Journal of Industrial Information Integration 41 (2024) 100668

Contents lists available at ScienceDirect

Journal of Industrial Information Integration

journal homepage: www.elsevier.com/locate/jii

Full length article

Model-based tool for the design, configuration and deployment of data-intensive applications in hybrid environments: An Industry 4.0 case study

Ricardo Dintén *, Patricia López Martínez, Marta Zorrilla

Software Engineering and Real-Time group, Universidad de Cantabria, Avda. de los Castros s/n, Santander, Spain

ARTICLE INFO

Dataset link: https://github.com/istr-uc/RAI4DeploymentTool

Keywords:
Model-based
Digital platform
Containerisation
Service-oriented
Cloud/edge computing
Internet of things

ABSTRACT

The fourth industrial revolution advocates the reformulation of industrial processes to achieve the end-to-end (provider-customer) digitalisation of the industrial sector. As is well known, the industrial environment is very complex, where legacy systems must interoperate and integrate with modern devices and sensors. Communication among them requires specific and costly developments, so architectures based on data sharing and services implementation are considered one of the most flexible and appropriate technological solutions to gradually achieve the desired horizontal and vertical integration of the value chain. The design and deployment of data-intensive applications is not straightforward, therefore this paper proposes a model-based tool to characterise the different elements to be configured in an application and to make its deployment easier by generating configuration, orchestration and deployment files and sending them to the corresponding nodes for their execution. In few words, this article highlights the advantages of distributed and data-centric architectures to face the challenge of integration and interoperability in data-intensive complex systems and presents the extension of the RAI4 metamodel proposed in Martínez et al. (2021) that now allows specifying how, containerised or not, and where, on the cloud, fog, edge or on-premise, each service can be hosted according to its functional and non-functional requirements, mainly issues related with real-time, security and cyber physical hardware dependencies. For the sake of comprehension, a pseudo-real use case addressed to pre-process and store pollution data from environmental sensors installed in a smart city is described in detail, including different deployment settings.

1. Introduction

Industries are currently undergoing a process of digital transformation fostered by Industry 4.0. The term Industry 4.0 (or I4.0), "Smart Factory" or "Industrial Internet" [1] emerged at the Hannover Fair (Germany) in 2013, with the aim of promoting a new strategy for organising and controlling industrial production processes based on the information generated in the life cycle of its products. Briefly, Industry 4.0 advocates a shift in manufacturing processes and organisations towards a more flexible, intelligent, predictive and controlled environment driven by data. According to [2], "for I4.0 to come true, it is essential to implement the horizontal integration of inter-corporation value network, the end-to-end integration of engineering value chain, and the vertical integration of factory inside". This leads to the need to strategically design and plan the deployment of a digital platform that allows the gradual implementation of fully digitised services and processes [3].

To reach this objective, Germany published RAMI [4], a unified architectural reference model that provides a collective understanding for I4.0 standards. It can be regarded as a tool to map I4.0 concepts and use cases. On the other hand, the Industrial Internet Consortium (IIRA) [5] enables Industrial Internet of Things (IIoT) system architects to design their own systems using a common vocabulary and a standard-based architecture framework and reference architecture. At the same time other organisations have proposed their reference guide such as NIST or MIIT from China [6]. In particular, our proposal is based on the layered databus architecture pattern proposed in IIRA as central element of the architecture, which is modelled along with other components needed to build data-intensive applications that meet the main requirements of Industry 4.0 scenarios such as internet of things, cyber–physical systems and smart factory. These requirements according to [7] are interoperability, virtualisation, decentralisation, real-time capability, service orientation and modularity, which were extended and refined in [8]. Furthermore, it is also of utmost importance

* Corresponding author.
E-mail addresses: ricardo.dinten@unican.es (R. Dintén), lopezpa@unican.es (P. López Martínez), marta.zorrilla@unican.es (M. Zorrilla).

https://doi.org/10.1016/j.jii.2024.100668
Received 8 June 2023; Received in revised form 20 June 2024; Accepted 25 July 2024
Available online 3 August 2024

that the architecture allows the integration of legacy systems present in industrial environments and enables their interoperability with the latest generation devices in a transparent manner. This is achieved by using a data-centric publish–subscribe communications model where applications on the databus simply "subscribe" to data they need and "publish" information they produce [9].

The design, development and deployment of modern data-intensive applications is a complex task mainly due to the great number of different technologies that must be orchestrated, configured and deployed in order to fulfil their functionality [10]. Among others, scheduling, distribution, communication, persistence and security services to name the most relevant ones. Furthermore, most times these applications must be deployed on a hybrid (cloud/edge/on-premise) environment [11] with the aim of meeting latency requirements, managing huge volume of data and having enough computing capacity. This means that design and, above all, deployment decisions are becoming more complex due to the heterogeneity of the devices, the diversity of execution environments, real-time constraints and security and legal requirements that involve dynamic reconfiguration changes. Therefore, it is highly recommended to work with software tools that facilitate their conception, setting-up as well as their deployment and monitoring following the new DevOps culture [12] where software developers collaborate with IT operators. At this regard, our proposal addresses the design supported on a reference architecture (RA) and, as pointed out by Ataei et al. [13], RAs are an effective artefact for addressing the development of complex big data systems that provide software engineers with knowledge about patterns designed to solve a class of problems that occur repeatedly. Furthermore, it increases the frequency, quality and speed of software delivery using automated procedures.

