frontiers | Frontiers in The Internet of Things

TYPE Methods
PUBLISHED 28 September 2023
DOI 10.3389/friot.2023.1242101

OPEN ACCESS

EDITED BY
Sye Loong Keoh,
University of Glasgow, United Kingdom

REVIEWED BY
Antonino Galletta,
University of Messina, Italy
Tomasz Szydlo,
AGH University of Science and Technology, Poland

*CORRESPONDENCE
Edgar Ramos,
edgar.ramos@ericsson.com
Senthamiz Selvi Arumugam,
senthamiz.selvi.a@ericsson.com

RECEIVED 18 June 2023
ACCEPTED 07 September 2023
PUBLISHED 28 September 2023

CITATION
Ramos E and Arumugam SS (2023), Process automation instantiation for intelligence orchestration.
Front. Internet. Things 2:1242101.
doi: 10.3389/friot.2023.1242101

Process automation instantiation for intelligence orchestration

Edgar Ramos[1]* and Senthamiz Selvi Arumugam[2]*

[1]Ericsson Research, Kirkkonummi, Finland, [2]Ericsson Research, Kista, Sweden

Devices and use case heterogeneity in Internet of Things (IoT) make it challenging to streamline the creation of processes that orchestrate devices that work interactively through actuation and sensing (data generation and data processing) toward fulfilling a goal. This raises the question of what is needed to realize this vision using serverless technologies and leveraging semantic description tools that would eventually complement standards to describe such orchestration in a similar way to microservices (e.g., using TOSCA). This study analyzes the different requirements necessary in such automation description and tests them in some practical use cases in a particular application (vertical agriculture). It also provides a set of analyzed instructions that might be incorporated into a description language or become a definition format for an automation orchestration description to provide the necessary process deployment options and interaction chains between multiple devices and data sources. These instructions are analyzed by implementing an execution deployment engine from the intelligence orchestration concept. The practical utilization of such instructions is verified and tested with the use case.

KEYWORDS

Internet of Things, Artificial Intelligence of Things, orchestration, intelligence, deployment, workflow, exposure, processes

1 Introduction

The emergence of intelligent devices and applications has increased in recent years. The need to interconnect such devices and services has resulted in the inception of what is known as Artificial Intelligence of Things (AIoT) (Arsénio et al., 2014) as an extension of the Internet of Things (IoT), where devices and systems interrelate without human intervention to provide intelligent services and automation to a use case. Automation and intelligence may rely on statistical machine learning models, generative models, or even simple rules and decision trees that simulate intelligence. To accomplish this vision of interconnected intelligence, several challenges and problems must be overcome. An initial analysis of these challenges and a proposed architecture to address them was given in Ramos et al. (2022).

The current research analyzes what is needed to make such an architecture functional through experimentation and application design of a real use case. It provides multiple implementation options and proposes a workflow description to model the process's automation and understand the limitations and additional work necessary for a complete functional solution. It also proposes specific solutions that target some of the challenges identified during the analysis of the implementations and the use case realization.

This article is organized as follows: Section 1 introduces intelligence orchestration, discusses the challenges involved, and provides an overview of relevant research already undertaken on the subject. A detailed analysis of requirements, motivation, and practical limitations is given in Section 2. Section 3 reviews the applicable methods, tools, and options

for implementation. An evaluation of a limited implementation is presented in Section 4. Section 5 discusses the observed challenges, further work, and conclusions from the work conducted.

1.1 Intelligence orchestration

"Intelligence orchestration" is the process of coordinating the services provided by several devices and intelligent entities to facilitate automation by the composition of workflows to achieve one or multiple goals (use cases) and to consider the relationship between multiple stakeholders and their assets. The relationships between the different entities, their goals, and fulfilling their requirements are established through policies that dictate what are acceptable outcomes and restrictions related to the interactions, especially when goals have conflicting best-result desired outcomes. Policies also help model regulation, commercial interactions, and security requirements that also need to be followed to realize the goals.

The use cases, in turn, are realized through workflows. A workflow can be defined as one or a set of processes that are executed by particularly defined parties according to a set of conditions and following a pre-established model or procedure. The processes are defined through flows that are composed of chained functions or services resembling an input → processing → output pipeline. This is one of the main differences compared to traditional cloud orchestration, which focuses on micro-services, their relationships, and their deployment in cloud platforms. Workflows assume a serverless architecture, which, compared to the cloud serverless concept, is not only about providing function-as-a-service (FaaS). Instead, it assumes that the functions are to be deployed in a platform-agnostic fashion and that the system can adapt to the underlying platform where the function will be instantiated.

1.2 IoT intelligence orchestration—Challenges and characteristics

The orchestration of intelligence for IoT use cases raises challenges that can be grouped into three categories. The first is its large heterogeneity. This refers not only to the different platforms, devices, and systems that operate in the IoT space but also to the use cases that are being addressed, the type of stakeholders and their relationship (including regulations, policies, and trust levels between them), the type of connectivity and communication conditions, and the levels of involvement from consumers and enterprises.

Another challenge is control, logic modeling, and instantiation. IoT use cases may have multiple levels of control loops and automation to be considered, and their modeling and implementation tend to be tailor-made to their intended deployment. Additionally, such modeling must consider the granularity of the processes (from a high level—for example, a maintenance process—all the way to details such as the tightness assurance of screws) including the interactions between entities that are external to each other, also known as "choreographies."

Finally, the interoperability and the semantics utilized by IoT use cases, processes, and data are another challenge due to the great number of standards, data formats, and hidden design semantics (e.g., the meaning of a client, the meaning of a connection, and the particular context where it is applied). The type of high-level modeling that later needs to be instantiated and the interoperability of AI functions and their life-cycle management processes are implemented mostly by tailor-made processes with very little interoperability.

1.3 Related work

Automation frameworks or solutions can be designed in two basic formats: declarative and imperative. Declarative methodology focuses on what is to be executed or the desired objective, while imperative methodology focuses on how is to be executed. Both these methods can be used separately or in combination, depending on the scenarios in which the components need to be deployed, since they both have their own advantages and disadvantages. We next review some examples currently in use in the deployment of software artifacts.

There are many frameworks that are available for enabling the automation of deployments in application development. One well-known framework for deployment in cloud computing and environments is Topology and Orchestration Specification for Cloud Applications (TOSCA). TOSCA is an OASIS standard[1] that enables the modeling, provisioning, and management of cloud applications. TOSCA allows the modeling of so-called "service templates" that describe the topology—the components and their relationships—of an application to be deployed to a cloud infrastructure.

The core modeling concepts in TOSCA are nodes and relationships. *Node templates* represent components of an application, such as virtual machines, web servers, or arbitrary software components; *relationship templates* represent the relations between those nodes—for example, that a node is hosted on another node, or that a node connects to another node. Node and relationship templates are given a type using *node* and *relationship types*, which define their semantics and structure them by listing their properties, attributes, requirements, capabilities, and interfaces. *Properties* and *attributes* are used to configure deployment—for example, a web server type may specify the property *port*, which is filled with concrete values in the node template upon deployment. In the TOSCA meta-model, nodes become related to each other when one node has a requirement against some capability provided by another. For instance, a virtual machine node may offer the capability that a web server node can be hosted.

Furthermore, TOSCA standardizes the so-called "life-cycle interface" by specifying that node types may have *create*, *configure*, *start*, *stop*, and *delete* operations to be used for installing, configuring, starting, and stopping them. Such

1 Topology and Orchestration Specification for Cloud Applications [Internet]. Available from: http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.html

functions are implemented by "implementation artifacts" in the form of, for example, executable shell scripts. In contrast, "deployment artifacts" implement a node's business logic. For example, the compressed binary files to run a web server are meant to be attached as a deployment artifact to the respective type. In addition, TOSCA uses the notion of "policies" to express non-functional requirements affecting an application topology at a particular stage of its life cycle; these are attached to single or groups of nodes in the application topology. In addition to input and output parameters to parameterize a service template, TOSCA defines the Cloud Service Archive (CSAR) packaging format to allow for the exchange of applications.

