Middleware Architectures for Fluid Computing

Chapter 3
Middleware Architectures for Fluid Computing

Roberto Casadei, Nicolas Farabegoli, Giancarlo Fortino, Claudio Savaglio, and Mirko Viroli

Abstract This chapter describes the Fluidware architecture with emphasis on its two main features, namely a truly full-distributed computing approach and a middleware component devoted to the actual creation, allocation and management of funnels to computational nodes and to their deployment across the IoT-Edge-Cloud continuum. Particularly, the pulverization paradigm will be presented, as an instantiation of the Fluidware middleware aimed at concretely spread an IoT services within an highly distributed applications.

Key words: Architecture, Edge, Cloud, Middleware, Continuum, Pulverization.

3.1 Introduction

Device, Edge/Fog and Cloud is the typical hierarchical architecture featuring the majority of current IoT systems, where tasks are statically and a-priori situated per-layer (e.g., mere data forwarding or simple aggregation at IoT layer, more complex data processing towards the Cloud) regardless of the actual resource availability of the devices. However, such a siloed tasks assignment collides with the opportunistic

Roberto Casadei
Università di Bologna, Cesena (FC), Italy e-mail: roby.casadei@unibo.it

Nicola Farebegoli
Università di Bologna, Cesena (FC), Italy e-mail: nicolas.farabegoli@unibo.it

Giancarlo Fortino
Università della Calabria, Rende (CS), Italy e-mail: giancarlo.fortino@unical.it

Claudio Savaglio
Università della Calabria, Rende (CS), Italy e-mail: csavaglio@dimes.unical.it

Mirko Viroli
Università di Bologna, Cesena (FC), Italy e-mail: mirko.viroli@unibo.it

essence of the IoT domain (where the QoS requirements should dynamically drive the identification of a target device for a given task) and with the rapid advancements in miniaturization and chips construction, which enabled even small and/or mobile IoT devices like smartphone or Raspberry for example, to potentially behave as an Edge Server, being equipped with performing hardware like multi-processors and GPU. Moreover, the distribution of the computing in such a strongly layered architecture is proved as effective exclusively intra-layer, thus limiting the development of IoT services whose operations should span simultaneously inter-layer. Conversely, the fundamental concept underlying Fluidware is to operate within a continuously changing infrastructure that encompasses a spectrum of IoT fabric (i.e., IoT, Edge and Cloud devices) and resources: this allows for swift adaptation to the internal dynamics of the environment and enables the coordinated execution of distributed activities across devices possessing different levels of computational capability. Therefore, the Fluidware abstractions introduced in the previous chapters (Funnel, Field, Fabric, etc.) need to be grounded on a solid architecture and specific components specifically outlined to mirror and to support the features as well as the desiderata of our project. To this end, in this Chapter first we briefly analyze main limitations of typical centralized architectures in Sec. 3.2 which led to the "IoT-Edge-Cloud" continuum, and then we introduce our solution in Sec. 3.3, followed by an in-depth about its key element, namely the Fluidware middleware in Sec. 3.4. A particular instantiation of the Fluidware middleware pivoted around the "Pulverization" computing paradigm in presented in Sec. 3.5 while final remarks conclude the chapter.

3.2 Motivations: From fully-centralized to fully-distributed "fluid" architectures

At the state-of-the-art, most of IoT services still mirror conventional web services (being based on the Web, they are often referred as Web-of-Things services [2]) and expose a two-level, service-oriented architectures (SOA)[1]: in such a setting, service provision is implemented in a centralized way, by having all data from sensors be 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 issued. Typical examples are the Smart Home Assistants: they rely on cloud-based IoT platforms [3] where thermostats, lights, security systems, and appliances forward data for being processed, analyzed, and used to trigger actions or provide insights to the users. In general, in these IoT services based on **Cloud Computing** are easily recognizable a:

- Device level: also identified as "perception level", it encompasses all the devices (more or less "smart", i.r., RFDI tags and readers, environmental sensors, wearable devices, video-cameras etc.) able to directly interact with physical phenomena (e.g., temperature or humidity) or real-world events (e.g., window opened, button pressed) and, hence, to communicate with the Cloud to offload

their data, raw or simply refined (e.g., aggregated temperature values on a hourly window, number of people passing their badge);
- Cloud level: it aims to gather all the available data (live, historical, etc.) in huge repositories hosted by powerful remote servers for feeding resource-demanding elaboration processes, and possibly, sending actuation commands back to the Device level (e.g., thermal comfort).

