Case studies for a new IoT programming paradigm: Fluidware*

Stefano Mariani[1], Roberto Casadei[2], Fabrizio Fornari[3], Giancarlo Fortino[4], Danilo Pianini[2], Barbara Re[3], Wilma Russo[4], Claudio Savaglio[4], Mirko Viroli[2], and Franco Zambonelli[1]

[1] DISMI, University of Modena and Reggio Emilia
{stefano.mariani, franco.zambonelli}@unimore.it
[2] DISI, University of Bologna
{roby.casadei, danilo.pianini, mirko.viroli}@unibo.it
[3] School of Science and Technology, University of Camerino
{fabrizio.fornari, barbara.re}@unicam.it
[4] DIMES, University of Calabria
{g.fortino, w.russo}@unical.it, csavaglio@dimes.unical.it

Abstract. A number of scientific and technological advancements enabled turning the Internet of Things vision into reality. However, there is still a bottleneck in designing and developing IoT applications and services: each device has to be programmed individually, and services are deployed to specific devices. The Fluidware approach advocates that to truly scale and raise the level of abstraction a novel perspective is needed, focussing on device ensembles and dynamic allocation of resources. In this paper, we motivate the need for such a paradigm shift through three case studies emphasising a mismatch between state of art solutions and desired properties to achieve.

Keywords: Fluidware · IoT programming · coordination.

1 Introduction

To reveal the full potential of large-scale IoT deployments demands for novel models and programming abstractions [6, 22]. The need to adapt to short, medium, and long-term contingencies, to operate on a dynamically evolving infrastructure comprising the full spectrum of Edge, Fog, and Cloud devices and resources, to react to the inner dynamics of the environment (there including the space-time fabric) in near real-time, and to execute goal-oriented orchestration of distributed activities across devices (with a varying degree of computational power), raises challenges yet to be addressed by currently available programming languages and executing platforms [22, 16, 14].

* Work Supported by the Italian MIUR, PRIN 2017 Project "Fluidware". Copyright©2019 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0)

A novel programming model for IoT services and applications has been conceived in [8] with the purpose of tackling the aforementioned challenges: grounded on results in the areas of *attribute-based coordination* [1, 2, 7], *field-based coordination* [13, 21], *collective adaptive systems* [4, 18], and *aggregate computing* [5, 6], the proposed "Fluidware" framework aims to address the complexity of designing and deploying large-scale IoT systems. Fluidware revolves around the core idea of abstracting away devices of the IoT fabric (sensors, actuators, gateways, etc.) as *sources*, *digesters*, and *targets* of *distributed flows* of *contextualised events* delivering information about data produced, manipulated, and consumed, over time and across space. Programming services and applications then amounts at designing "funnel processes" to *capture*, *process*, *manipulate*, and *route* such flows in a fully distributed way, as a means to coordinate the activities of devices towards achievement of system (application or service) goals. Funnel processes are *declarative* specifications defining *what* event flows to consume and produce over time and across space, operating on distributed and contextualised event streams in terms of pattern-matching mechanisms considering both semantics of data and space-time constraints.

In this paper we advocate the need for such a paradigm shift in designing and developing IoT services and applications by discussing three envisioned deployment scenarios: a smart retirement centre, a smart university campus, and a generic stress-free workplace. Accordingly, the paper is organised as follows: Section 2 overviews the Fluidware approach to IoT systems engineering; Section 3 describes the issues in developing IoT services according to current practice, and how Fluidware may overcome them, in a smart retirement centre scenario, whereas Sections 4 and 5 do the same for a university campus and a generic workplace, respectively; finally, Section 6 provides final remarks.

2 Fluidware

The key idea of Fluidware is that IoT services and applications can be conceived and designed as transformations of widely distributed *event streams* ("flows") encompassing *ensembles* of devices. This is achieved by means of the "funnel process" abstraction, acting as both producer and consumer of flows, and able to capture, merge, split, replicate, redirect, filter, process, namely manipulate them in time and space, over the network of distributed IoT devices, and ranging the whole spectrum from Edge to Cloud. As a consequence, Fluidware raises the abstraction level for developers willing to design an IoT service or application: they no longer think in terms of programming physical devices and orchestrating their communications, but rather in terms of programming purely logical processes (the funnels) dealing with event streams detached from the physical devices actually producing and consuming them—as depicted in Figure 1.