The issue of heterogeneity increases even further due to the availability of many deployment automated technologies that have their own features and modeling languages. Wurster et al. (2020b) proposed The Essential Deployment Metamodel (EDMM), which provides a normalized metamodel for creating technology-independent deployment models, since the majority of them follow a declarative form of deployment instruction. The structure and names of the entities are inspired by the TOSCA standard and the Declarative Application Management Modeling and Notation (DMMN) with its graph-based nature. The first entities to be defined are the components forming an application, as well as the component types that allow them to distinguish between each other and give them semantics. They also present a methodology that shows how the EDMM can be semantically mapped to configuration management tools such as Puppet, Chef, Ansible, OpenStack Heat, Terraform, SaltStack, Juju, and Cloudify.

EDMM has been further developed by Wurster et al. (2020a) by added tooling support in the form of the EDMM Modeling and Transformation System, which enables graphical EDMM model creation and automatic transformation into models supported by concrete deployment automation technologies (for example, from EDMM YAML to Kubernetes).

Serverless computing takes advantage of the cloud-computing benefits of providing a high degree of automation for the execution of services without having to worry about the infrastructure setup. The services are scaled according to the application needs, thus simplifying the management of cloud-native applications. This is achieved by serverless technologies such as "function-as-a-service" (FaaS) that are triggered when events are observed by the cloud applications. RADON (rational decomposition and orchestration for serverless computing) is a framework based on the serverless computing paradigm (Casale et al., 2020) that enables defining, developing, and operating (also known as DevOps) applications based on FaaS computing. It applies a unified model-based methodology to define these applications and aims to be reusable, replicable, and independent of proprietary DevOps software environments, thus leading to the creation and use of templates for cloud-based applications.

The RADON framework deployment implementation (Dalla Palma et al., 2023) is based on the characteristics of serverless computing such as execution of service on demand, dynamic scaling, focused and stateless functions, use of third-party functions, and reusability of templates. Its life-cycle model takes care of the basic DevOps needs over six phases: verification, decomposition, defect prediction, continuous testing, monitoring, and continuous integration/continuous deployment (CI/CD). RADON also provides a graphical modeling tool to ease the design of functions in the form of workflows and verifies the orchestration procedure before deployment. It relies on the extensive collection of standardized TOSCA-based templates to abstract deployment characteristics. RADON supports event-based application deployments involving FaaS as processing components, object storage services, and database-as-a-service (DBaaS) offerings as respective event sources. New custom extensions can be developed using the definitions provided by RADON's TOSCA and community-sourced extensions. The main limitation of RADON is that it only targets cloud-based orchestration and so does not interoperate with systems that do not follow their toolkit or devices with their own platform systems, which cannot instantiate cloud jobs. In addition, some work on FaaS workflow instantiation is ongoing in the form of an open-source framework called OpenWolf[2] (Sicari et al., 2022), which shares similar principles but also the limitations of RADON.

2 Intelligence orchestration requirements (architectures, services, and support systems)

The ideas outlined in Ramos et al. (2022) and in the "Intelligence orchestration for future IOT platforms" newsletter article[3] can be reduced to three main components: an exposure and discovery service, a workflow and processes composition service, and a workflow instantiation and deployment engine. In relation to the first and part of the second services, some work was attempted in Jayagopan and Saseendran (2022) but was limited to an implementation model where the discovery was provided by a script that fed a database. The sources of exposure need to be known in advance for this kind of model to work. Therefore, dynamically changing sources (for example, a car that moves from one place to another or a service that is available between some opening hours) are difficult to incorporate into such an implementation.The discovery of the exposed services, capabilities, and functions is a critical functionality for providing dynamic orchestration and effective mapping to the blueprints and templates of workflows. The realization of this discovery is not trivial: it requires not only knowledge of the location and access information of the different services but also a semantic description that is understood by the respective domains that make use of the workflows. The cost and requirements for discovery would vary according to its implementation models. With respect to exposure and discovery, we could think of four types of models (Figure 1).

2 OpenWolf: A Serverless Workflow Engine for Native Cloud-Edge Continuum. https://github.com/christiansicari/OpenWolf (Accessed July 2023).

3 https://iot.ieee.org/newsletter/november-2021/intelligence-orchestration-for-future-iot-platforms, IEEE Internet of Things Newsletter (Nov. 2021), authors: Edgar Ramos, Senthamiz Selvi Arumugam, Roberto Morabito, Aitor Hernandez, Valentin Tudor, and Jan Höller.

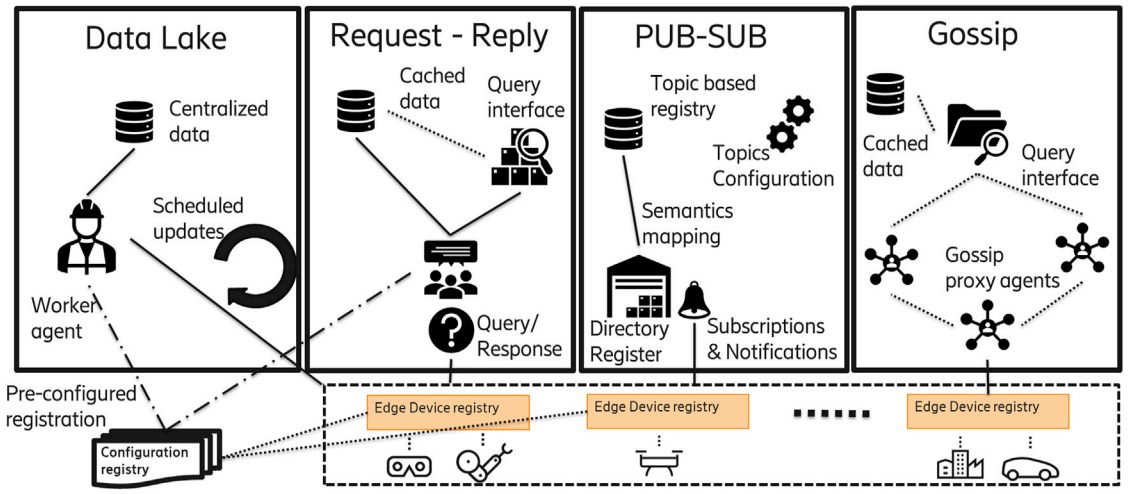

FIGURE 1
Possible discovery models for capability exposure.

- Centralized data lake, which implies that knowledge of all the services and capabilities of the devices is in a centralized repository. Every device change is updated and known, and every platform or device management system is pre-defined (known in advance); there is therefore limited support for dynamic registries. Any update of the central database needs to be scheduled, and, consequently, information may not necessarily be up to date between updates. The query of each registry for the states of devices (active, inactive, available, *etc.*) or their capabilities may also consume bandwidth and time, depending on the context and number of associated registries and devices.
- Request–reply model, which would require querying the registries to find the needed functionality or devices. A typical request–reply model is exemplified by REST (REpresentational State Transfer) APIs. These queries might place quite a high load on the network due to the spread of the request and the replies from each of the registries, depending on the number of devices and registries accounted. Some optimizations, like a cache, could be used, and then only updates to the stored values are needed. However, that would also require some sort of validity label for the cache; the risk of changes during such a time interval not to be considered during the discovery. In addition, the target registers need to be known in advance or at least pre-registered so they can be addressed by the query process. Furthermore, a higher-level query interface to the orchestration layer is necessary to provide a semantic and contextual environment that can be translated into the particular API or request that each registry type supports.
- Pub-Sub model, where it is possible to enable asynchronous management of exposed capabilities and services. This feature makes possible dynamic registries and devices, services, and capabilities that are mobile or have multiple states or availability. The services being provided with a semantic that matches topics used to be published in a directory register. Therefore, a translation between semantic contexts might be needed to match specific applications and domains of use cases to match the vocabularies used by the topic exposure. The network aspects of pub-sub protocols should also be considered. The size and number of registries belonging to an orchestration context might create considerable stress in the network when the updates or publication process is executed, especially during bootstrapping or mobility events.
- Gossip protocols operate in a peer-to-peer environment and use the same principles as an epidemic spread. Even with their inherent limitations (Birman, 2007), their dynamism and fast converging features provide a good base to utilize a gossip-like structure through agents that might represent either a registry or a device. The system requires a query interface to the orchestrator that, like the pub-sub, might require semantic equivalence translation. In addition, a caching system could be used to optimize semantic search and mapping to the agents with the capabilities and services required.