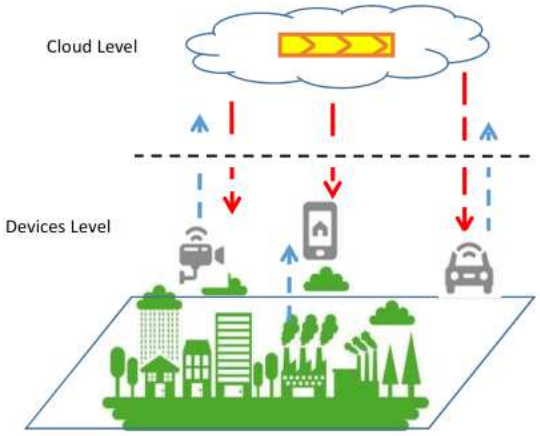


Fig. 3.1: Two conventional two-level architecture based on sense-and-forward approach

Being adopted by many Cloud IoT platforms such as Microsoft Azure, Amazon AWS, etc., this simple "sense-and-forward" approach originally raised (early 2000s) to address the resources limitation typical of Wireless Sensors and Actuators Networks (WSANs) [4]: these were the only resources of the Device level, strictly limited in terms of storage/processing capabilities and, hence, forced to periodically offload (almost raw) data. Moreover, the confluence of sensors data into an unique point helped in the development of high-performance AI models, which were trained over huge data set. However, over the years, the exploding number of heterogeneous IoT devices (referred as IoT fabric), (Big) data and services did such a Cloud-dependant approach collide with their inherent real-time nature and their non negligible requirements in terms of privacy, reliability, adaptivity and scalability. For example, in the e-Health and Ambient Assisted Living domains [11] sensitive patients' data need to be analyzed "in-situ" and immediately flow by the proper actuator if we aim to prevent or assess potentially dangerous health conditions or simply dangerous behaviors and, hence, to take immediate actions. In such a context (or in similar ones, like Smart Transportation or Virtual Reality), the natural non-determinism as well as vulnerability of the network made the Cloud, even the most powerful, as a bottleneck for the provision of many key IoT services. Therefore, in the last two

decades, distributed architectures according to the **Fog Computing** (from 2012) and **Edge Computing** (from 2016) have arisen with the intent of decentralizing and bringing the processing of IoT data closer to the event sources [5]. In particular, a well-defined set of activities characterizes each level of their architectures so that system administrators deploy each service at the most appropriate location directly at design time:

- Device/Edge level: at the front-end, IoT devices and Edge nodes still perform sensory and actuation tasks but also additional, lightweight activities like data compression, filtering, caching, and human-interfacing (e.g., through smartphones) or local communication;
- Fog level: at the near-end, Fog nodes are purposely deputed at local ML inference tasks, orchestration and coordination of services and resources, provided with relevant resources and stable Internet connectivity;
- Cloud level: at the far-end, cloud servers are still involved in massive parallel data processing, knowledge discovery, persistent storage, ML models training, etc. as well as in data exploration (e.g., dashboarding).

For example, with respect to the aforementioned AAL scenario, it would be necessary a web services stack for data processing and the appropriate communications with sensor and actuator devices 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. In such direction, it happens that many Cloud IoT platforms have started embedding models, technologies and protocols specifically devoted to connect the Cloud to the Edge and Fog levels, where it emerges servitization paradigms different from SOA: microservices, for example, can be developed, deployed, and scaled independently thus improving modularity, flexibility, resilience and interoperability (but at cost of additional complexity in terms of service orchestration, deployment, and monitoring).

