] Tomás Bures, Frantisek Plasil, Michal Kit, Petr Tuma, and Nicklas Hoch. Software abstractions for component interaction in the internet of things. *IEEE Computer*, 49(12):50–59, 2016.

[5] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4), 2015.

[6] Yeongbok Choe and Moonkun Lee. Algebraic method to model secure iot. In *Domain-Specific Conceptual Modeling*, pages 335–355. Springer, 2016.

[7] Riccardo Cognini, Flavio Corradini, Andrea Polini, and Barbara Re. Business process feature model: an approach to deal with variability of business processes. In *Domain-Specific Conceptual Modeling*, pages 171–194. Springer, 2016.

[8] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. A semantic web approach to simplifying trigger-action programming in the iot. *Computer*, 50(11):18–24, 2017.

[9] Rocco De Nicola, Diego Latella, Alberto Lluch-Lafuente, Michele Loreti, Andrea Margheri, Mieke Massink, Andrea Morichetta, Rosario Pugliese, Francesco Tiezzi, and Andrea Vandin. The SCEL language: Design, implementation, verification. In *Software Engineering for Collective Autonomic Systems - The ASCENS Approach*, pages 3–71. 2015.

[10] Giancarlo Fortino, Wilma Russo, Claudio Savaglio, Weiming Shen, and Mengchu Zhou. Agent-oriented cooperative smart objects: From iot system design to implementation. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, (99):1–18, 2017.

[11] Ivar Jacobson, Ian Spence, and Pan-Wei Ng. Is there a single method for the internet of things? *Queue*, 15(3):20, 2017.

[12] Marco Mamei and Franco Zambonelli. Programming pervasive and mobile computing applications: The TOTA approach. *ACM Trans. Softw. Eng. Methodol.*, 18(4):15:1–15:56, 2009.

[13] Daniel Moldovan, Georgiana Copil, and Schahram Dustdar. Elastic systems: Towards cyber-physical ecosystems of people, processes, and things. *Computer Standards & Interfaces*, 57:76–82, 2018.

[14] Andrei Palade, Christian Cabrera, Gary White, Mohammad Abdur Razzaque, and Siobhán Clarke. Middleware for internet of things: a quantitative evaluation in small scale. In *2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–6. IEEE, 2017.

[15] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Context aware computing for the internet of things: A survey. *IEEE communications surveys & tutorials*, 16(1):414–454, 2014.

[16] Xiang Sun and Nirwan Ansari. Edgeiot: Mobile edge computing for the internet of things. *IEEE Communications Magazine*, 54(12):22–29, 2016.

[17] Thiago Teixeira, Sara Hachem, Valérie Issarny, and Nikolaos Georgantas. Service oriented middleware for the internet of things: A perspective. In *Proceedings of the 4th European Conference on Towards a Service-based Internet*, ServiceWave'11, pages 220–229, Berlin, Heidelberg, 2011. Springer-Verlag.

[18] Massimo Villari, Maria Fazio, Schahram Dustdar, Omer Rana, and Rajiv Ranjan. Osmotic computing: A new paradigm for edge/cloud integration. *IEEE Cloud Computing*, 3(6):76–83, 2016.

[19] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016.

[20] Franco Zambonelli. Toward sociotechnical urban superorganisms. *IEEE Computer*, 45(8):76–78, 2012.

[21] Franco Zambonelli. Key abstractions for iot-oriented software engineering. *IEEE Software*, 34(1):38–45, 2017.

[22] Franco Zambonelli, Andrea Omicini, Bernhard Anzengruber, Gabriella Castelli, Francesco L. De Angelis, Giovanna Di Marzo Serugendo, Simon A. Dobson, Jose Luis Fernandez-Marquez, Alois Ferscha, Marco Mamei, Stefano Mariani, Ambra Molesini, Sara Montagna, Jussi Nieminen, Danilo Pianini, Matteo Risoldi, Alberto Rosi, Graeme Stevenson, Mirko Viroli, and Juan Ye. Developing pervasive multi-agent systems with nature-inspired coordination. *Pervasive and Mobile Computing*, 17:236–252, 2015.