In [14], the authors described RAI4, a data-centric reference architecture focused on Industry 4.0 that allows us to conceive and specify all the elements involved in a data-intensive application: data sources, physical and virtual resources, workflows and monitoring elements according to which perform the scaling and self-tuning. This architecture is supported by a metamodel that contains a detailed information about how all these elements are conceived and related. In this paper, we extend this metamodel to add the possibility of deploying applications on hybrid environments as well as of specifying and building partially or totally containerised applications. Containerisation is essential in this type of applications as it is the most efficient way to deploy software solutions [15] whilst facilitating portability and reuse in different infrastructures.

Oppenheimer et al. [16] pointed out that IT teams lack tools that help them build the configuration files to orchestrate the set of containers that comprise a complex application. This fact is even more challenging if the solution must combine the deployment of containerised services with others that are not. Nowadays manual deployment is the only possibility, which is prone to errors and hard to repeat. To overcome this limitation, together with the architecture metamodel we provide a tool for the deployment of the digital platform (services) and its applications. To show the usefulness and validity of our metamodel to design data-intensive applications as well as to deploy them in hybrid environments by means of our deployment tool, we describe a use case aimed at pre-processing and storing pollution data from environmental sensors installed in a smart city. Likewise, we list other use cases developed and available in our online repository.

The paper is organised as follows. Section 2 relates architectures proposed in the literature for the design and deployment of data intensive applications and discusses the importance of containerisation in this context. Section 3 summarises our RAI4 metamodel, which models a reference architecture under a highly flexible data centric architecture for Industry 4.0 based on a set of middleware services that provide management and monitoring capabilities to the platform. Section 4 explains the metamodel extension proposed in this paper for supporting containerisation and describes and justifies the changes performed. Section 5 presents the case study and explains the process of building a model, compliant to the metamodel, of an IoT data-intensive application deployed in a hybrid environment. In addition, deployment variations of this model are described with the aim of showing the reutilisation capacity of our tool and the associated costs. Section 6 discusses the advantages of adopting our RAI4 metamodel and tool and points out issues still open. Finally, the last section summarises our conclusions and our future research lines.

2. Related work

This section is organised in two subsections. The first relates the proposed architectures in the literature designed for the instantiation of big data platforms and, in particular, those that are focused on facilitating the design and deployment of containerised or non-containerised applications in hybrid environments. Next, the containerisation technology and its relevance to our proposal is commented as well as the importance of the use of orchestration technologies.

2.1. Reference architectures for big data-intensive applications

First of all it is convenient to clarify the term reference architecture (RA) since this is profusely used in literature. In general, it is understood as a collection of related elements that perform certain task or set of tasks. The degree of detail, the perspective or the mechanisms under which each module is described mark the difference among them. The conceptualisation provided by reference architectures not only helps to identify the key components of the system, their function and interrelationship with each other, but also facilitates their maintainability and scalability. Furthermore, it serves as a communication medium or language that provides stakeholders with unified elements and symbols to discuss and develop big data projects.

Big data intensive applications can be characterised in addition to the three V's (velocity, variety and volume of data) by requiring according to [13]: (i) distributed scaling to address batch and streaming processing demand, (ii) the need for real near-time performance (stream processing), (iii) the complex technology orchestration to create effective communication channels between components and data flows, (iv) metadata to manage quality, security and privacy issues and (v) a continuous delivery to continually disseminate patterns and insights into various business domains (DataOps). In our particular context, IoT/I4.0, it is also desirable that RA conforms to industry standards such as IIRA or RAMI. Furthermore, for practical reasons, the provision of tools for pipeline process automation (model to code), especially in complex environments, should be included [16]. For all these mentioned premises and literature studied, the authors state that the features that RA for big data intensive applications should fulfil are:

- (R1) standard-based adopted by industry and academia
- (R2) distributed, scalable and data-centred
- (R3) support for metadata
- (R4) measurable performance
- (R5) formally described by means of a standard language or model
- (R6) deployment tool available

In the literature we find several architectural proposals for data-intensive applications but based on the pipeline specification as a set of ad-hoc blocks, without using an abstraction layer that makes the solution reusable, scalable, domain-independent (general) and technology-agnostic. Some examples are the architecture proposed by [17] for the integration, processing and analysis of heterogeneous and unstructured data from water supply networks supported by a CEP engine; the one published by Carcillo et al. [18] for detecting credit card fraud using streaming analysis with Kafka and Spark or the one defined by [19] for predictive maintenance use cases applied for the railway transportation industry and the wind turbines energy industry. Others such as the

one proposed by the CAPRI project [20] that aims to innovate the process industry, with a modular and scalable reference architecture, based on open source software and deployable both in brownfield and greenfield scenarios, being general purpose, it lacks a model that helps software architects to describe, maintain and deploy their big data platform. Unlike the previously mentioned works, this one is based on industry standards (RAMI, IIRA) and incorporates security issues. In the same line, Neomycelia [21], is a RA that provides a business domain-independent event driven microservices architecture comprised of 14 components. Among them monitoring and metadata modules are included as well as a bus backbone and batch and streaming processing. They mention that its RA follows ISO/IEC 42010 for architectural descriptions, however it cannot be checked in the paper. Nor does it offer a deployment tool.

Prior to the publication of our proposal, Ataei et al. [13] performed a systematic literature review about big data reference architectures. They analysed 22 reference architectures (RA) and their findings yielded the fact that the majority of the big data RAs analysed were running underlying some sort of monolithic data pipeline following kappa or lambda architecture with a central storage unit and issues such as privacy and security did not seem to have been discussed enough. Metadata management is also lacked in most RA, except Neomycelia and Bolster [22]. The latter refines the lambda architecture to create a software reference architecture for semantic-aware big data systems.