While both Fog and Edge paradigms address the challenges of distributing and outsourcing the computing from the cloud while preserving real-timeness, bandwidth consumption, and data privacy, their architectural designs and deployment models differ mainly for *(i)* the proximity to the data source, since ”fog nodes” are intermediate yet powerful devices located between edge devices and the cloud servers; and for *(ii)* the network connectivity requirements, since ”edge nodes” can operate in environments with intermittent or limited network connectivity, being able to function autonomously and make decisions locally with negligible data transfer. Moreover, with the great advancements and consequent diffusion of single-board computers (SBCs, e.g., Google Coral Board, Nvidia Jetson Nano, RaspberryPi, Intel NeuroStick) and flash-memories, most of the AI/ML workload and storage tasks ini-

tially aimed at fog nodes have efficiently carried out by edge nodes directly at the real data/event source, reason why the Fog computing has rapidly lost attractiveness to the advantage of Edge Computing. Finally, more recently (2020), the idea of an architecture with clearly marked and compartmentalized levels has been replace with the seamless **"IoT-Edge-Cloud" continuum** [6]. This can be visualized as a gradient, where all the aforementioned computing paradigms are integrated and, potentially, interchangeably into an opportunistic manner, accordingly to the specific needs of an IoT services and the actual availability or resources. In such an approach, there are no a-priori defined responsibilities or deployment targets but these are case-by-case established by means of purposed middleware components operating according to specific policies. This provides great flexibility and exactly matches the recent AI trends, heading towards a dynamic distribution of traditional models and techniques (at different degree of accuracy) across a spectrum of heterogeneous computing resources, towards an actionable intelligence [1], i.e., the so-called Edge Intelligence [7].

3.3 The Fluidware architecture

To overcome the aforementioned limitations of a compartmentalized two/three layered architecture and by adhering to the continuum principles, we outline a fully distributed, super-peer architecture where Fluidware services are supported by IoT devices, edge nodes and/or cloud platforms which are dynamically activated and re-configured.

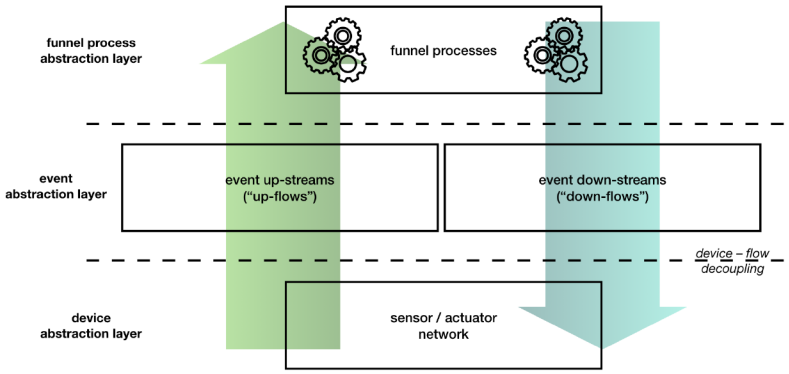


Fig. 3.2: Three layer architecture with different abstraction degrees

Consequently, instead of conventional layered architecture based upon a static tasks assignment, we outline an organization based on abstraction levels so that developers willing to design an IoT service or application in terms funnels and event streams detached from the physical devices actually producing and consuming them:

- device abstraction level: encompassing sensor and actuator networks;
- event abstraction level: encompassing streams of data flowing in both directions, i.e., ”up-flows” and ”down-flows”;
- funnel process abstraction level: encompassing funnel processes which digest contextualized streams of events.

Indeed, to promote scalability and flexibility, Fluidware will support three kinds of interactions and coordination (as from Figure3.2):

- 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;
- intra-edge and inter-edge interactions, 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-based interactions, for centralized monitoring, coordination, and storage.

The outlined architecture based on the continuum concept over-performs conventional 2/3 layered architectures for many reasons: WoT 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, thus 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 requires in most cases a re-deployment of the whole application.

The execution of funnel processes along the entire IoT/edge/cloud stack is enabled by the key component of the Fluidware architecture, the middleware, which fully provide utility-driven exploitation of infrastructure and it is described in the following Section.