2.1 Instantiation and execution control of workflows

The instantiation of the workflows requires an execution engine that can access the devices and services that are orchestrated in the workflow description. The architecture of an execution engine requires workflow ingestion that resolves any prioritization of the workflow and a parsing function to interpret the workflow description. The function calls and triggers are managed by the engine using adapters that match the function calls to each system and a platform to their native commands. These adapters (deployment agents) use the principle of a device driver to provide the mapping, and each protocol, data model, or platform can be onboarded in the engine, providing direct support for the heterogeneity of devices. Mapping to the particular commands requires a set of primary instructions that can be derived from the workflow descriptions. These primitives can then be mapped

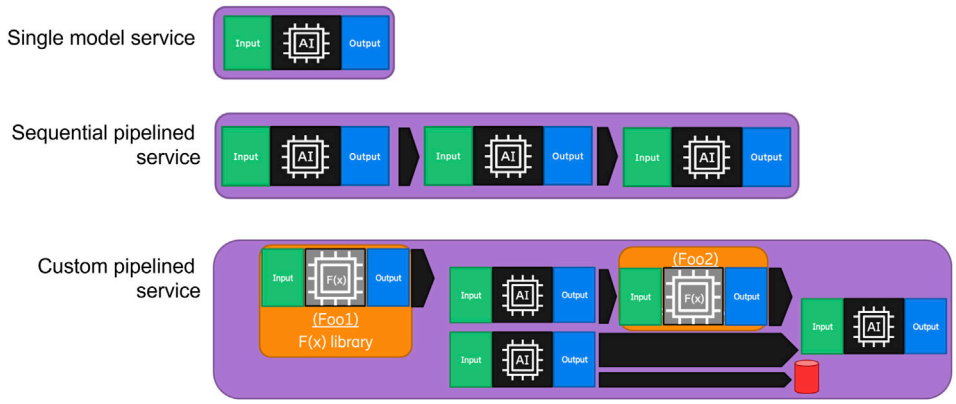

FIGURE 2
Pipelining and service composition models.

onto any of the different platform implementation interfaces (commands and APIs).

2.2 Workflow composition

A workflow is defined as one or a set of processes that are executed by particularly defined parties according to a set of conditions and following a pre-established model or procedure. Workflows have one or multiple goals, and the processes help realize such goals by orchestrating the concerns of the digital domain related to the goals. This means that the workflow does not model processes that are realized in the physical domain but represent the role of the digital domain, in, for example, a business logic. Whenever there is a dependency on a physical world event, there should be a mapping of this event onto the digital world so it can be addressed by the workflow processes. For example, if a process requires that an item is received physically at a location to start, then the reception of the item should be modeled with a digital trigger, such as a button, form, or any other digital asset that could signal such action. The workflows resemble flows or pipelines of services or functions where the outputs of a function may be connected to the input of another. The functions may also be a digital realization of actuation commands or interactions with the physical world.

2.2.1 Intelligence pipeline

Defining a pipeline means providing a description of how data flow from one function to another, and the triggers of each function take into account the cadence of each step. Some functions are executed at a higher frequency than others, and the data intake might need to be adjusted accordingly. For example, in some cases, the input data used are an average over a given period, the last value, or the most frequent sample from a group. In other cases, the data need to be collected and used in totality as input or treated in a moving window fashion. The pipelines may resemble multiple types of constructions (Figure 2). In the simplest case, a pipeline depicts just one service that has input and output. However, even in that case, a service may be a composition pattern using pipelining recursively, meaning that the single service depicted may in turn to be a pipeline in itself.A pipeline defines an operation that implies multiple services. In some cases, the definition of the pipeline might be abstract, meaning that it is defined as a blueprint using service definitions that can later be instantiated from the template to a particular implementation. On the other hand, the definition may already be particular for a system, including information that is relevant only to that system. One aspect to consider is how this blueprint is defined and the differences with an instance-based definition. Even when the semantic definition of the services may help mapping, the actual instantiation may require more parameters and practicalities that also must be addressed during design. For example, as mentioned previously, the cadence could be different between two functions in different systems, as well as the speed of execution and performance and the intake and outtake volumes of data that can manage each instance. Furthermore, the nature of the triggers and how they are connected to real systems' implementations should be considered. Another consideration is data fetching and forwarding between the functions. In some cases, this can be handled by the devices themselves, or the exposure platforms used or required to be implemented as an additional function. Such a function would ensure the output data are published so that those data can be retrieved by the function that next uses them as input. Some possibilities are to store the data in databases, permanent or temporal data storage, or publish it through an external platform service. In some cases, data retrieval would be consumable by a device requiring a similar function. These functions may be also added to the device's code base or provided by support infrastructure either in edge, network, or cloud.

2.2.2 Choreographies

The pipelines may address processes from multiple stakeholders and subsystems (Figure 3) and model their practical digital interactions. In Figure 3, a factory that produces goods has its own processes and requires interaction with customs and the store that receives the items. Each entity has its own processes and, in some cases, even separated subsystems such as the security and inventory of the store. The subsystems and entities may cooperate and may need to follow up on their own processes and those of other concerned entities (for example, the clearance process from

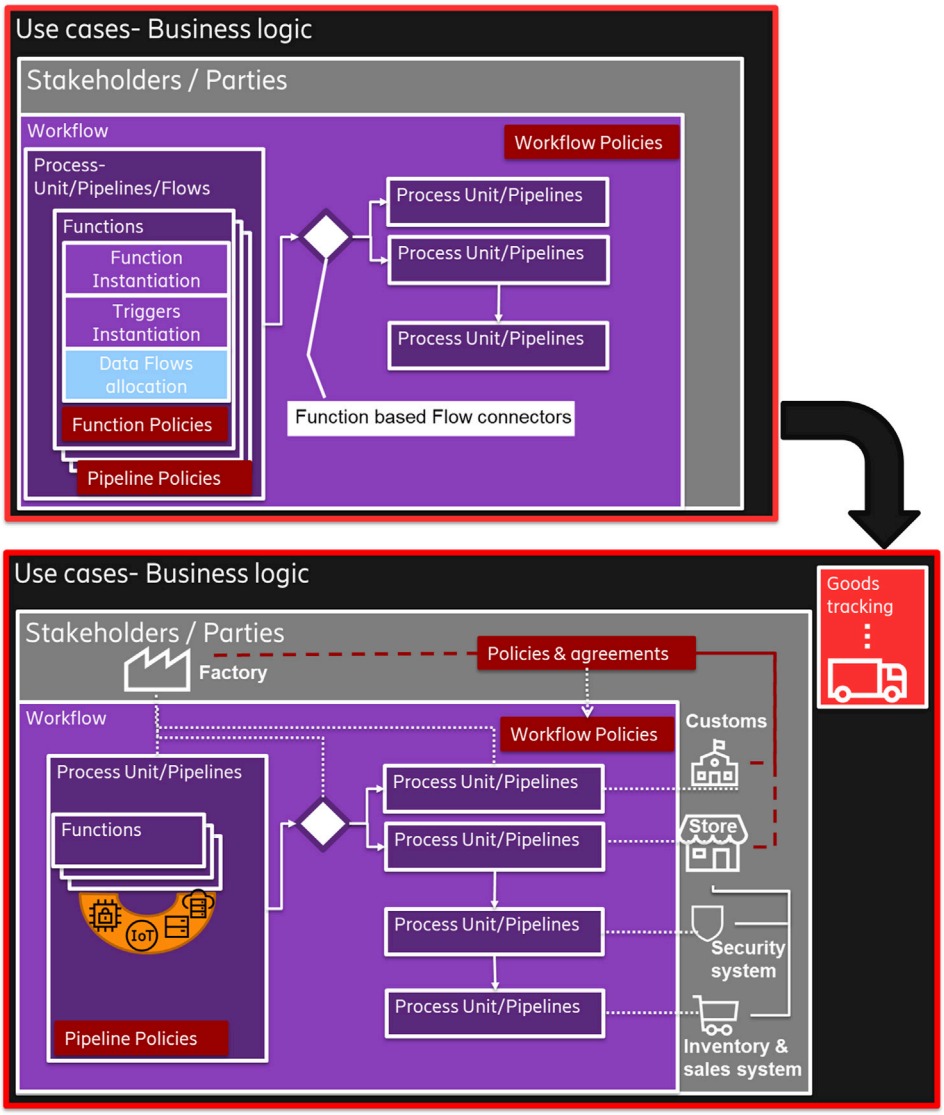

FIGURE 3
Functions and intelligence pipelines in the context of use cases, including choreographic workflows.

customs). The whole relationship is regulated in many cases by non-functional requirements provided by policies, regulations, and agreements.