Until now, none of previously RA mentioned relies on a model-based approach. This limits developers to reason, reuse and automate, by means of translation rules, the setting, deployment and reconfiguration of their models. At this regard, we find Margara et al.'s work [23] that presents a unifying model that dissects data-intensive systems into a collection of abstract components that cooperate to offer the system functionalities. This is very useful for software engineers that need to deeply understand the range of possibilities to select the best systems for their application scenario and once selected, could use our metamodel to design and deploy their digital platform.

Another interesting model-driven proposal is the one published in [24]. They built a UML profile conceived at three abstraction levels to assist data-intensive applications developers in three facets. The first level allows designers to define the main architecture of an application; the second level includes the specifics of the technology used to develop the application, e.g., Apache Storm; and, the third level allows them to automate the deployment of the application and the frameworks it exploits in the specific cloud environments that designers have selected. Unlike RAI4.0, they include a module in the second level that, based on certain parameters such as number of users, average task execution time, number of cores and degree of parallelism, etc., provides an estimated performance model. In contrast, RAI4.0 includes a monitoring module that allows to measure in real time the parameters of interest (CPU usage, memory, latencies, etc.) of the deployed services and based on them, DevOps teams can reconfigure the platform by adding or reducing nodes or hosting the services on the cloud, fog or edge according to the functional requirements of the application. Both proposals include virtualisation, feature that none of previously mentioned works collected and, furthermore, RAI4.0 enables the deployment in hybrid environments and not only in the cloud. [24] does not model metadata.

Table 1 summarises the features of the reviewed works according to the requirements previously specified.

As far as we know, there are no other proposal of model-based reference architecture that addresses the design and deployment of a data centred big data platform on which a set of global, heterogeneous and decentralised services, virtualised or not, are orchestrated. Besides, it takes into account security and metadata. Security is enabled by means of an access control list (ACL) of data bus publishers/subscribers using the TLS protocol and data confidentiality is preserved by means of encryption. Data integrity is kept thanks to event management system whose features guarantee that events are immutable. Finally, data privacy must be controlled ad-hoc using metadata registered. In addition, data is governed by metadata description, which is essential for the platform to operate in a decentralised and autonomous way. The metadata is responsible for collecting the nature, semantics and quality of the data required by the agents that process it; the estimation of the volume, velocity and variety of data to be managed, the metrics to be measured in order to reconfigure resources dynamically and the security requirements (authentication, integrity, confidentiality and availability). Due to this last issue is highly important in these complex environments, we rely on ontologies such as Onto-Carmen [25], an Ontology-driven approach for Cyber–Physical System Security Requirements meta-modelling and reasoning which helps in the early stages of development by identifying, modelling, and analysing security requirements.

Finally, the fact that RAI4.0 has a deployment tool available makes this RA more useable and cost-effective after a short training.

Table 1
Big data reference architecture comparison.

Ref.	R1	R2	R3	R4	R5	R6
[17]	N	Y	N	N	N	N
[18]	N	Y	N	N	N	N
[19]	N	Y	N	N	N	N
[20]	Y	Y	Y	N	N	N
[21]	Y	Y	Y	Y	Y	N
[22]	Y	Y	Y	N	N	N
[24]	Y	Y	N	Y	Y	Y
RAI4.0	Y	Y	Y	Y	Y	Y

2.2. Containerisation and orchestration

Container technology has acquired great relevance in recent years, in fact has emerged as a new paradigm to address intensive scientific applications problems. Thanks to the containers, developers can create predictable environments isolated from other applications and run on a shared operating system as a process, which can be moved quickly and reliably from one computing environment to another. Not only containerisation makes deployment easy in a reasonable amount of time and with few required computational resources [26], but also application performance is not significantly penalised as demonstrated in [27] by running an IoT application inside a Linux container, rather than directly on the device's host operating system. Therefore, as [15] pointed out, nowadays the most efficient way to deploy a software solution on the cloud or edge is by means of container-based virtualisation.

The main reasons for the adoption of containerisation can be summarised in: (i) the time to start to operate containers is practically instantaneous, better than virtual machines; (ii) containers show much better scalability than virtual machines; and, (iii) with the same workload, containers achieve higher CPU and memory utilisation [28].

Nowadays there are different container technologies, being Docker [29], an open source project that automates the faster deployment of Linux applications, one of the most adopted [30]. But Docker is not the only container system option, CoreOS provides a streamlined alternative to Docker called CoreOS rkt [31] and Canonical offers the LXC containerisation engine [32] for Ubuntu to name a few [33].

In complex systems, the volume of containers is huge and it is therefore desirable to use orchestrators for the deployment, management and scaling of applications, facilitating portability between infrastructure providers [34]. Kubernetes [35], Docker Swarm [36] or more recently Hasicorp Nomad [37] are some examples.

Kubernetes and Nomad support similar core use cases for application deployment and management. However, Kubernetes performs better in use cases addressed to containerised big data applications deployed in the cloud. Furthermore, Kubernetes provides all the features needed to run Linux container-based applications including cluster management, scheduling, service discovery, monitoring, etc. whereas Nomad only aims to focus on cluster management and scheduling using

other tools like Consul for service discovery/service mesh and Vault for secret management. On the other hand, Nomad is architecturally much simpler, a single binary, thus lighter and more general purpose (not focused only on Linux containers). Regarding Swarm, it is lightweight and more beginner-friendly that Kubernetes but it only manages Docker containers. Those researchers interested in testing these technologies can use COFFEE, a recent tool [38] for Benchmarking of Container Orchestration Frameworks.