3.4 The Fluidware Middleware

Middlewares, widely used in conventional distributed systems, provide general and specific abstractions (e.g. object computation model, inter-object communication, hardware and software interfaces, data types, discovery service, knowledge management, physical location) through which IoT devices and systems and their related services can be easily and securely built up and integrated. Therefore, middlewares

play a key role for the design and implementation of the whole [14],[16] acting as a bridge between the IoT devices and the services that interact with them.

As anticipated in Chapter 2, funnel processes will be specified in a declarative way independently of their actual allocation and distribution, being managed by the Fluidware middleware. At the implementation and deployment level, the 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 [13], or even at the level of some centralized cloud. 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.

In more detail, we expect the Fluidware middleware, possibly guided by some "allocation" directive, to automatically handle the allocation and possible replication of Funnel processes, their optimised execution and transparent replication/relocation across the network [15]. Specifically, the activities to develop the Fluidware middleware platform will address five related objectives:

1. at run time, to realise the funnel processes specification, by connecting the event streams to the actual source devices' data and establishing 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 publishsubscribe blackboard, or with a tuple space);
2. define a mapping from the funnels processes specifications into a set of distributed components to be deployed atop the Fluidware platform—such mapping will be based on a model-driven development and will distinguish abstract platform-independent specifications from deployable platform-dependent components;
3. to deploy such a funnel process over the platform-dependent components that can be dynamically activated and re-configured in IoT devices, edge servers and/or cloud platforms as individual process, or as part of a swarm of distributed replicas, or as a worker in a map-reduce like architecture;
4. to dynamically manage the deployed funnels by instantiating local proxies of processes and to relocate them as needed;
5. to analyze interoperability and security issues, by defining guidelines for enabling devices to connect to the Fluidware platform, and analyzing how a trust-oriented distributed infrastructure for inter-component security can be integrated within it.

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 subprocesses: one gathering temperatures from sensors, thus directly deployed onboard 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.

3.4.1 Towards the Fluidware Middleware implementation

The outlined abstract components of the Fluidware Middleware need to be grounded on a solid architecture and a flexible computing paradigm: indeed, the achievement of the aforementioned objectives demands for applications deployable across the ICT infrastructure, thus allowing multiple potential deployment configurations for the same application logic with potentially different performance and costs. Therefore, in the following we introduce the ”pulverization” paradigm that inherently supports essential features of autonomy, decentralization and adaptiveness and that allows exploring and assessing the set of potential deployment configurations.

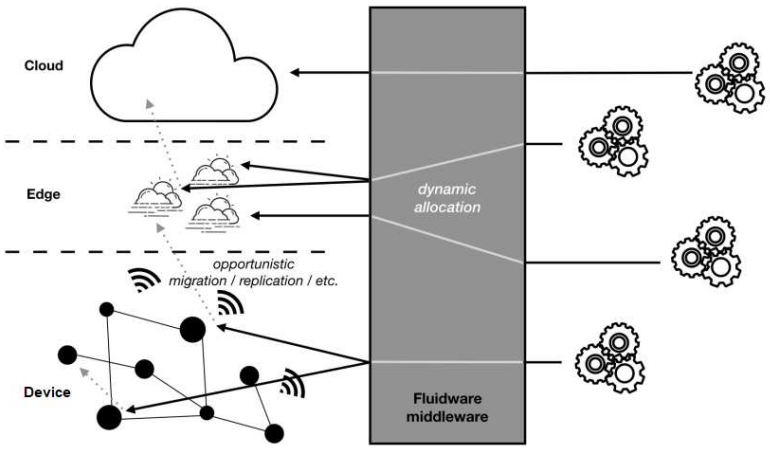


Fig. 3.3: Fluidware middleware deploys funnel processes opportunistically

3.5 The *Pulverization* Approach to Distributed Application Partitioning and Deployment

The target infrastructures of IoT applications may vary depending on the application requirements and the available resources. Regarding the infrastructure, it is mounted the idea of edge-cloud continuum, where a multi-layer heterogeneous network of devices ranging from cloud servers to small embedded devices, represents a valuable solution for the deployment of IoT applications. Such applications can benefit from this kind of infrastructure, however, the effective exploitation of the edge-cloud continuum poses new challenges to application designers. Commonly, the design of an application is made having in mind a specific infrastructure, thus, making it difficult to exploit the full potential of dynamic and heterogeneous infrastructures.