Funnel processes are *declaratively* specified in a totally *agnostic* and *orthogonal* way with respect to their actual allocation to computational nodes, and distribution across the Edge-Cloud spectrum—which are aspects delegated to the Fluidware middleware. Such a specification defines *which* flows to capture

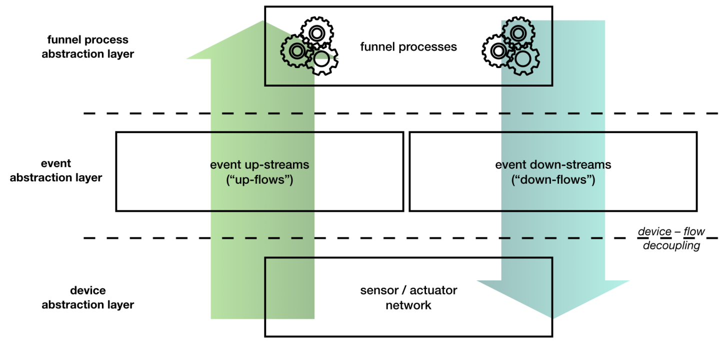
funnel process abstraction layer
funnel processes
event abstraction layer
event up-streams ("up-flows")
event down-streams ("down-flows")
device – flow decoupling
device abstraction layer
sensor / actuator network

Fig. 1. Fluidware approach: devices are abstracted away as sources and targets of event flows, captured and manipulated by funnel processes.

and *what* to get out of them depending on contextual, spatio-temporal, and semantic constraints (e.g., "gather all temperatures from this area in the last 15 mins which are above the defined critical threshold, then chart a heatmap"). This enables to define *scale-independent* computational processes inherently to contextual conditions and seamlessly fitting heterogeneous computing and network infrastructures. More specifically, an event flow is manipulated by funnel processes to produce a new flow, by a combination of techniques typical of stream processing and self-organising systems, such as merging, splitting, filtering, aggregation, spreading, and so on [20]. Funnel processes may work as *filters*, selecting events from flows if and only if they match semantic and spatio-temporal criteria defined by a suitable Domain Specific Language (e.g. as for Regular Expressions); *aggregators*, to fuse different event flows into new ones, depending on how their attributes relate to each other, so as to change the *structure* of the flow (e.g. summarising information by averaging data); *transmuters*, that modify the *content* of events in flows, leaving the structure unchanged (e.g. resetting counters for a "rolling horizon" measure); *routers*, that may re-locate, grow, shrink, etc. the span of events in the space-time fabric (e.g. moving a source of events to a more reliable device). Basic primitive operators will take care of manipulating event flows and funnel processes as well, as a higher order calculus, such as splitters and mergers to manipulate cardinality of input and output streams, pipes and multiplexers to handle sequential and parallel composition of streams and funnels, etc. As an example, the following pseudo-code snippet may implement the aforementioned "heatmap charting" funnel:

```
function CRITICAL_TEMP_NEARBY:
    if TEMP(bind-to [area(here), last(15, 'm')]) > thresholds('temp') then
    end if
    sys.viz.chart('heatmap', temp)
end function
```

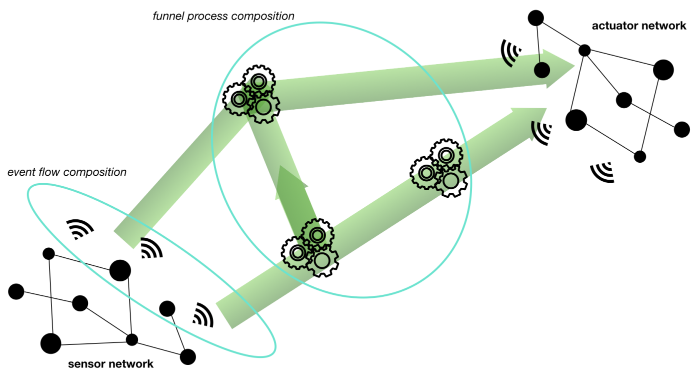

Fig. 2. “Funnels” opportunistically capture “flows” as well as compose with each other.