Due to the irruption of the container technologies, Pahl et al. [10] carried out a survey with the aim of identifying, taxonomically classifying and systematically comparing the existing research body on containers and their orchestration and specifically the application of this technology in the cloud. They concluded that there was a need for research beyond the performance and isolation concerns for container virtualisation, particularly regarding methodological and tool support. At this regard, and with the aim of helping researchers and technologists to compare deployment automation technologies, Wurster et al. [39] proposed the Essential Deployment Metamodel (EDMM) which provides a common denominator of the features of the most important deployment technologies and maps these to native constructs of each technology. Therefore, its purpose is different from ours, although it can help DevOps team in the definition of the configuration file for the image orchestrator to be used in the digital platform and specify it for deployment using our metamodel.

Following with proposals based on models related to containerisation, Paraiso et al. [40] employs Model-Driven Engineering (MDE) techniques, in order to handle and analyse Docker containers at a higher level of abstraction. Their model collects the structure of containers, the relationships among them, and the hosts in which they are deployed but omits information on services and workflows describing the application. As the previous reference, this could be used to generate, validate and update Docker configuration files, if this virtualisation tool was the one selected for the digital platform. One aspect to be taken into account is it is linked to OCCI API that conditions the use of its infrastructure to previously provision machines, unlike our metamodel, which is technology agnostic.

Finally, SMADA-Fog [41] is a complete solution that enables automated deployment and service adaptivity in container-based applications executed within the fog computing environment. This includes models, notations, algorithms and tools for this purpose. It thus focuses on deployment stage, leaving out of scope the description of the workflows and data sources that make up the system. In this regard, it provides a semantic module as well as optimisation algorithms that allow the reconfiguration of the parameters specified in the deployment, thus enabling dynamically the improvement of the performance of the applications. This last option of reconfiguration is lacked in our proposal.

3. RAI4 reference architecture

RAI4 is a general reference architecture for the definition and deployment of industrial applications. This is supported on a data-centric architecture and on a set of middleware services that provide management and monitoring capabilities to the platform. A compliant RAI4 metamodel has been defined to describe the targeted applications. Models compliant to the RAI4 metamodel serve two purposes in two different phases of the development process:

- They help designers to conceive their data-intensive digital platforms, providing both all the modelling artifacts required to describe the elements involved in such kind of systems (data, resources, applications and monitoring metrics) and a common way of organising these elements.
- They gather all the information required to configure, deploy and execute the applications on the platform. In this sense, RAI4 models are used as input of the tool RAI4DeploymentTool, which automates the generation, transfer and execution of the set of configuration files and scripts that are required to deploy and launch the applications described in the model.

The main elements defined in the RAI4 metamodel to support both the design and the deployment of data-intensive applications are explained in the following subsections.

3.1. Design of a RAI4 data-intensive application

As it is shown in Fig. 1, each model compliant to the RAI4 metamodel is organised around a root element, instance of the `ComputationalSystem` class. This acts as container of the three sections that comprise the model:

- The platform model, made up of the set of physical and virtual resources and the middleware services that comprise the digital platform. All of them modelled through classes that inherit from the `PlatformResource` abstract class.
- The workload model, constituted by the set of data streams (topics) that flow in the environment, modelled through instances of the `WorkloadStreamData` class, along with the workflows (set of processing tasks) that produce and/or consume these data streams (`Workflow` class).
- The monitoring model, i.e. the set of metrics that must be measured at runtime. The root class for the hierarchy of possible metrics is `Metric`.

The platform model is the section most affected by the extension proposed in this paper, so a more detailed explanation is given in what follows. Fig. 2 depicts the classes that inherit from `PlatformResource` and hence, are used to describe the different types of resources available in the platform. `ProcessingNode` represents the processing nodes that participate in the system, either physical nodes at dew or edge computing level [42], modelled as instances of the `PhysicalProcessingNode` class, or virtual resources hired on the cloud, represented as instances of the `VirtualProcessingNode` class. `ProcessingNode` identifies each node with its IP address and describes some common properties such as the number of processors, or the external IP address in case of virtual nodes. `PlatformResource` instances can be grouped in `ResourceCluster` elements, if needed.

`PlatformServer` is an abstract root class that gathers generic information about the configuration, deployment and identification of the brokers that provide access to the distributed services available in the platform. As shown in Fig. 2 this class is, in turn, specialised in seven abstract classes, one for each type of service included in the architecture: `SerializationServer`, `SecurityServer`, `CommunicationServer`, `DistributionServer`, `SchedulingServer`, `MonitoringServer` and `PersistenceServer`.

As said before, these classes are described at abstract level, so they must be specialised when a certain set of specific technologies are chosen for their implementation. In our case, a platform almost completely based on Apache technologies is used, so the following classes were added to the metamodel: a Kafka-based communication service (`KafkaServer`), a Zookeeper-based distribution server (`ZookeeperServer`), an Apache Storm scheduling server (`StormServer`) and a Cassandra persistence server (`CassandraServer`). Likewise, a Prometheus-based monitoring server (`PrometheusServer`) was added. Two of them are shown in detail in Fig. 2, with each class collecting the attributes with the specific information required for the configuration of the corresponding services. These services usually have a great amount of configuration properties which in most cases, are used with their default values. For simplifying the task of the designers, these default values are provided

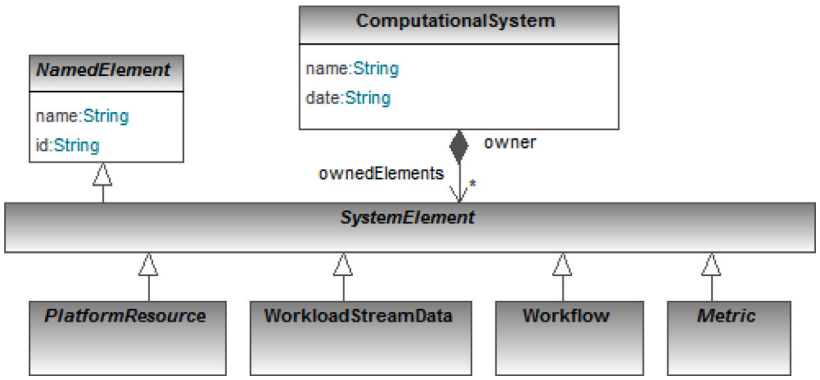

Fig. 1. Root elements of the RAI4 metamodel.