3.5.1 From monolith to component-based software systems

The traditional approach to software system design relies on the modelling of the system as a monolith. In this approach, the different software components and functionalities coexist in the same codebase, and typically the software is distributed and operated as a single deployment unit. Although this approach has evident advantages like *simple design*, *performance* and *security*, and *simple operation*, the monolith nature of this kind of software makes it difficult to evolve the software, resulting in a lower flexibility.

With the emergence of the Cloud, this kind of architecture exhibited several limitations. One of the main advantages of the Cloud is its scalability and flexibility depending on the changing requirements; these characteristics cannot be fully exploited with monolithic applications. The complexity is represented by the fact that if scaling is required, the replication of the entire software system is required, when it is likely that only one part of the system should be subject to replication, resulting in a waste of resource with less effective results.

To better exploit the full potential of Cloud services, recent software system are designed to be highly *modular*, *extensible* and *flexible*. With this approach, the software system is broken down into several independent software components that communicate each other via specific API. In this way, each component can be updated, modified or replicated without affecting the rest of the system. Moreover, this modular nature of software systems promotes dynamic reconfiguration according to the changing requirements of the system such as cost, bandwidth, storage, etc. Additionally, with the subdivision in components, we can demand certain responsibilities to other components, distributing and organising the complexity in a system.

3.5.2 The *Pulverization* approach

In the context of CAS (Collective Adaptive System), an effective way of decomposing an application in order to achieve flexible deployment is the *Pulverization* approach [17]. This approach fosters the decoupling of the application logic from deployment aspects, as it is explained in the following. Our target systems consist of collections of devices of two kinds:

1. *application-level devices*: these are the entities that we would like to program or control to provide application-level services;
2. *infrastructure-level devices*: these are computers that are *not* part of the application itself but rather support its *operational needs*.

In the first place, the pulverization approach distinguishes a *logical system* from the *software system* that implements it and the *physical system* that supports the execution of the implementation. By a logical perspective, a system consists of a network of *devices*, equipped with *sensors* and *actuators*, that can interact by exchanging messages with a dynamic subset of other devices known as a device's *neighbourhood*. Each device has a *behaviour* organised into *rounds*, with each round logically consisting into *sense-compute-interact* steps.

A simple deployment and operational scheme for this logical system could consist in deploying one software component per logical device into the corresponding application-level physical device. However, this has various drawbacks: (i) resource-constrained devices may not be able to execute the software components; (ii) the exploitation of the infrastructural elements is limited; (iii) the deployment solution space is very small, and hence the possibility for optimisations.

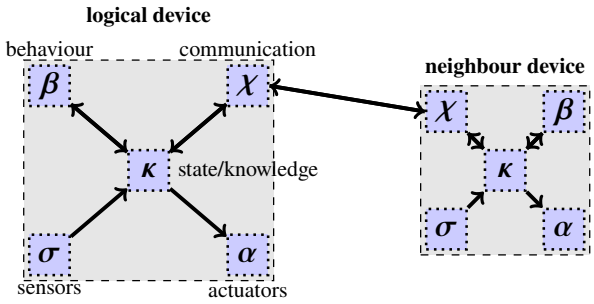
Then, the idea of the pulverization approach consists in designing the activity of each logical device in terms of an ensemble of *five components*:

1. *behaviour*
2. *state*
3. *communication*
4. *sensors*
5. *actuators*

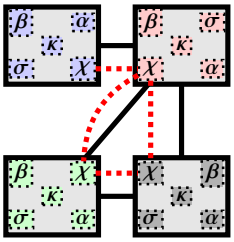
With this design, each component can be implemented as a *software component*, also working as a *deployment unit*. All five components can be deployed independently on the available infrastructure without affecting the application logic. In this way, the designer can focus on the business logic of the application, delaying the decision about the deployment infrastructure to the deployment phase.

Figure 3.4 exemplifies the pulverisation model and examples of deployments.