Indeed, a suitable platform/middleware [16] is a key enabler of the Fluidware approach, as the responsible for the actual *allocation* of funnel processes at the more appropriate layer of an Edge-Fog-Cloud network of devices and computational nodes [19], as well as of their optimised execution and transparent replication/relocation across the network. The Fluidware middleware will thus take care of performing two critical kinds of decision during run-time, both with the goal of fully realising the funnel processes specification: *(a)* which flows to connect to a given funnel process (e.g., all flows in target area, only flows with certain properties, etc.), and how (e.g., as a message stream, through a publish-subscribe blackboard, or with a tuple space), and *(b)* where to deploy such a process (e.g., at the Edge, in the Fog, or on the Cloud), and how (e.g., as individual process, or as part of a swarm of distributed replicas, or as a worker in a map-reduce like architecture).

For instance, a funnel process collecting temperature measures from a given area to chart a heatmap of that area may be actually composed by three sub-processes: one gathering temperatures from sensors, thus directly deployed on-board Edge devices, one aggregating results to average out temperatures depending on sub-areas (e.g., the kitchen, the living room, 1st floor, 2nd floor, left-wing, right-wing, etc.), hence deployed in Fog gateways, and the last one charting the heatmap, deployed on the Cloud (as it needs global information and more computational capacity). Such sub-processes, which can be funnel processes themselves, are *opportunistically* connected to the most suitable event flows (Figure 2), and deployed to the most suitable device (Figure 3). For instance, should the Fluidware middleware detect failure of a sensor, it could replace the connection to the event flow coming from such sensor with another, non faulty one. Also, should the middleware detect a performance drop in processing at the Edge, it could decide to re-locate the correspondent sub-process to an available Fog gateway.

In next sections, we discuss three Fluidware deployment scenarios so as to shed light on why the kind of paradigm shift proposed by Fluidware is needed, how it advances beyond the current state of art in delivering services in the IoT, and what benefits it delivers.

3 Smart Retirement Centre

We envision the scenario of a retirement facility for elderly people or people with chronic diseases, both requiring peculiar care and *continuous assistance*. There, we assume guests can live together with a member of their family or an assistant (social worker, nurse) in small furnished flats, properly equipped with IoT devices and services meant to support health and general lifestyle monitoring (e.g., smart cameras, A/C, illumination, doors, etc.). Also, we assume that the centre has dedicated rooms for providing care to guests (e.g., rehabilitation rooms and spa services) as well as rooms for social and leisure activities (e.g., the canteen, and common living rooms), also equipped with smart devices (e.g., smart TVs, pill boxes, refrigerators, beds, sofas, chairs and wheelchairs, etc.).

Several specific IoT services can be delivered to guests, which usually require coordination of several components (software and hardware), and each having its solutions and issues to deal with—as summarised in Table 1.

3.1 Services

In the flats of the facility, other than services to access and control individual appliances, one can think at providing *coordinated services* that, by accessing and directing the lightening system, the light sensors, and the window curtains in a specific room, can modify the overall light situation of that room depending on the specific needs/preferences of the persons occupying it. Similarly, sensors

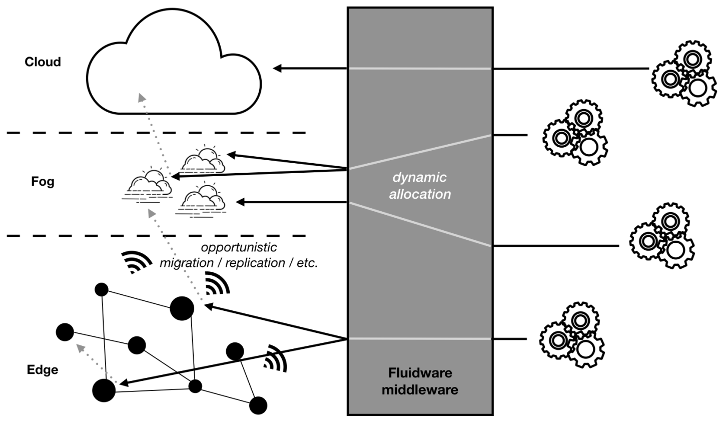

Fig. 3. Fluidware middleware deploys funnel processes opportunistically.

and actuators associated to the heating and A/C systems can set up the climate conditions in accord to the specific comfort levels of the inhabitants.