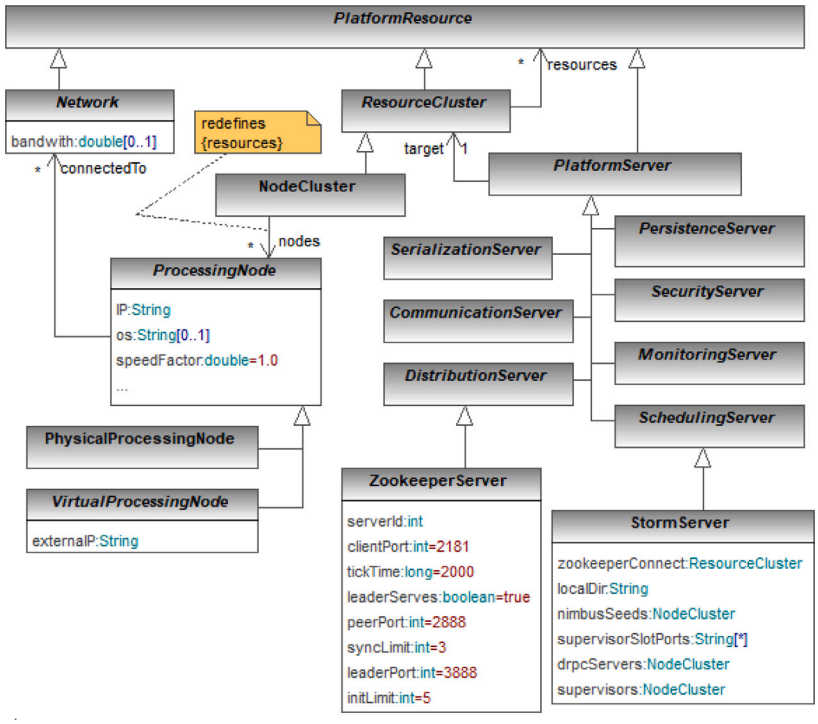

Fig. 2. Platform resources and middleware services (Platform Model).

by the metamodel, so they only need to assign those properties that require a different value for the modelled scenario.

Although not directly affected by the extension, it is interesting to give some ideas about how the metamodel describes the workload model, especially for a better understanding of the use case exposed in Section 5. The rest of the metamodel, e.g. the monitoring section can be consulted in [14].

Two elements contribute to the definition of the workload of a system: the set of topics that flows through the environment and the set of processing workflows that produces/consumes/transforms these topics. These workflows constitute the applications that provide the functionality in the environment. They are conceived as sets of processing tasks that operate following a reactive strategy, in response to the occurrence of a specific pattern of instances coming from one or more topics, which triggers the execution of the root task of the workflow. The rest of tasks that constitute the workflow are chained following an acyclic graph.

The metamodel describes topics as instances of the `WorkloadStreamData` class while the processing tasks that exploit the topics are organised in `Workflow` elements, whose main structure is shown in Fig. 3. The execution of a workflow is triggered in response to the occurrence of a topic (`rootTask.trigger`). Next, a set of processing tasks (`ownedTasks`) related by the control flow are executed. The task pointed by the `rootTask` attribute represents the one that starts the workflow execution. The control flow among the tasks is implemented by means of a set of private topics (`ownedStreamData`) defined by the workflow, which are instances of `WorkflowStreamData`, a special type of topic. When a task finishes its execution, a data flow (`returnedTopic`) is returned which, in turn, can trigger the execution of other task(s) of the workflow. The scheduling service responsible for the assignment of resources and for launching the execution of the tasks is referenced through the `scheduler` attribute.

3.2. Deployment of a RAI4 data-intensive application

The complexity that involves the configuration, deployment and execution of an initial RAI4 digital platform as well as its reconfiguration and the installation/uninstallation of the workflows were the main drivers that boosted the development of the RAI4DeploymentTool. This tool processes the model of the system (compliant to the RAI4 Metamodel) and, as a first step, generates all the required configuration files

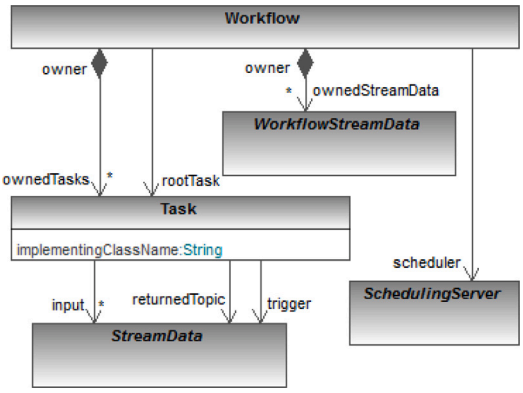

Fig. 3. Workflow and topics (excerpt of the Workload Model).

and launch scripts for the deployable components (services, workflows, topics, etc.). Then, the tool sends these files to their corresponding physical or virtual processing nodes using SCP commands, and finally, it executes remotely the scripts that launch the execution of the components on the nodes using SSH. This last step is order-concerned, taking all possible dependencies among components into account, so that the RAI4DeploymentTool can be executed from a console opened in one of the nodes of the platform or from any other node connected to the platform through the network.