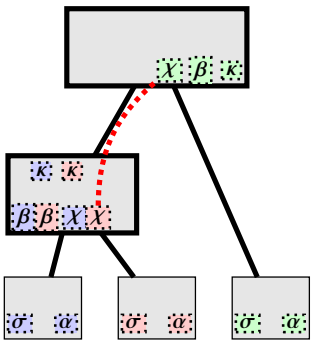
Each device, in the pulverization model, performs a well-defined cycle composed of the following steps: *context acquisition* where information about the context (e.g. sensors' state, neighbours' messages) are collected, *computation* where the behaviour function is applied against the context information, *coordination* where from the state, coordination messages are sent to neighbours devices, and finally, *actuation* where



(a) A logical device, split into sub-components, and one of its neighbours.



(b) Peer-to-peer architecture: one-to-one mapping between logical and physical devices, with no offloading.



(c) IoT hosts can be thin, with some components offloaded at the edge and cloud.

Fig. 3.4: Pulverisation model and examples of deployments (Figure adapted from [**?**]). Notation: solid-border boxes denote physical hosts (bold borders are for thick devices); solid lines denote connections between hosts; dashed-border boxes denote software components; different colours denote (software components of) different logical devices; red dashed lines denote connections between the software components, i.e., neighbouring relationships (not shown for co-located software components).

the actuators are activated according to the device state. The continuous execution of this loop determine the overall behavior of the device.

Once the application has been "pulverised", a mapping must be provided between the pulverised logical system and the hosts that will execute the pulverised components. In this vision, a single logical device, usually ends up being executed on multiple hosts.

3.5.2.1 Target infrastructure and applications

The pulverization approach tries to tackle the complex problem of deploying CASs in the context of CPS. In these contexts, a huge number of device should coordinate each other to achieve a common goal and sensing and acting operation are require to interact with the physical environment. Moreover, due to the high dynamic nature of CAS, modern infrastructure such as the Cloud-Edge continuum can provide an effective solution for the deployment of these kind of systems. The intrinsic complexity of the managing of this type of infrastructure makes the Pulverization approach and effective solution for hiding it and letting to the platform/middleware its management.

Relevant scenarios in which the Pulverization approach can be applied are *IoT Systems*, *Smart Cities*, *Wearable Computing Systems*, or more generally, in all scenarios characterized by an high level of heterogeneity in terms of computational resources, devices, etc. and cooperation and coordination between the devices are required.

3.5.3 *Pulverization* middleware

The *Pulverization* approach requires the support of a middleware for executing one or more applications. In this section we will provide a possible architecture of a middleware supporting the Pulverization approach, the main components and responsibilities, as well as the expected outcomes of the middleware.

A possible architecture can be composed of the following components: *communication manager*, *deployment unit control manager* and *reconfiguration manager*. Based on these three components, the middleware can manage different aspects, spanning from communication, to component management.

Communication manager

This component is responsible for manage all the communications with the other middleware instance distributed among the infrastructure of the system. In particular, this component is responsible for determining the physical network topology upon which the communications of the system are implemented. Ideally, this component should manage network-related operation like component to component communication, disconnection, reshape of the net in terms of reachable devices, etc. This component manages two types of communication: *service communication* and *application communication*. The former are communication needed by the middleware to build the net graph and react to network changes and events. The latter are the communication at application level, specifically the messages that the components needs to exchange to implement the Pulverization device cycle.

Deployment unit control manager

According to the Pulverization model, each component belonging to a logical device can be independently deployed over the hosts that constitute the infrastructure. This component of the middleware is responsible for taking into account the execution (the start and stop) of a component instance, according to the deployment plan generated by the **Reconfiguration manager**, managing the so called "deployment unit": a set of components that belong to a specific instance of a logical device. To support the execution of multiple systems at the same time in the same infrastructure, this component can manage multiple deployment unit belonging to logical devices of different application. In this way, we can efficiently exploit the underlying infrastructure by maximising the number of application that can coexist on a single infrastructure.

Reconfiguration manager

To exploit the full potential of the underlying infrastructure, we need a mechanism to dynamically adapt the deployment strategy according to the changing requirements or condition of the system. This component represents the way we can achieve the dynamic reconfiguration of the system, relying, ideally, on declarative reconfiguration rule expressing the needs of maintaining certain QoS or condition of the system. Based on the information produced by the **Communicator manager** about the current state of the network and the infrastructure, and the specific condition that should be maintained, this component try to generate the best component allocation. In this vision, due to the high complexity for the managing of dynamic conditions, AI can be injected to support the decision of the best deployment strategy to adopt at run-time.