Concerning care delivery, one can think at services to monitor the health conditions of patients and their everyday activities, and possibly send notifications to the medical personnel to alert about unusual or potentially safety-critical situations. Furthermore, in presence of suitable actuators (e.g, robotic personal assistants or properly instrumented furniture such as electro-mechanical beds and sofas) the system can directly trigger the necessary actions to prevent damage (e.g.,, put the bed in vertical position upon sensing choking). All of the aforementioned IoT services are *local* in the sense that they do not require a global understanding of the situation.

In addition to the local services conceived to orchestrate specific portions of the facility, a number of *global* ones can be envisioned to regulate its overall functioning. For instance, services to control the heating and lightening systems, and in particular to monitor and keep under control energy expenses. Services to monitor fire and other dangerous situations, and automatically organise evacuation plans upon need. Services to control the overall distribution of people in the facility and steer assistance personnel accordingly.

3.2 IoT devices

In order to set up the aforementioned services, the following sensors and actuators are needed:

– general ambient sensors and actuators: light and heat controllers, gas and smoke detectors, presence and motion sensors, energy consumption sensors on electric appliances, shutter/curtain controllers, as well as daily life objects augmented with sensing capabilities (e.g, cup, fork, cane).

Table 1. Smart retirement centre services and related state of art solutions issues.

	Local	**Global**
Services	Occupant-centred illumination & climate conditions Safety-critical automated procedures Smart alerts & notifications	Energy efficiency Automated evacuation plan generation Staff routing and occupants steering
State of art	Smart Object / WoT resource / WS Synch. REST services Static 2/3-tiers architecture Broker-based publish-subscribe	Event-Condition-Action rules Service composition Cloud-hosted processing
Issues	SO abstraction level WoT full network stack Adding device/service	Simple message-based interaction Static network topology Design-time process allocation
Fluidware	Device independent Context dependent	Semantic pattern matching Dynamic network re-configuration Run-time process allocation

– specific sensors and actuators needed to monitor and assist with the health conditions peculiar to each guest, there including body sensors such as heart rate, glucose, blood pressure, temperature, or medical devices such as ECG monitors, sleep quality monitors, spyrometry devices, smart scales.

Besides devices inside guests' flats, all other places in the facility may include sensors to detect people presence and ambient conditions, actuators to regulate ambient conditions, and digital signals (e.g., audio system) or screens to provide notifications, messages, and alerts.

3.3 State of art solutions

At the modelling level, solutions readily available to deliver the aforementioned services are either based on the concept of *Smart Object* (SO) [11], on the *Web of Things* vision (WoT) [9], or on service-orientation. From the architectural standpoint, *service-oriented architectures* (SOA) [17] are most prominent, as the service abstraction can span the whole spectrum of Edge-Fog-Cloud layers typically featured in modern IoT deployments. Also, SOA is a good fit for both SO and WoT modelling. A the infrastructural level, the *2/3-tiers model* encompassing the whole network of devices from the Edge to the Cloud (possibly including Fog nodes) is the most popular right now, as it enables system administrators to deploy each service at the most appropriate location [12].

The above considerations imply that, for instance, realising services for monitoring patient clinical conditions and trigger appropriate action in case anomalies are detected would amount, essentially, at developing a web services stack for data processing and the appropriate communications with sensor and actuator devices. For instance, these may be connected to a local gateway (e.g., a RasPi or a desktop computer) for sending data, which is then processed by an ensemble of web services to realise the system functionalities. Cooperation between services may happen in a point-to-point fashion through simple RESTful endpoints, or by relying on a publish-subscribe broker. Finally, deployment would be mostly decided at design-time: the web service backends would be likely deployed on the Cloud, or possibly to a sufficient powerful Fog node, and devices connected to the gateway would have their own logic embedded to send data and listen for commands. This practice limits scalabilty as forces developers to individually deal with each specific device individually.

3.4 Issues

The solutions described above have a few limitations which current research proposals fail to address in a holistic and conceptually sound way. WoT modelling requires a full *network stack* to be available on connected devices, which is impractical for the resource-constrained ones usually employed at the Edge. SOA does not cope well with moving services across tiers' boundaries depending on available resources. The 2/3-tiered network topology model is usually accepted as a *static* one, where services are allocated to one specific tier at design time.

This means that, for instance, adding a new device to the system usually implies adding a new service to the stack, taking care of connecting the device to the rest of the application, re-wiring service composition routing paths, and such. Also, scaling the service to, for instance, a whole flat instead of a single room usually requires a re-deployment of the whole application.