Fig. 4 summarises the main elements defined in the metamodel to support the deployment tool. As can be observed, the root class of the model, `ComputationalSystem`, defines a method called `deployAndLaunch` that implements the whole deployment and launching process:

- First, this method invokes the `configureDeployment` method on each element that can be independently deployed on the system, modelled as elements that inherit from the `SystemComponent` class. Taking the configuration data assigned to each instance, this method generates all the configuration files and scripts required to launch the component in a node and adds them to the `ProcessingNode` element that represents the node in which they must be later instantiated and executed (in the scripts case). The behaviour of this method must be overridden for each concrete type of component.
- When all the deployable components have been processed, the `deploy` method is invoked on each `ProcessingNode`, which causes the transfer, using SCP, of all the configuration files stored in the `configFiles` attribute and the scripts corresponding to the `launchingScripts` attribute to the corresponding physical node.
- Finally, all the previously transferred scripts are remotely executed in their corresponding nodes using SSH. These will launch the execution of the corresponding components using the configuration data also transferred in the previous step. As said, the scripts are launched in a specific order, and not arbitrarily node by node, since the latter could raise some errors due to dependencies among components that are executed on different nodes.

A more detailed description of the deployment process implemented by the tool can be consulted in [14].

The current implementation of the tool and metamodel, available at git,[1] is based on Eclipse/EMF [43], hence, the RAI4 metamodel has been formalised using Ecore and the RAI4 models used as input of the deployment tool are formalised as .xmi files compliant to that metamodel, using the Sample Reflective Ecore Model editor for their edition. The deployment tool has been written using Java and Maven, so it is packaged as a .jar file. Therefore, it is triggered as a common jar application, using a RAI4 model as input argument. The only restrictions that must be met for the deployment tool to work are that the virtual nodes must be already provisioned and reachable, platform services, e.g. Kafka or Cassandra, must be previously installed, i.e. available, in the nodes of the platform, and the Kubernetes cluster must be configured and running so that the generated scripts can launch their corresponding instances on them.

[1] https://github.com/istr-uc/RAI4DeploymentTool.

4. Metamodel extension for supporting containerisation

The advantages provided by application containerisation led us to carry out a refactorization of the RAI4 metamodel. The changes were addressed to allow some platform servers to be deployed on premise or cloud nodes directly, while others could be deployed by means of container orchestration services, such as Kubernetes, Docker Swarm or Nomad.

Consequently, the platform model has been significantly changed, as it can be observed in Fig. 5. The first change consists in the definition of the new `ProcessingResourceCluster` class, which inherits from `PlatformResource`. This is an abstract class used to model a resource with processing capacity, i.e. a resource in which platform services or workflows can be executed. More specifically, it models a cluster of this kind of processing resources. Two classes inherit from it: (i) `ProcessingNodeCluster`, which redefines the old `NodeCluster` class, and it represents a cluster of `ProcessingNode` elements, either physical or virtual, and (ii) `OrchestrationCluster`, which represents a container-based orchestration cluster, responsible for managing the creation, scaling, communication and arranging of the container-based deployed services throughout a hidden set of nodes. Due to the inner complexity of configuring orchestration services, like Kubernetes, it is common to use managed versions of these services supported by cloud providers. In this case, it is not possible knowing in advance the specific nodes in which the services will run (in fact, their management is usually discouraged), so these nodes are not included in the model.

As shown in Fig. 6, the `OrchestrationCluster` abstract class is currently specialised in three concrete classes, `KubernetesCluster`, `SwarmCluster` and `NomadCluster`, to give support to the main current contenarisation technologies. Each one defines the properties required for configuring the deployment of platform services on them. For example, both `SwarmCluster` and `NomadCluster` define the ip and port required to access them, while Kubernetes uses a mechanism based on a file containing its access metadata (referenced by the `kubeConfigPath` attribute).

The definition of this new modelling elements is accompanied of a simple refactorisation of the previous `PlatformServer` class. As shown in Fig. 5, it has been renamed as `PlatformService`, and its semantics has been redefined so it no longer represents a single broker with a `ProcessingNode` as host but an entire service with a `ProcessingResourceCluster` as host. This allows the deployment of services either on nodes or according to container-based strategies. Moreover, this strategy reduces the number of modelling elements needed to create a model of an application, since e.g. to model a three broker Kafka cluster in the previous version, three `KafkaServer` instances were needed, each one hosted in a different `ProcessingNode`, while now it can be modelled by a single instance of `KafkaService`, hosted in a `ProcessingNodeCluster` made up of three nodes. This makes the model closer to reality, the building and understanding of models easier, and the process of refactoring simpler.