The workflows are not necessarily static: they may require updates that may vary from the actual process to the actual instantiation or service executer. This may be the case when highly mobile or state-oriented devices and applications participate in a use case. Devices in such use cases may appear and disappear from the system context, such as a truck that moves from a construction site to a supplier location. This leads to different courses of processes that may be followed according to the application state—if the truck is fully loaded, first then it might require unloading before loading new material. In addition, the policies may have an impact on the process execution and data sharing between parties that may result in the modeling of multiple branches of behavior that match each policy type. This also raises the issue of resource administration, particularly when such resources are shared (multi-tenancy). Security and policies must be taken into consideration, such as when, by whom, and for what a resource can or cannot be used, what are the priorities between the different tenants, and the need for isolation between their "execution spaces." Each workflow has an owner whose goal is being fulfilled by the workflow, which orders the execution of the processes.

2.2.3 States in workflows

The workflows required from intelligence orchestration may require states in some cases and can be stateless in others. Several machine learning frameworks (e.g., reinforcement learning or online learning) require updating a set of parameters to the model that are being adjusted according to the learning algorithm during the execution. This means that the algorithm may sometimes be in an "inference state", while, in other cases, the function may be used for a "learning state" and therefore update the learning operation parameters. This may be a feature that is specific to the functions that are applied, and the facilities provided by the platform

underneath that may facilitate or provide support for state saving. The implementation of the function may be achieved such that the state is fetched from storage and updated accordingly. The main problem with those approaches is the lack of transparency for a hypothetical automated orchestrator, which would have no way to fully semantically comprehend the function and its characteristics (such as the states). This function might require establishing independent or complementary flows that address the states of the functions; awareness of this situation would then be better handled in the orchestrator if made explicit. Therefore, an explicit definition of state in multiple levels (workflow, pipeline, function, and trigger) is useful for increasing the flexibility and simplicity of automation. The states may then be defined as part of a function description or in the support system for orchestration to handle the workflow composition.

2.2.4 Blueprints and templates

Blueprints are comparable to a reference architecture that can be used to replicate the design. The blueprints are derived from patterns observed when certain requirements of use cases match. This facilitates reuse and replicability, which are crucial for scaling purposes. This can be applied to workflows. Many processes follow the same steps and functions, but their pre-conditions, participant elements, instantiation entities, and output mapping may differ, so they need to be adjusted to their context. To facilitate the composition of workflows, templates may need to be introduced to facilitate the process definition. When used, a template is only required to explicitly define the instantiation nodes of the functions, the mapping of the function to the ones executed in such instantiation nodes, and their trigger configuration. The blueprints are a more abstract step than templates. They serve more as guidelines to construct workflows and could be implemented as a component (or set of processes) to be fulfilled by a part of the workflow. Such components could also be provisioned by instantiating a template; however, the difference is that a blueprint component could map to multiple templates, each of them fitting some particular case. For example, a massive monitoring blueprint may have a component where the different sensors and measurements are defined and fetched; depending on the monitoring, a template could be used to define the monitoring of, for example, a power grid. A blueprint component could specify processing for the monitored results, which could be as simple as storing them in a database, filtering them through special functions, or forwarding them to some entity. Yet again, several templates could be applied, depending on the regulation of a country or a particular company's practices, or the type of equipment being monitored. Finally, the last blueprint component could be the exposure of the meaningful output. This may be realized by templates that apply to the exposure platform needed for the target systems that would use the output. A format definition of templates could be thought to be derivative of business logic. Therefore, BPMN (Business Process Model and Notation) could be a candidate for providing such templates.

2.2.5 Multi-tenancy and shared access

When instantiating multiple workflows, a device might finally be shared by multiple stakeholders. In the exposure of values or data, several IoT systems address this by access control combined with an exposure API that gives access to the data. This access control is basically another interaction model that should be described by policies. Therefore, several aspects of policy implementation (enforcing) and monitoring are important to consider, from the discovery of the services and the configuration of the data sources to the actual instantiation, prioritization, and triggering aspects of the workflows. Another consideration concerns the sharing and isolation of states and data. For some use cases, it is necessary to share state and information between stakeholders, while, in other cases, processing and functions should be preserved and kept separate between multiple users and the contexts of the functions' applications. From a platform perspective, there is therefore a need for a specific level of isolation of the exposed result after workflow processing; this would result in multiple outputs being exposed, depending on the workflow's stakeholder. This exposure could be temporal (intermediate results) or persistent (unchanged until new processing by the same stakeholder).

When considering multi-tenancy and shared access in devices, some aspects need to be considered. One is the secure execution of (encrypted) workflow components when required, where enclaving and access policies play an important role. This needs to be supported by both software and hardware security solutions. Access control and policies need to be defined and enforced to ensure the confidentiality of processes and data in the device itself. The support of multi-tenancy could be enabled by virtualizing the device services and capabilities. The virtualization directly supported by device management systems (such as LwM2M) is a potentially powerful enabler of multi-tenancy use cases by providing support for instantiation, shared and private states and outputs, and access control.

2.3 Orchestration control

Orchestration control should be implemented by the same tools provided by the orchestration framework. Orchestration control would become the use case whose goal is to support the orchestration of workflows, thus becoming its own set of workflows, policies, and blueprints. The orchestration control would not only the workflows and functional requirements but also fulfill non-functional requirements, such as policies.

2.3.1 Workflow management

Workflow management implies the placing of workflows in the relevant deployment engines, their prioritization, triggering, and storage, which is highly related to life-cycle management (Section 3.5). Each life-cycle management state would map to a workflow implementation that is complemented with specific policies that, in addition to providing access and management control, would also provide priorities, trust management, and ownership detailing administration. Workflow management must, in most cases, be instantiated in a distributed manner so that multiple systems can manipulate and deploy workflows on associated engines. This implies a relationship between the entities that control and manipulate the workflows and the deployment engines that deploy the services in the nodes. This association could be performed separately through trust management mechanisms (Section 2.3.3) or may be inherited on the implementation

platform if it does not require multi-stakeholder integration from the workflows. Workflow management compresses an API that taps into the different parts of the system where the workflows are utilized. This set of APIs can also be exposed, for example, through a proxy agent to provide services to another AI agent (for example, a reasoner), making it possible to autonomously manage and orchestrate workflows. Such an agent would either have additional services to facilitate the association of the AI agent with the other entities that require a specific trust association or would have direct access to the management controls.

Another aspect of management is observing and monitoring workflows and processes. Enabling such observability is possible with the same pipelining framework. The functions required for observability are exposed and implemented by the deployment engines, and they can be organized in observability workflow templates that are applied to the actual orchestration workflows. The configuration and specifics related to the timing and detail of the monitoring can be applied to the triggers and management of the data collected (exposed, stored, streamed, *etc.*). Thus, the workflow concept can be applied to help control other workflows and provide real-time or close to real-time follow-up of the processes. This monitoring would allow the control to branch out and create more complex orchestrations where alternative workflows could be executed according to the overall system state.