With this architecture, once the application is "pulverized", the middleware take into account the execution of the component via the **Deployment unit control manager**, the management of the underlying infrastructure and the communications are demanded to the **Communication manager**, and finally, the **Reconfiguration manager** adapt the system to maintain specific QoS of the system providing real-time reconfiguration plan.

In figure 3.5 is depicted the described architecture and the relationship between the middleware components.

3.5.4 Fluidware and Pulverization middleware

This section motivates how the Pulverization middleware can be an instance of the Fluidware middleware model for CAS and CPS scenarios.

Section 3.4 exemplifies the principal traits for the Fluidware middleware, showing the five principal distinguish elements of it. The aforementioned characteristics refer

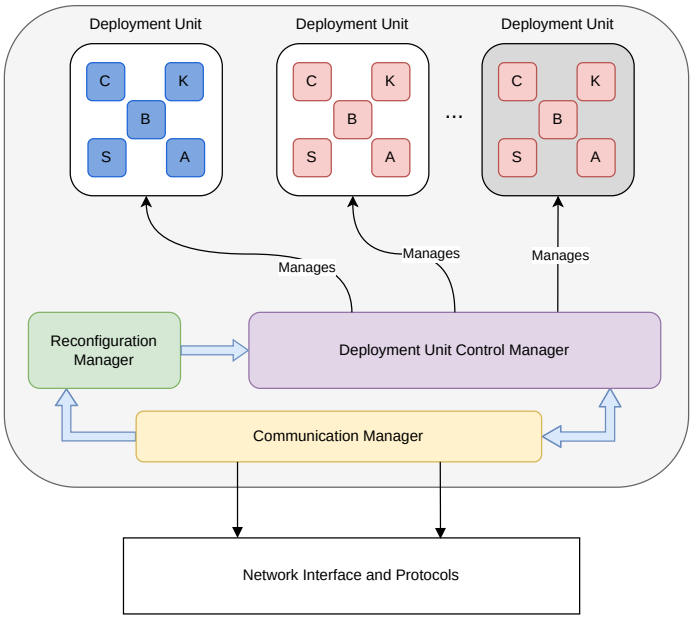


Fig. 3.5: Overall architecture of the pulverization middleware. The colored boxes on the bottom of the figure represents the main components of the middleware and the light blue lines denote the relationship between components. Different colors of the components belonging to each "deployment unit" denote a different instance of a logical device. The different background color of each "deployment unit" denote that device belongs to a different application. The box named *Network Interface and Protocols* represents the platform-specific interface an protocols made available by the host on which the middleware is executed on.

to a generic model for an effective Fluidware middleware pointing common characteristics. Based on these characteristics, the Pulverization middleware represents a possible instance of the Fluidware middleware model where the main focus is the deployment of AC applications.

The Pulverization middleware satisfies the five objectives as follows. The funnel process specification is defined by the middleware leveraging a message-stream communication pattern, which allows the connection of the stream of data produced by the devices to the rest of the infrastructure demanded to elaborate and process the stream; then, the elaborated stream is used to control the actuators of the system.

The mapping from funnel processes and a set of distributed components, pointed out in 2., is inherently captured by the pulverization model and consequently by the middleware. Moreover, the provided mapping can evolve in time based on dynamic conditions and requirements by leveraging a reconfiguration strategy supported i.e. by AI techniques. As a consequence, this dynamic and opportunistic system recon-

figuration, allowing fluid deployment, addresses the points 3. and 4. of the main objectives of the Fluidware middleware model.

Since each middleware instance can manage multiple components belonging to multiple device in the system, privacy and security concerns can be addressed by defining appropriate *reconfiguration rules* to enforce the appropriate placement of components execution. In this way, also the objective expressed by the 5. is addressed by the pulverization middleware.