The rest of applied refactorisations, summarised in Fig. 7, are related with the configuration and deployment management. We have renamed the `SystemComponent` class to a more suitable name, `DeployableComponent`, and changed the way the information required for the deployment is included in the model. Before, all the

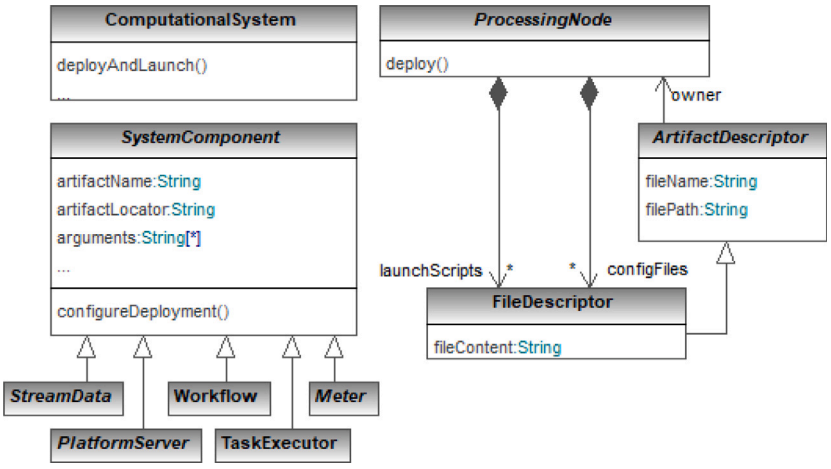

Fig. 4. Elements of the metamodel supporting the deployment tool.

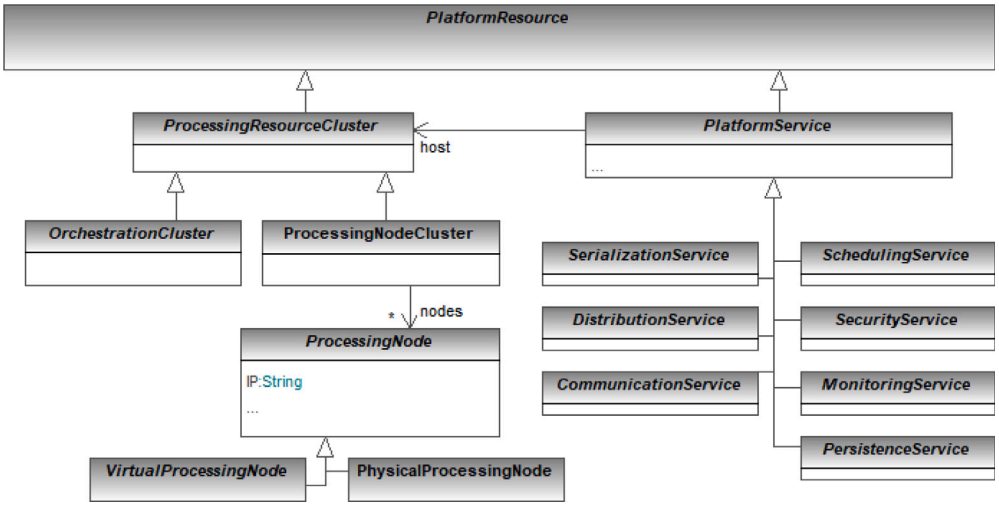

Fig. 5. Refactorization of the platform model.

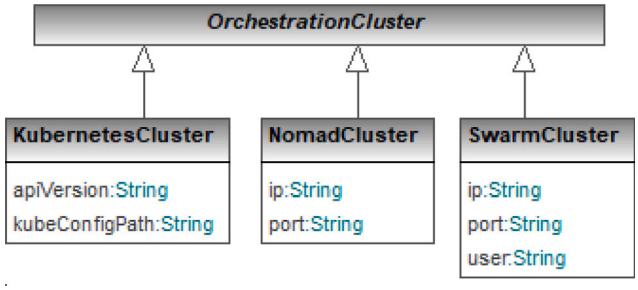

Fig. 6. OrchestrationCluster hierarchy.

information was defined in the SystemComponent itself, and hence inherited by each subclass, but now, since the deployment data depends strongly on where the component is deployed, node or container-based orchestrator, a new strategy has been defined. Each `DeployableComponent` contains an instance of a new class called `DeploymentConfiguration`.

As can be seen in Fig. 7, DeploymentConfiguration is specialised in two classes: `OrchestratorDeploymentConf` and `NodeDeploymentConf` to represent the specific configuration of each deployment mode. `NodeDeploymentConf` keeps the same old configuration parameters previously present in the SystemComponent class and aimed to configure deployment on physical or virtual nodes, whereas `OrchestratorDeploymentConf` defines a new set of parameters and classes needed to deploy services on containerisation technologies. The latter gathers the main technology-independent metadata of a containerised application, which are:

- `image` and `imageTag`: represent the registry, repository and version of the container image to use on the deployment of the service.
- `imagePullPolicy`: tells the orchestration server when to pull or download the image from the repository.
- `replicas`: sets up the desired number of replicas to be running on a certain moment.
- `command`: gathers a command line order to execute after the creation of the container.
- `volumes`: models the persistence configuration of the services deployed.
- `ports`: models the port mapping between containers and actual nodes to make services accessible from outside.
- `constraints`: this feature allows modelling restrictions to ensure that applications are deployed only on some specific nodes (via labels) or on nodes that have a minimum level of specifications (via resources).
- `env`: collects environment variables that must exist on the containers.