3.5 Fluidware approach

Fluidware would enable developers to define the event streams they are interested in (e.g., the blood pressure or heart rate time series), the funnel processes they want to deploy (e.g., one triggering alerts if a combination of vital signs is worsening, and another one gathering vital signs measurements from body sensors) and how they combine the aforementioned streams, and let the Fluidware middleware taking care of *(a)* connecting the event streams to the actual source devices' data, and *(b)* allocating the funnel processes to the computational nodes composing the network. This means that developers need no longer to bother with re-wiring service connections if new applications are deployed, or to re-configure the network topology if new devices come in, nor to deal with device failures as long as other providing the same data are available.

Overall, this means that not only system designers have an *higher abstraction level* provided, but also a *higher degree of decoupling* between system hardware and software components. With Fluidware, indeed, "flows" decouple computational processes from the actual data sources they need, while "funnels" decouple developers from the burden of statically deciding where to deploy their services.

4 Smart University Campus

Another scenario suitable for Fluidware is a smart university campus, as it should provide comfortable living spaces for both students and faculty staff to support individual studies and research, as well as student teamwork and staff collaborative research. There, spaces could be re-organised on a per-need basis, such as in the case of planned temporary events (e.g., meetings, conferences, seminars, industry recruiting, etc.) or unexpected contingencies. For instance the earthquake of October 2016 seriously damaged the whole city centre of Camerino, causing damage to buildings of the Department of Computer Science, among the many. In presence of scarcity of space, organisation is thus even more critical, and opportunistic exploitation of available rooms is of paramount importance. Several services can then be conceived to deal with the shortage of spaces—as summarised in Table 2.

4.1 Services

A first service that can be delivered to students and faculty staff is the real-time *(a)* tracking of rooms occupancy and *(b)* adjustment of rooms' environmental conditions to the occupants' needs. At the *local* level, students, researchers, and professors may utilise a smartphone app indicating the place that better matches the users' requirements: for instance, some students require a study place which is isolated, quiet, warm, and bright, whereas others are just interested in finding the closest spot. At the *global* level, administration staff and campus managers may find convenient to have a Decision Support System for governance, aimed at analysing data generated from the university IoT infrastructure to better understand how to allocate the available spaces and to find the most suitable spaces for locating new services.

Another service could be a navigation system to steer people towards their destination, featuring *integrated contextual information* such as current or future crowded areas, planned events, etc. Different path optimisation methods (e.g., the shortest distance, the shortest time, accessible paths only, stairs minimisation, etc.) can be applied according to individual preferences and conditions. Another example is assisted parking: free parking slots (for cars or bikes) can be suggested according to planned activities of individuals; for instance, slots nearby the branch of the campus where activities will be located could be prioritised.

Lastly, an interesting service would be a recommendation system informing about opportunities of interaction and activities contextual to students' and researchers' goals and preferences. For example, in a multi-disciplinary campus, students may grow interest in seminars or events which belong to different degree courses; in these cases, it may be easy not to get appropriately notified. If the campus is big enough, it is also likely the opposite happens: students receive so many notifications about upcoming events that they simply ignore them all. In this case, the recommender system should be able to customise the information conveyed to each student, dynamically considering dominating interests, as well as maximising the interests of groups, as people are more likely to attend an event if an entire group is involved.

4.2 IoT devices

Several devices have to be scattered across the campus to support the aforementioned services: presence and motion sensors; temperature, humidity, light, and

Table 2. Smart university campus services and related state of art solutions issues.

	Local	**Global**
Services	Student profiling & recommendation Room re-purposing	Space management Service planning
State of art	Smart Object Process Modeling	Model based design Service composition Complex Event Processing
Issues	Modeling notation	Modeling notation for IoT Dynamic allocation
FluidWare	Device independent Context dependent Process-aware configuration	Event stream fusion Dynamic process & service configuration

noise sensors; RFID tags and readers; cameras; electric actuators; Edge devices such as single-board computers (e.g., Raspberry Pi). These devices will not form a *finite and static set*, *immutable* for the whole services lifetime: instead, it can dynamically change for device addition or failure and replacement. Moreover, it is worth emphasising that students and faculty staff, too, can serve as crowd-sensing sources sharing data through their own personal devices, for instance by simply accepting to communicate their position to the smart campus services.