3.6 Conclusion

Concluding, the Fluidware architecture is mainly featured by its fully distributed and super-peer approach and the pulverization middleware represents an effective implementation for the deployment of AC applications. Indeed, the high flexibility of the Pulverization model joined with the architecture of the middleware make the funnels definition straightforward and dynamically manageable. More details about a methodology for specifying how Fluidware funnels can be decomposed through the pulverization approach a toolchain for supporting the formal description and final deployment are reported in Chapter 8.

References

1. Guinard, D., Trifa, V., Karnouskos, S., Spiess, P. & Savio, D. Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services. *IEEE Transactions On Services Computing*. **3**, 223-235 (2010)
2. Tzavaras, A. & Petrakis, E. Web of Things Functionality in IoT: A Service Oriented Perspective. *2021 12th International Conference On Information, Intelligence, Systems & Applications (IISA)*. pp. 1-8 (2021)
3. Alam, T. Cloud-based IoT applications and their roles in smart cities. *Smart Cities*. **4**, 1196-1219 (2021)
4. Sharma, R., Prakash, S. & Roy, P. Methodology, applications, and challenges of WSN-IoT. *2020 International Conference On Electrical And Electronics Engineering (ICE3)*. pp. 502-507 (2020)
5. Laroui, M., Nour, B., Moungla, H., Cherif, M., Afifi, H. & Guizani, M. Edge and fog computing for IoT: A survey on current research activities & future directions. *Computer Communications*. **180** pp. 210-231 (2021)
6. Taheri, J., Dustdar, S., Zomaya, A. & Deng, S. Distributed Computing Continuum Systems. *Edge Intelligence: From Theory To Practice*. pp. 1-30 (2023)
7. Barbuto, V., Savaglio, C., Chen, M. & Fortino, G. Disclosing Edge Intelligence: A Systematic Meta-survey. *Big Data And Cognitive Computing*. **7**, 44 (2023)
8. Casadei, R., Viroli, M., Audrito, G., Pianini, D. & Damiani, F. Engineering collective intelligence at the edge with aggregate processes. *Engineering Applications Of Artificial Intelligence*. **97** pp. 104081 (2021)
9. Mamei, M. & Zambonelli, F. Field-based coordination for pervasive multiagent systems. (Springer Science & Business Media,2006)
10. Murturi, I. & Dustdar, S. Decent: A decentralized configurator for controlling elasticity in dynamic edge networks. *ACM Transactions On Internet Technology (TOIT)*. **22**, 1-21 (2022)

11. Aloi, G., Fortino, G., Gravina, R., Pace, P. & Savaglio, C. Simulation-driven platform for Edge-based AAL systems. *IEEE Journal On Selected Areas In Communications*. **39**, 446-462 (2020)
12. Savaglio, C. & Fortino, G. A simulation-driven methodology for IoT data mining based on edge computing. *ACM Transactions On Internet Technology (TOIT)*. **21**, 1-22 (2021)
13. Ansari, N. & Sun, X. Mobile edge computing empowers internet of things. *IEICE Transactions On Communications*. **101**, 604-619 (2018)
14. Palade, A., Cabrera, C., White, G., Razzaque, M. & Clarke, S. Middleware for Internet of Things: A quantitative evaluation in small scale. *2017 IEEE 18th International Symposium On A World Of Wireless, Mobile And Multimedia Networks (WoWMoM)*. pp. 1-6 (2017)
15. Villari, M., Fazio, M., Dustdar, S., Rana, O., Jha, D. & Ranjan, R. Osmosis: The osmotic computing platform for microelements in the cloud, edge, and internet of things. *Computer*. **52**, 14-26 (2019)
16. Fortino, G., Guerrieri, A., Russo, W. & Savaglio, C. Middlewares for smart objects and smart environments: overview and comparison. *Internet Of Things Based On Smart Objects: Technology, Middleware And Applications*. pp. 1-27 (2014)
17. Casadei, R. & Fortino, G. & Pianini, D. & Placuzzi, A. & Savaglio, C. & Viroli, Mirko A Methodology and Simulation-Based Toolchain for Estimating Deployment Performance of Smart Collective Services at the Edge *IEEE Internet of Things Journal*. pp. 20136-20148 (2022)