2.3.2 Policy management

As with workflows which require their life cycle to be managed, policies require a similar handling. In the scope of this framework, the expectation is that policies are specified with a descriptive language. Policies populate multiple levels of the stack and platform, and they must be processed on different levels which may have totally different consequences for the target systems. For example, in a cloud system, a policy related to privacy protection may be just a restriction on visualization from certain database fields or data anonymization, but, in a device, might mean restricting reading access to a directory from the persistent storage. Therefore, a policy engine that configures, enforces, or monitors policies would have to be present on multiple levels of the platform. In each level, the actions and the description of the policy might vary to suit the right level of the stack and might require a reinterpretation of a general policy for more concrete action to be enforced at a lower level of the architectural stack. This study does not cover the generation and life-cycle management of policies; this will be a matter for future research. Similarly, the implementation required to manage the policies can be based on their own "systems" workflows. Another issue is that the sensitivity of the data produced and the security related to the access on devices might be easy to be compromised; some work on Shared Secret (SS) (Galletta et al., 2019) and Nested Shared Secret (NSS) (Galletta et al., 2021) are being investigated for application to such cases.

2.3.3 Trust management

Framework deployability depends on trust establishment. There are multiple ways to establish trust and dependencies for each platform participant of the orchestration framework. The main requirement is that the different components of the architecture can communicate and trust that their identities and access rights are agreed upon. To facilitate this part of the integration, a trust management function is required that mediates between the different systems and provides security and trust; it must support the security associations between the components of different domains, entities, or platforms. The mediation itself requires a trusted entity, which may provide either a distributed trust schema (e.g., ledger-based) or delegated trust that can provide the roles and certification required to map between multiple domains. The research questions relevant to this point are not in the scope of this study, but several possibilities and solutions could be explored to develop and cover the requirements of trust between the architectural components.

3 Applicable tools and methods for intelligence orchestration workflows

3.1 Workflow description

Several factors need consideration when deploying the functions of the devices. There are considerations apart from the actual definition of what service the function is providing and how a machine can match such a service to its own needs semantically.

The first consideration is where the function is being executed or instantiated by means of exposure. The exposure of functions and capabilities must be addressable from a network perspective. This means that the functions can be remotely activated by an API or a command to access functionality. For example, in a Lightweight M2M (LwM2M) object, a resource or a function is called by identifying the resource (an atomic piece of information that can be read, written, or executed) using a URI to access it. The URI follows the pattern where the exposure server URL is followed by the clientId/objectId/instanceId/resourceId (Alliance, 2014). The description of a workflow utilizes this remote addressing of capabilities and includes the necessary information to address functionality.

A second consideration is the definition of the triggering mechanism of the function—control of when the function should be executed and, in some cases, stopped (execution terminated). The instantiation of the triggering functions should also occur in the execution engine, where the triggers could be derived from custom functions, exposed functions, or a set of predetermined functions (such as for scheduling). The triggers also apply to the execution of workflows since a workflow can also be represented as a composed function.

Finally, the data flow setup is related to the inputs and outputs of the functions: how the required inputs are fed into the function, and what is carried out with the result of the computation and the function output.

A proposal for a functional structure of a workflow description could include the following components for addressing the functions and services.

- Model, code, or script of the function or service to execute. This field may have a binary or a piece of code to execute that is compatible with the target platform or device.
- A version number of the current model.
- Description URL pointing to information describing the operation executed by the function and possible metadata related to the input and/or output. This description may be

just a reference to an ontology vocabulary or a relationship description in a semantic description language (e.g., by means of Web Ontology Language—OWL).
- Model URL: If a function that is natively provided and exposed by the device or platform is to be executed, then instead of a model, a URL of where to retrieve the model to execute it or a URI to identify it to execute it (and retrieved with alternative methods) is provided here.
- Model integrity: The cryptographic checksum that might be required to ensure that the function meant to be utilized matches the one to be executed if required by the deployment.
- Model description: A URL pointing to information describing the operation executed by the function and possibly metadata related to the input and/or output (e.g., following the model proposed in Ramos et al. (2020)).
- Inputs: The data sources for the model when it requires them. It could be a set of links/URLs that are pointing to resources at the device or from another device to use as input for the model, or a stored or exposed value to be retrieved.
- Input transforms: Sometimes the input data cannot be used as is and might require certain transformations. A function or model used for transforming the data from a local resource/external source to a format that can be used with the given model can be declared in this field (or a URI to such a function).
- Input labels that describe which input source is connected to which input of the model, mostly to provide semantic description to the input targeting humans or audit systems.
- Outputs: A set of results of the model operation and specification of management (store, forward, or expose). In the case of an actuation function, the output would only indicate that the service was executed.
- Output transforms: Transforming output from a model-specific format to a resource-specific format; similar to input transforms.
- Output labels that identify and semantically describe the outputs of the model, similar to input labels.
- Condition expression to trigger the execution of the model if needed, such as executing a service after a specific number of input samples is received.
- Implementation ID of the particular service implementation or belonging device.

In order to manage the triggers, these additional components are also needed in the workflow description.

- Trigger-type: START, STOP, SCHEDULED, SEQUENTIAL, CONDITIONAL, according to Section 3.2.
- Trigger function URL: If the trigger requires the execution of a binary function, this URL points to where to retrieve the model to execute or a URI to identify the model to execute (and retrieve with alternative methods).
- Trigger function: Code, script, or model of the function or service to execute (similar to the model field from the services).
- Variable inputs: A set of links/URLs that point to resources at the device or from another device to use as input for the trigger function or as a discriminator (for a trigger to be activated or not).
- Cycles: Number of repetitions of the trigger activation, meaning that the target service or function would be executed several times according to the configured values.
- Transformation: Changes to variable inputs that are required after each trigger executes (same as the input and output transformations of the models).

3.2 Trigger control

Like the functions, the triggers need to be instantiated. They require an execution environment where they are executed and monitored. The triggers are basically a Boolean function where, due to the processing of the input, the output is a true or false statement (or a Boolean bitmap). The output is then connected to the pipeline logic so that the associated function or functions are executed when the condition of the trigger is fulfilled. The handler of the pipeline should then take care of two aspects related to the trigger. One is the application of the trigger, which has to do with the trigger's nature. A trigger could be.

- On-demand: A trigger that monitors a control that, once activated (e.g., the push of a button), fires up the function or is explicitly called upon to be executed in the platform that manages the specific trigger.
- Event-driven: The trigger would activate the function based on the fulfillment of one or multiple events.
- A sequential trigger is somehow related to an event-driven trigger. It consists of activating a function once a previous function's execution has finished. The ending of the execution of a function can be modeled as an event; however, in some platforms without event support, a sequential execution can be scheduled instead. A variation of a sequential trigger is starting a function once another function has been started instead of waiting for the end of the execution.
- A scheduled trigger would execute a function at a specified time or delay after an event or end of the execution of a previous function. The scheduling can also include repeating the activation of the function at certain intervals of time, at the same time, or as a recurrent activation according to an input calendar event.
- A condition-based trigger activates a function when a predefined condition becomes true. This requires that the precondition is verified either according to an event, at certain time intervals, or a combination of both. The condition to trigger the function execution could apply to a particular/single data source (e.g., if the air temperature is greater than 20 ℃, then monitor the water pressure) or to a combined data source condition that requires that multiple conditions are true (if temperature is 20 ℃ and humidity is greater than 60%, then execute the function). In addition, matching a pre-provisioned pattern test may span multiple samples—for example, if the temperature follows a +10, −10, and +10 ℃ change during the last three measurements, then activate the function. Another interesting possibility is adding a ML or AI model to make the decision of triggering; since the output required is Boolean, then any AI model that produces such output could be used to control a trigger—for example, a

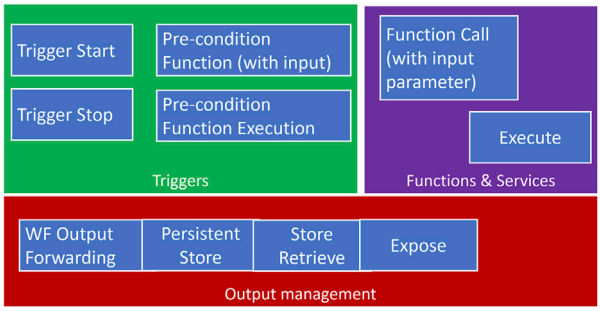

FIGURE 4
Instantiation primitives from the workflow descriptions, which are then mapped to IoT management systems.

match for a face recognition-trained model. In some cases, this AI model has to be provisioned to the execution environment through the LCM mechanisms in equal form as the functions and services.