4.3 State of art solutions

Several techniques are available for designing the aforementioned functionalities, however, none of them takes a holistic stance, which is instead quintessential for a system where decisions depend on a variety of factors. Also, existing solutions do normally require to send data through the Cloud to third parties, which can be undesirable. From the point of view of the developers, traditional approaches such as object orientation or web services quick fall short in terms of expressiveness, extendability, and integration in such highly and heterogeneous scenarios such as the IoT.

4.4 Issues

Using graphical notations for representing processes is a well-known practice in the areas of Model-driven engineering and Business Process Management (BPM). For instance, in our scenario we can represent processes for the search of a room, the information retrieval from sensors, the analysis of sensors information, the activation of sensors failure procedures, the engagement of users and their devices in the data retrieval procedure. However it is challenging to capture all the required details about sensors and smart devices with the actually available notations (e.g., BPMN). Especially, there is no standard way to represent an IoT device, and data are generally represented by a single element called data-objects. No support is given for the concept of flow/streaming of data which are necessary in many IoT deployments.

4.5 Fluidware approach

The objective of Fluidware funnel processes is to leverage on sensors opportunistically, so as to enable seamless provisioning of timely and reliable data for supporting the services offered to users. We may conceive a fluid process in two way: on one side, a *top-down approach* can be used by designing all the process variations that may be activated in some circumstances, whereas on the other side a *bottom-up approach* based on mining techniques can be used to discover relevant emerging processes. We could of course also think about a mixed approach. In any case, the activated processes are strongly dependent from sensors data, which means the processes are differentiated in terms of control-flow but also in terms of data-flow, which is something Fluidware can nicely support thanks to device decoupling and dynamic resource allocation.

Table 3. Stress-free workplace services and related state of art solutions issues.

	Local	**Global**
Services	Worker's stress monitoring & counteracting	Workplace liveability & safety management
State of art	Smart Object / WoT resource / WS Static 2/3-tiers architecture Gateway-based interoperability	Event-Condition-Action rules Service composition Edge-/Cloud-hosted processing
Issues	Scenario-dependent design Vendor lock-in and cost Mobility & context-awareness	Scalability Ad-hoc interoperability Implementation
Fluidware	Device independent programming QoS / SLA awareness	ECG streams aggregation Dynamic network re-configuration

As sensors and actuators can dynamically change, the *decoupling between devices and data flows* offered by Fluidware becomes of paramount importance. E.g., if each student has a dedicated application on his smartphone, hence is working as a crowdsensing source, as he/she moves around the campus the information that his/her smartphone provides changes context, hence may be *opportunistically* connected to another funnel process by the Fluidware middleware. Not only connection between event flows and funnel processes may change, but also allocation of such processes across the IoT network can: for instance, funnel process may migrate across Fog nodes (e.g., RasPis deployed in strategic positions across the Campus) with the goal of minimising latency in delivering services to students and researchers.

5 Stress-free Workplace

Overworking, conflicting relationships, psychological harassment, etc., contribute to workplaces stress, with negative impacts on workers' productivity and, especially, health. Cardiac monitoring is a key activity to detect high-stress conditions and the ECG monitoring has become one mainstream activity in ambient assisted living. A variety of services may be engineered to relieve workers and to help the governance to manage stressful collective situations (e.g., noisy office rooms, dangerously overcrowded canteen, etc.) while improving the livability and safety of the workplace.

5.1 Services

At a *local* level, a cardiac monitoring IoT service can acquire the real-time ECG of a single user from a worn chest band, process the received heart signals, and enable the individual HRV analysis. Eventually, doctors and psychologists can outline the stress profile of the monitored individual and take actions to preserve worker safety and well-being (e.g., notify a worker to take a break).

At the *global* level, instead, the IoT service can exploit *distributed acquisition* of ECG flows related to several monitored workers. In this case, a multi-user stress monitoring, along with other contextualised data, allows the workplace governance to identify both common uncomfortable situations (e.g., a noisy office room) and episodic dangerous ones (e.g., an overcrowded parking slot). Therefore, situations threatening the workplace livability and safety can be properly managed through immediate actuation (e.g., opening a window) or plan future actions (e.g., re-thinking distribution of parking spaces).