3.3 Instantiation primitives

The instantiation of workflows requires an adaptation to the multiple platforms and systems that host the executing engine. An implementation that parses a workflow description may implement the same basic commands that can be mapped to each of the different platforms and systems that have access to the functions and services that are referenced in the workflow description. This set of basic commands can be considered primitives that any execution engine would have to apply. Figure 4 provides a summary of the primitives and their context. In relation to functions and services, there are two basic primitives: an execution call for services that do not have any input and a function call that also includes parameters and inputs. The result of the function execution may either activate an actuation or produce a data output. In the latter case, the management of the output needs to be considered, and there are four options that can also be mapped to primitives. Workflow output forwarding streams the data over the network to another location or to another workflow or external process. Persistent store stores the output so that it can be used later and even retrieved several times for either other workflows or for later stages of the same workflow. Store retrieve is itself another primitive command that allows the fetching of a stored value and uses it directly in a workflow. The storage must be defined as a variable in the workflow description and be instantiated by the executing engine in a particular storage location that can be retrieved by the same executing engine or exposed for use by other executing engines or by additional workflows.

Finally, the triggers' primitives allow the instantiation of all the types of triggers that are defined in Section 3.2. The trigger start specifies the function that should be started, either immediately or after the evaluation of a pre-condition, and the execution should keep being recalled until the trigger stop for that function is called. The trigger stop may also have a precondition function (e.g., the number of executions of the function or a time of the day) with or without input parameters. All the different trigger types (*event-driven*, *sequential execution*, *scheduled*, *on-demand*, or *conditional*) can be instantiated as a combination of these primitives.

3.4 Deployment agent

The deployment agents adapt the primitives to the protocol, management system, or API utilized by the device (e.g., Web of Things, LwM2M, and OPC-UA). The workflow composition only considers commands or services that could be possibly instantiated by the particular device or service. Therefore, the instruction should allow the system to be mapped. Instructions or values that are not compatible with the particular protocol, platform, or API should not be present in a workflow to be instantiated. The reason is that only supported functions from a service or device are exposed and possible to be orchestrated. There could be cases where a template workflow cannot be fulfilled by the available services; in such cases, the template needs to be modified to adapt to what the services or capabilities can offer. However, this would be resolved during the workflow design, and the deployment agents do not need to deal with such complexity. A typical deployment agent would, for example, set a value for a device function (e.g., a thermostat of an air conditioning system) or execute an actuation function. The development of Semantic Definition Format (SDF) (Koster and Bormann, 2023) provides a good starting point for interoperability between the different types of agents and their implementations. SDF is being designed to convert most of the IoT data and interaction models between each other, and it is certainly a promising technology for supporting the deployment agents' development.

3.5 Workflow life-cycle management

Several aspects of the workflow life cycle need to be considered, from the moment of design to the point of decommissioning.

3.5.1 Design and composition

Workflow composition should consider the context of the workflow's operation. This is not only domain-oriented but also concern-oriented, requiring mapping to the stakeholders' own goals and reflected in the process' final result. There could be workflows that require coordination with each other that would depend on synchronization functions to enable data sharing between workflows that may concurrently execute. In addition, policies would also need to be considered to regulate interactions, such as prioritization and queuing. The design of workflows is based on the available and exposed functions to the orchestrator. There could thus be access policies that limit what an orchestrator could utilize and a context where the discoverable functions are set. This means that what is available to the orchestrator depends on what the discovery functions can return and make available for the workflow design. Another consideration is the actual description of the workflow. Once composed, a workflow requires a way to detail each of the aspects that are relevant to it, so it can be used to instantiate it or as a blueprint. Since the principles for the workflow are inspired by serverless technologies, the descriptions used for FaaS are relevant. The requirements for the description of the workflow are studied further to enable their mapping to a particular description language.

3.5.2 Publishing, onboarding, and instantiation

Even when a workflow is distributed in the nature of the processes that compose it, its description must be specified and stored (published) in a particular control instance. The workflow description could be either shared in full with each of the entities involved and require instantiation, or only the relevant orchestration commands to a particular entity need to be delivered to be instantiated. In both cases, there is a need for a control function to contact the instantiation targets. The reference architecture introduces the concept of deployment engines, which are capable of receiving the workflow description and addressing each of the entities to instantiate the workflows through deployment agents. One deployment engine must integrate a deployment coordinator that provides communication and coordination functions with other deployment engines that have access to a different set of devices or instantiation entities. The deployment agents are adaptation functions that translate the orchestration calls or commands found in the workflow description to a configuration or programming of a specific device or node. The agent would act as a device driver that enables the instantiation of the workflow in a particular node. In some cases, the agent is merely a communication interface to a device management system, but, in others, it may deploy a function from a FaaS execution environment in a platform or device. Once a workflow instantiation is introduced in a device, it remains instantiated until a new workflow is provided, the workflow lifespan is finished, or the workflow is explicitly canceled.

3.5.3 Lifetime: time span when a workflow is applicable

The applicability of a workflow may have a limited time. This means that a workflow should be functional for a period of time that is synchronized to all the different entities that instantiate a workflow. One task of a deployment engine is to track the lifespan of a workflow and coordinate with other deployment engines involved to terminate it. The workflow's lifespan could be measured in units of time (seconds, minutes, hours, days, *etc.*), in its number of iterations (how many times a workflow is executed in full), or event-based (terminated after an event happens). There is a connection between the lifespan of a workflow and its termination (or decommissioning). However, a workflow might be scheduled periodically or its re-execution is triggered multiple times by an event. If there is some sort of recurrency configured within the workflow, the system should not fully decommission it; instead, it should be reactivated later. This raises the question of how a device may be mapped to active and inactive workflows and how it transitions between them.

3.5.4 Updates

A workflow may require updating and re-instantiation after it has been deployed. This means that it can be modified and change the process's sequence or elements, in addition to where the instantiation is deployed and the impact on the data flow between its origin and target execution nodes. The transition that applies to an updated workflow should not only be considered when the new workflow starts but also in the propagation of workflow updates to the new and old relevant nodes. Additionally, in some cases, the target node that implements a workflow may not be the same in different iterations of the workflow. An example is if a nomadic (mobile) node that arrives at one location and becomes part of a workflow leaves, but it is substituted by another node and requires an instantiation of the function in the new node (and termination in the previous node). One possible solution is to model these updates as a "control workflow" that manipulates the process's workflows using supporting orchestration functions from the underlining platform and the deployment engines. This would be either part of a version or implementation of such workflow control system or be manually specified by an orchestrator.

3.5.5 Optimizations

Goal-oriented systems always seek ways to optimize and improve their operation so that their goal fulfillment becomes optimal. This means that the processes might require updating or amending to adapt to situations, particular conditions, infrastructure, or changes in the environment. Such optimization may sometimes be part of workflows that aim to provide such adaptations. Another possibility is that such optimizations are modeled in the workflow itself, where functions make the best decision according to the optimization model. A third option basically runs workflows and analyzes their performance using an external optimization tool to create a new and optimized workflow that is later updated. Workflows including reinforcement learning naturally run an optimization method. The changes related to the optimization are modeled as part of the workflow itself and are a good example of how the optimization processes can be explicitly embedded in operations.

3.5.6 Pre-conditions and certifications

Certain processes must be fulfilled as prerequisites for other processes to commence or finish. This is particularly challenging when the intelligence orchestration processes have multiple stakeholders, and platforms or include multi-tenancy aspects. First, the precondition definition can become quite complicated since it might require multiple processes and even a limitation on the time when it was last executed. This means that a workflow or a specific process execution may need fulfilling in a minimum time range that might be set in the precondition (e.g., a specific process must be run within 10 min before starting the new workflow). A possible solution to simplify the management of preconditions is to introduce certifications when required. The certification would be proof that a process or workflow has been completed and include some additional details such as the time commenced and completed. It could be signed by a mediator and mapped to either a workflow or a pipeline execution. The certificate is provided to the orchestrator of the workflow (owner of the workflow) being executed. The certificate might need to be exposed to allow other stakeholders to be notified that those processes have been completed or were completed before starting their own processes. The certification would mean in some cases that a verification mechanism be accessible to the mediator for process execution to be validated. The mediation could be just part of the workflow and powered by ledger technologies, as one possible solution, but also could be part of a delegated trust schema.

3.5.7 Quality-of-service (QoS) and quality-of-experience (QoE)

An additional aspect to be considered when executing a process is elements such as its cost, performance, resilience, and adaptability.

Several of these can be modeled as policies or non-functional requirements. Thus, in addition to the possibility of executing a function, service, or process, there are additional requirements that require consideration. Some can already be addressed during the design phase of the workflow. In some situations, quality assurance should be guaranteed during the whole execution of the process and may either limit other processes or services that compete with the same resources or limit the type of additional actions for an actor participating in the process (e.g., restricted mobility). This calls for prioritization of processes and possible blocking of resources—challenging in multi-tenancy environments.

One possibility of addressing real-time QoS is utilizing a policy engine that monitors and ensures that the requirements are always being fulfilled and maintains the state of the resources to perform accordingly and also administers availability according to prioritization policies that are also input to the system. This would work as an overlay system on top of the workflow, and the policies applied should possibly be part of the metadata used on the exposure of the resources and functions whenever used. Changes in policies should be notified to all the parties affected by the new policies who depend on the resources.

The policy engine may also be modeled as a workflow, but it might have dependencies on each particular node or equipment. The challenge of this approach is related to transparency. If the policies are handled in a separate workflow, then how they impact related workflows is communicated as a concern. Furthermore, if a workflow has already started and suddenly a function is blocked or unavailable due to prioritization of another function in the node, the fate of such a workflow must be defined and not left to the implementation. The risk of re-prioritization should be known at the time of design and the workflow provisioned with contingencies.

3.5.8 Decommission

As mentioned previously, a workflow may terminate its execution cycle, but it might be re-executed at a different time, sometimes even sequentially after finishing the previous execution, and, in some cases, even in parallel. This termination would not mean that the workflow is decommissioned since is re-used again. Whenever a workflow is no longer meant to be used, it can be considered decommissioned. Decommissioning implies that the publishing of the instantiation of the workflow will be eliminated from all the related deployment engines, and whatever policies related to such a workflow are also removed from use. Some workflows may be kept in the form of templates, which means that they are used to instantiate other workflows. A template normally lives in a workflow design environment and is not visible to deployment engines. The trigger for decommissioning could be on-demand or follow a condition, such as the programmed lifespan of the workflow to expiration or an error condition. A clean-up procedure for decommissioning needs to be exposed by the orchestration platform and tailored to each deployment engine involved.

4 General execution engine implementation and testing

To test the technical requirements and needs of the instantiation of an execution engine, a prototype was built and tested using a vertical agriculture use case. The prototype was built using a simple architecture composed of a parser that interpreted the workflow description using a JSON input file. The parser translates the workflow description to a set of primitives (Section 3.3); depending on which platform, protocol, or data model interface was used for the exposure, the primitives were then evaluated and the execution commands called by a deployment agent plugin. The deployment agent was built in this prototype only for the LwM2M protocol since the test environment operated only with that management protocol. However, the design allowed the introduction of additional deployment agents for the APIs of other protocols or platforms. The triggers are instantiated and controlled in the prototype engine, and the scheduling functions are implemented in it. The engine implemented also had some limitations. In this first version, there was no support for persistent storage or exposure of the workflow or process results. Additionally, the life-cycle management of the workflows was very limited, and only the onboarding and termination operations were supported. The engine had direct access to the function exposure platform and no dynamic security establishment was needed.

4.1 Vertical agriculture use case

Vertical agriculture or farming is a growing trend where vegetables are cultivated using vertical surfaces under a fully controlled environment[4]. Such crops utilize control systems to provide irrigation, fertilizer, pest control, and light that are optimized for the specific type of crop to maximize production or produce a specific improvement in the final result (e.g., to heighten the sweetness of strawberries). Our deployment engine was tested in a testbed including devices with sensors and actuators using LwM2M for device management. The main studied use case was the irrigation, humidity control, and ventilation of specific crops during the growing phase (after germination).

A complete system would include the components outlined in Figure 5. However, for this testbed, we managed an irrigation valve, a fertilizer fluid valve (to mix with the water), and a fan switch control. Every crop was located in a particular part of the vertical trellis and the location and type of crop were recorded at the start of the growing phase, as well as the state of their development cycle. The location was matched to specific valves and fans that affected that specific crop as well as the sensors collecting information from the different types of crops. This information could be controlled by means of digital twins (Purcell and Neubauer, 2023) and follow-up and control of their values according to the life cycle of the crop, following the model of the digital twin.

4.2 Modeling of crops management by workflows

Different varieties of crops require different care, not only in the quantity of water, airflow, fertilizer, or soil type but also depending

4 "What Is Vertical Farming? Everything You Should Know About This Innovation," https://www.edengreen.com/blog-collection/what-is-vertical-farming, Edengreen Technology, January 2023.

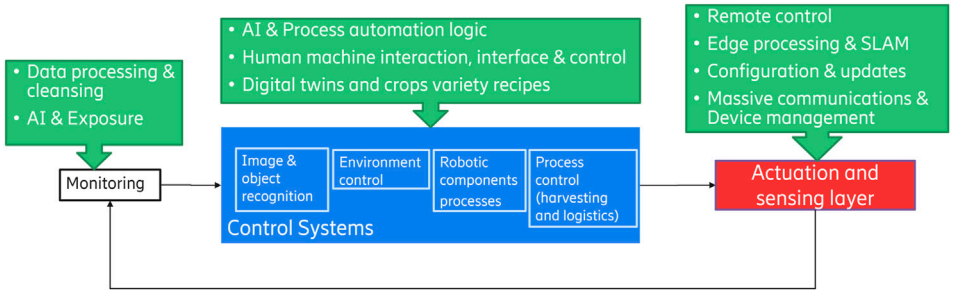

FIGURE 5
Vertical agriculture control systems.

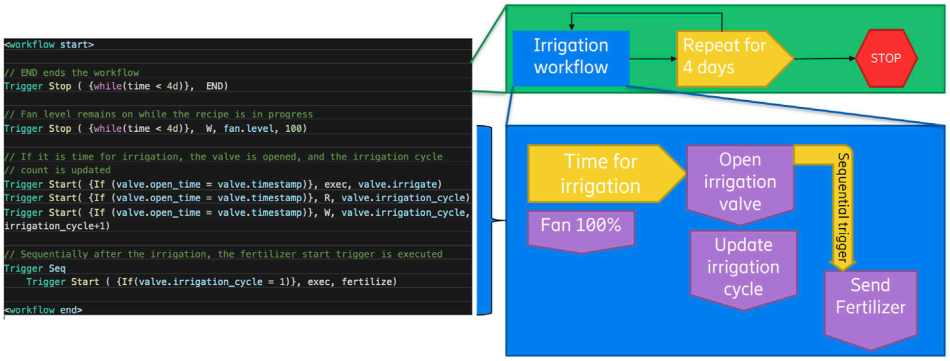

FIGURE 6
Vertical agriculture workflow template example.

on the time in their life cycle. Different treatment is thus needed in different stages of their growth. These models can be also customized according to growing management wishes for optimization; they could lead to proprietary growing recipes according to the entity. As an example of particular models (or growing recipes), take two crops: radish and sunflower. These recipes are optimized for mass growth for maximum yield of edible greens.

1. *Radish recipe*:
   - Growing cycle: 4 days.
   - Irrigation: 3 L/2 days, first irrigation at the beginning of cycle.
   - Fertilizer: First irrigation without, second with nutrients.
   - Airflow: Full ventilation whole cycle.
2. *Sunflower recipe*:
   - Growing cycle: 4 days.
   - Irrigation: 3.5 L/2 days, first irrigation at the beginning of cycle.
   - Fertilizer: First irrigation without, second with nutrients.
   - Airflow: First 2 days, no ventilation; last 2 days full ventilation.

These recipes are then translated into two different workflow templates. The templates are then instantiated into the actual equipment mapping the management of those crops. Figure 6 shows the example for radishes.