5.2 IoT devices

The envisioned stress monitoring service would hence need wearable, specific-purpose devices, such as Polar H10 chest band, for the heart signal acquisition and the Bluetooth-based communication to the user smartwatch/smartphone; portable, general-purpose devices, such as Android-based smartwatches Huawei Watch2, to be interfaced with the wearables; a single-board computer (e.g., Raspberry Pi3, Zotac CI540 NANO Pc, etc.) acting as a gateway for collecting and locally processing ECG data. These IoT devices are all commercially available and affordable. Besides these devices, the monitored workplace might include sensors to detect people presence and ambient conditions, actuators to regulate ambient conditions, and the like.

5.3 State of art solutions

Conventional ECG-based stress monitoring systems are typically designed for a single, specific (indoor or outdoor) setting provided with a set of (both fixed and mobile) physiological and wearable devices [10], considered as *Smart Objects* (SOs) [11] which can be accessed and controlled through the Web in the *Web of Things* vision (WoT) [9]. Cloud-based, Edge-based or hybrid solutions are chosen according to the specific requirements (real-time, costs, mobility, etc.) of a given scenario [15]. Private/public Cloud platforms typically support the system to store and process the data while lighter tasks are performed directly at the Edge. Gateways are often exploited to overcome the heterogeneity of communication protocols adopted by the different devices. Service composition typically happens in a *point-to-point* fashion through simple RESTful endpoints, with interactions *statically* defined by Event Condition Action rules [3].

5.4 Issues

The aforementioned solutions are typically tailored on a single, well-defined scenario (e.g., a highly instrumented infrastructure of an office room) and do not effectively handle the gradual transition between different environments. Device/worker mobility introduces the need for the seamless, dynamic execution of computational task according to changes in the devices' availability (e.g., a worker's smartphone might run out of battery and be automatically replaced by the smartwatch). Ad-hoc mechanisms are also typically provided to make heterogeneous devices interoperable (software bridges, data model translators, etc.), but these solutions often result in not scalable architectures, difficult to be extended. maintained, and upgraded in the future.

5.5 Fluidware approach

The Fluidware approach allows overcoming many of the aforementioned limitations. Being device independent, Fluidware does not assume the existence of specific types of devices in specific locations: this allows continuous and mobile monitoring of the workers, thus enabling the ECG-based stress monitoring system to *(a)* adaptively and reliably serve its purpose, despite contingencies like unreachable devices; *(b)* dynamically integrate new devices; *(c)* support semantic interoperability specifying the event/data flow in a declarative way. Indeed, Fluidware design and programming abstractions natively and seamlessly support Edge to Cloud computations, thus fostering the development of a scalable ECG-based stress monitoring system which adapts to the application requirements (ECG acquisition responsiveness, HRV calculation accuracy, etc.).

6 Conclusion

In this paper we discussed three case studies motivating the need for a paradigm shift in engineering and programming IoT services and applications. Such a paradigm shift is actually envisioned by the Fluidware approach already presented in [8], which we here conceptually evaluate against three different case studies with the aim of emphasising how it allows overcoming issues in current state of art solutions.

References

1. Alrahman, Y.A., De Nicola, R., Garbi, G., Loreti, M.: A distributed coordination infrastructure for attribute-based interaction. In: Baier, C., Caires, L. (eds.) Formal Techniques for Distributed Objects, Components, and Systems. pp. 1–20. Springer International Publishing, Cham (2018)
2. Alrahman, Y.A., De Nicola, R., Loreti, M., Tiezzi, F., Vigo, R.: A calculus for attribute-based communication. In: Proceedings of the 30th Annual ACM Symposium on Applied Computing. pp. 1840–1845. SAC '15, ACM, New York, NY, USA (2015)
3. Andreoli, A., Gravina, R., Giannantonio, R., Pierleoni, P., Fortino, G.: Spine-hrv: A bsn-based toolkit for heart rate variability analysis in the time-domain. In: Wearable and autonomous biomedical devices and systems for smart environment, pp. 369–389. Springer (2010)
4. Andrikopoulos, V., Bucchiarone, A., Gómez Sáez, S., Karastoyanova, D., Mezzina, C.A.: Towards modeling and execution of collective adaptive systems. In: Lomuscio, A.R., Nepal, S., Patrizi, F., Benatallah, B., Brandić, I. (eds.) Service-Oriented Computing – ICSOC 2013 Workshops. pp. 69–81. Springer International Publishing, Cham (2014)