The implementation could handle these vegetable recipes since the system exposed all the needed functions, and they were fully connected and remotely accessible through the LwM2M management server. The devices required customized object templates, but their implementation was quite trivial. The mapping of the crop location to which the devices were operating for a specific crop was more complicated. Additional metadata needed to be formulated to separate the resources per crop batch. This is part of the semantic description of the device exposure capabilities, which means that the description is also not fully static since the same devices can be used for different crops and the crop's life-cycle stage also must be considered. To also automate the device assignment and documentation through the metadata, use of the digital twins of the infrastructure plus the digital twin of the crops (Purcell and Neubauer, 2023) is a very feasible way to maintain the right state and device assignments.

One major benefit of using the approach described in this application use case is that there is no need for coding or extra equipment, such as services or appliances, to make the application work. If the recipes or crops change, then only a configuration adjustment is necessary to repurpose the equipment and accommodate the new processes. Moreover, if new equipment is added to the infrastructure, it can be easily adapted to the workflows using templates and exposed functions, which may only require changes to the workflow description.

5 Discussion and conclusion

Automation and control systems are mostly integrated and managed by design. All possible functionalities, services, and devices are thus known beforehand, and very little variation is expected. Today's systems evolve quite rapidly, as do their characteristics, capabilities, and services, their adaptability to the environment, and their requirements for executing their functions (data sources, sensor characteristics, output formats, *etc.*). Therefore, systems integration by design is not fully evolvable, and the integration of new components, nomadic devices or services, or connection to new services and interfaces has become challenging. The standardization of interfaces, data models, and APIs has eased integration difficulties, but their use has also made it part of the static design of processes and system components.

The orchestration of intelligence enables the possibility of process dynamism where the processes can be changed and assigned to different components to optimize the results or adapt to the local characteristics of the components available to execute the processes. The architecture suggested by Ramos et al. (2022) enables the handling of heterogeneous platforms and systems, even on several separated operative domains. One of the main components required to realize this vision is the possibility of describing the processes and their mapping to operations (functions and services) that are executed in specific components (devices, platforms, networks). This description concept is composed of the design of workflows using a simple input–output pipelining of functions. Another aspect is how such a description can be instantiated and deployed by each of the heterogeneous components. The analysis of the deployment of such workflows helps our understanding of the requirements of a practical implementation. The set of instructions required and their mapping to the implementation when considering a variety of use cases and tested with a vertical agriculture application allow us to understand and realize the potential challenges and issues that need to be addressed to enable the intelligence orchestration vision.

Some additional considerations for further research are the seamless integration of data management solutions to the framework (storing, streaming, exposing, *etc.*) considering the addition of an instruction set to describe the management of the data in a particular execution environment, in a whole system, or even between separated platforms and domains. The descriptive approach may need implementation in place of similar primitives as the one discussed for execution which would be instantiated by the deployment engines. A wider and deeper study of the requirements and data-intensive use cases will help outline such a set of instructions and recommend an implementation framework.

In addition, the networking aspects related to mobility and topology, and network reachability, availability, latency, and reliability should be considered in some use cases. They could be modeled as a service exposure if there is the possibility of interacting with the network to signal a required quality of service or expose network observed characteristics (such as load and latency); they could then be accessed as a service that can be used in the workflow orchestration. Furthermore, the possibility of discovering, joining, or departing a network could be another set of services that a device and a network interface may expose.

The proposed description of the workflows and the intelligence orchestration architecture are designed to accommodate any type of automation process, regardless of their domain. The results and attributes observed during the development of this work suggest that this approach is both feasible and effective. Moreover, any practical limitations or challenges encountered can be addressed by utilizing state-of-the-art solutions, standardization, and additional developments based on known solutions. In conclusion, a comprehensive implementation of this concept should be able to realize the vision of intelligence orchestration for any domain.

Data availability statement

The original contributions presented in the study are included in the article; further inquiries can be directed to the corresponding authors.

Author contributions

ER is the main contributor to the manuscript and helped finishing its main body. SA helped finish Section 1.1. All authors contributed to the article and approved the submitted version.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors, and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

Alliance, O. M. (2014). *Lightweightm2m technical specification v1. 0.* Tech. rep.

Arsénio, A., Serra, H., Francisco, R., Nabais, F., Andrade, J., and Serrano, E. (2014). "Internet of intelligent things: bringing artificial intelligence into things and communication networks," in *Inter-cooperative collective intelligence: Techniques and applications*, 1–37.

Birman, K. (2007). The promise, and limitations, of gossip protocols. *SIGOPS Oper. Syst. Rev.* 41, 8–13. doi:10.1145/1317379.1317382

Casale, G., Artač, M., Van Den Heuvel, W.-J., van Hoorn, A., Jakovits, P., Leymann, F., et al. (2020). Radon: rational decomposition and orchestration for serverless computing. *SICS Software-Intensive Cyber-Physical Syst.* 35, 77–87. doi:10.1007/s00450-019-00413-w

Dalla Palma, S., Catolino, G., Di Nucci, D., Tamburri, D. A., and van den Heuvel, W.-J. (2023). Go serverless with radon! a practical devops experience report. *IEEE Softw.* 40, 80–89. doi:10.1109/MS.2022.3170153

Galletta, A., Taheri, J., Fazio, M., Celesti, A., and Villari, M. (2021). "Overcoming security limitations of secret share techniques: the nested secret share," in *2021 IEEE 20th international conference on trust, security and privacy in computing and communications* (TrustCom), 289–296. doi:10.1109/TrustCom53373.2021.00054

Galletta, A., Taheri, J., and Villari, M. (2019). "On the applicability of secret share algorithms for saving data on iot, edge and cloud devices," in *2019 international conference on Internet of things (iThings) and IEEE green computing and communications (GreenCom) and IEEE cyber, physical and social computing (CPSCom) and IEEE smart data (SmartData)*, 14–21. doi:10.1109/iThings/GreenCom/CPSCom/SmartData.2019.00026

Jayagopan, M., and Saseendran, A. (2022). *Intelligence orchestration in IoT and cyber-physical systems*. Master's thesis. School of Information Technology.

Koster, M., and Bormann, C. (2023). *Semantic definition format (sdf) for data and interactions of things. Working Draft, Internet Engineering Task Force*. *Internet-Draft* 1.

Purcell, W., and Neubauer, T. (2023). Digital twins in agriculture: A state-of-the-art review. *Smart Agric. Technol.* 3, 100094. doi:10.1016/j.atech.2022.100094

Ramos, E. J., Montpetit, M.-J., Skarmeta, A. F., Boussard, M., Angelakis, V., and Kutscher, D. (2022). "Architecture framework for intelligence orchestration in aiot and iot," in *2022 international conference on smart applications, communications and networking (SmartNets)*, 01–04. doi:10.1109/SmartNets55823.2022.9994029

Ramos, E., Schneider, T., Montpetit, M.-J., and De Meester, B. (2020). "Semantic descriptor for intelligence services," in *2020 international conferences on Internet of things (iThings) and IEEE green computing and communications (GreenCom) and IEEE cyber, physical and social computing (CPSCom) and IEEE smart data (SmartData) and IEEE congress on cybermatics (cybermatics)*, 45–53. doi:10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics50389.2020.00027

Sicari, C., Carnevale, L., Galletta, A., and Villari, M. (2022). "Openwolf: A serverless workflow engine for native cloud-edge continuum," in *2022 IEEE intl conf on dependable, autonomic and secure computing, intl conf on pervasive intelligence and computing, intl conf on cloud and big data computing, intl conf on cyber science and technology congress (DASC/PiCom/CBDCom/CyberSciTech)*, 1–8. doi:10.1109/DASC/PiCom/CBDCom/Cy55231.2022.9927926

Wurster, M., Breitenbücher, U., Brogi, A., Falazi, G., Harzenetter, L., Leymann, F., et al. (2020b). "The edmm modeling and transformation system," in *Service-oriented computing – ICSOC 2019 workshops*. Editors S. Yangui, A. Bouguettaya, X. Xue, N. Faci, W. Gaaloul, Q. Yu, et al. (Cham: Springer International Publishing), 294–298.

Wurster, M., Breitenbücher, U., Harzenetter, L., Leymann, F., Soldani, J., and Yussupov, V. (2020a). *TOSCA light: Bridging the gap between the TOSCA specification and production-ready deployment technologies*.