5. Audrito, G., Viroli, M., Damiani, F., Pianini, D., Beal, J.: A higher-order calculus of computational fields. ACM Trans. Comput. Logic **20**(1), 5:1–5:55 (Jan 2019)
6. Beal, J., Pianini, D., Viroli, M.: Aggregate programming for the internet of things. Computer **48**(9), 22–30 (Sep 2015). https://doi.org/10.1109/MC.2015.261
7. De Nicola, R., Latella, D., Lafuente, A.L., Loreti, M., Margheri, A., Massink, M., Morichetta, A., Pugliese, R., Tiezzi, F., Vandin, A.: The SCEL Language: Design, Implementation, Verification, pp. 3–71. Springer International Publishing, Cham. https://doi.org/10.1007/978-3-319-16310-9_1
8. Fortino, G., Re, B., Viroli, M., Zambonelli, F.: Fluidware: An Approach Towards Adaptive and Scalable Programming of the IoT, pp. 411–427. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-21485-2_22
9. Guinard, D., Trifa, V.: Towards the web of things: Web mashups for embedded devices. In: Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences), Madrid, Spain. vol. 15 (2009)
10. Jovanov, E., Lords, A., Raskovic, D., Cox, P.G., Adhami, R., Andrasik, F.: Stress monitoring using a distributed wireless intelligent sensor system. IEEE Engineering in Medicine and Biology Magazine **22**(3), 49–55 (2003)
11. Kortuem, G., Kawsar, F., Sundramoorthy, V., Fitton, D.: Smart objects as building blocks for the internet of things. IEEE Internet Computing **14**(1), 44–51 (December 2009). https://doi.org/10.1109/MIC.2009.143, http://usir.salford.ac.uk/id/eprint/2735/
12. Li, W., Santos, I., Delicato, F.C., Pires, P.F., Pirmez, L., Wei, W., Song, H., Zomaya, A., Khan, S.: System modelling and performance evaluation of a three-tier cloud of things. Future Generation Computer Systems **70**, 104 – 125 (2017). https://doi.org/https://doi.org/10.1016/j.future.2016.06.019, http://www.sciencedirect.com/science/article/pii/S0167739X16302047
13. Mamei, M., Zambonelli, F.: Programming pervasive and mobile computing applications: The tota approach. ACM Trans. Softw. Eng. Methodol. **18**(4), 15:1–15:56 (Jul 2009)
14. Namiot, D., Sneps-Sneppe, M.: On iot programming. International Journal of Open Information Technologies **2**(10), 25–28 (2014)
15. Pace, P., Aloi, G., Gravina, R., Caliciuri, G., Fortino, G., Liotta, A.: An edge-based architecture to support efficient applications for healthcare industry 4.0. IEEE Trans. on Industrial Informatics **15**(1), 481–489 (2018)
16. Palade, A., Cabrera, C., White, G., Razzaque, M.A., Clarke, S.: 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). pp. 1–6 (June 2017)
17. Perrey, R., Lycett, M.: Service-oriented architecture. In: 2003 Symposium on Applications and the Internet Workshops, 2003. Proceedings. pp. 116–119 (Jan 2003). https://doi.org/10.1109/SAINTW.2003.1210138
18. Pournaras, E.: Overlay service computing - modular and reconfigurable collective adaptive systems. Scalable Computing: Practice and Experience **16**(3), 249–270 (2015)
19. Rausch, T., Dustdar, S., Ranjan, R.: Osmotic message-oriented middleware for the internet of things. IEEE Cloud Computing **5**(2), 17–25 (Mar 2018)
20. Viroli, M., Audrito, G., Beal, J., Damiani, F., Pianini, D.: Engineering resilient collective adaptive systems by self-stabilisation. ACM Trans. Model. Comput. Simul. **28**(2), 16:1–16:28 (Mar 2018)

21. Viroli, M., Casadei, M., Montagna, S., Zambonelli, F.: Spatial coordination of pervasive services through chemical-inspired tuple spaces. ACM Trans. Auton. Adapt. Syst. **6**(2), 14:1–14:24 (Jun 2011)
22. Zambonelli, F.: Key abstractions for iot-oriented software engineering. IEEE Software **34**(1), 38–45 (Jan 2017). https://doi.org/10.1109/MS.2017.3