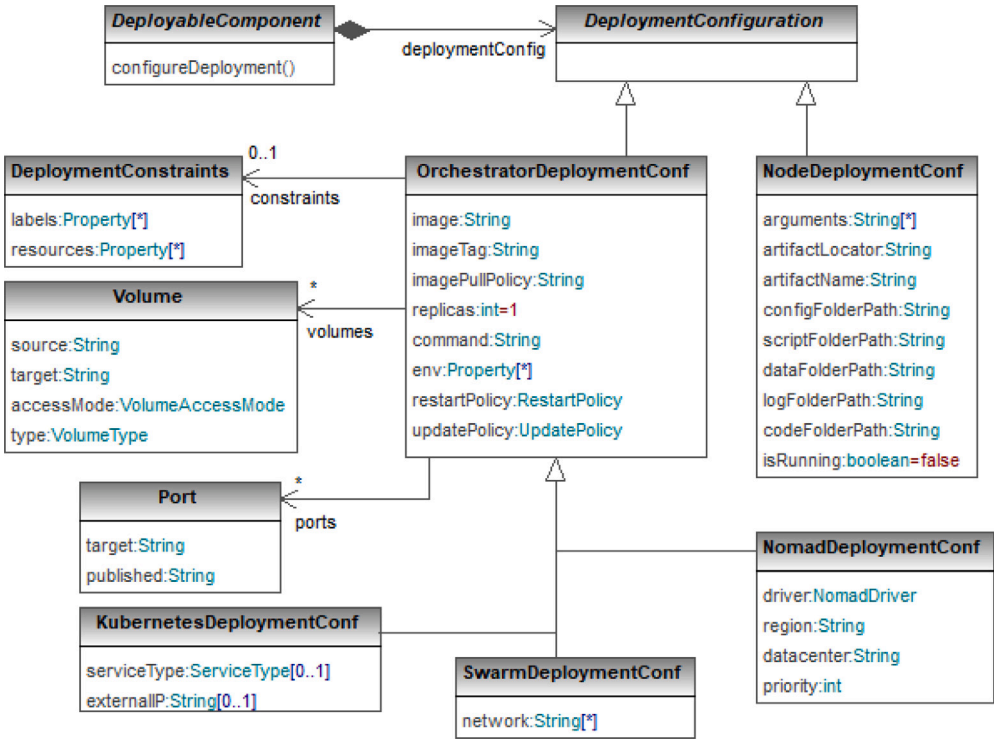

Fig. 7. Extension of RAI4 metamodel to support containerisation.

- `restartPolicy`: models the behaviour of the applications when they fail or restart.

Three classes have been defined as extension of `OrchestratorDeploymentConf`, called `KubernetesDeploymentConf`, `SwarmDeploymentConf` and `NomadDeploymentConf`,[2] to gather some properties specific for the deployment of services on each technology. As example, these are the properties added for Kubernetes:

- `externalIP`: external IP of a load balancer available on a cloud provider. This will be used to redirect traffic from outside the cluster to the Kubernetes service responsible for managing the communication of the pods.
- `serviceType`: type of Kubernetes service used to expose the services within and outside clusters. It can take the following values: LoadBalancer and ClusterIP.

Regarding the deployment process, it is important to remember that the RAI4Deployment tool generates the required information to configure and execute only those elements that inherit from `DeployableComponent`. The elements that represent the underlying processing resources, either processing nodes or orchestrators in this new version, are considered to be available for the deployment, i.e. they must have been previously configured and started for their use.

These changes make the metamodel be more general and cover a wider variety of deployment settings, as it will be demonstrated in the following sections.

5. Use case: IoT application

This section describes a case study to show the applicability of our proposal. This models a real-time application that analyses and processes pollution data from sensors installed in a smart city. This data intensive application has been designed following a kappa architecture [44] where data are collected and published in Kafka, then processed by Storm and persisted in Cassandra for being queried and, in a future, used to build prediction models [45,46].

Next, we explain the model of the application defined according to our metamodel. It must be remembered that the reference architecture organises the system model in three independent but complementary sections: the workload of the system, the platform resources available for execution, and the monitoring metrics. Most of the modelling elements used in these sections are independent of the deployment scenario, since they are related to the system design itself. So, the elaboration of the model starts with the definition of those elements.

For elaborating the workload model, the first step consists in defining and qualifying the topics that flow in the environment. In this case, there is a single topic, called `PollutionT`, which gathers the ingested pollution data. Secondly, the workflows that produce, consume or transform the topic instances must be described. The description of a workflow includes three features: (i) the reactivity, i.e., the topics that trigger its execution, the topics that it consumes and the ones generated as result of the execution; (ii) the activity, i.e., the processing tasks that are executed and the control flow among them and; (iii) the scheduling, i.e., the scheduler that defines the concurrency and multiplicity strategy applied for the execution of the tasks in the available resources and services. Fig. 8 depicts a schema of the behaviour of the `PollutionProcess` workflow, triggered by `PollutionT`. The first step of the workflow aims at filtering the data to get rid of out of range values and classifying them by region and temporal window (10 s). Next, the result of this step is stored in Cassandra. Afterwards, each environmental parameter included in the topic is averaged by region and temporal window, and finally, these mean values are also stored in Cassandra. In this case, all these functionalities are executed inside a single task, called `mainTask` in the model.

Then, as part of the definition of the platform model, we start with the description of the middleware services used in this case: Kafka is used as communication service, Zookeeper as distribution service, Storm as scheduling service and Cassandra as persistence service. So,

[2] Nomad is currently supported only at modelling level, the deployment tool still needs to be adapted.

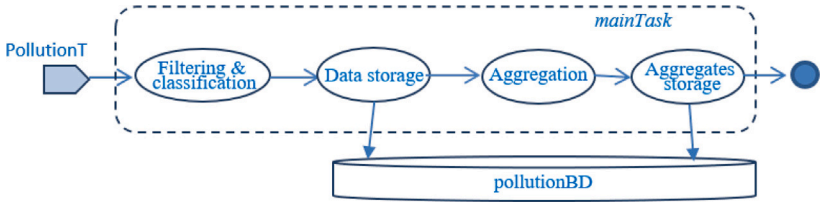

Fig. 8. Behavioural description of the PollutionProcess workflow.

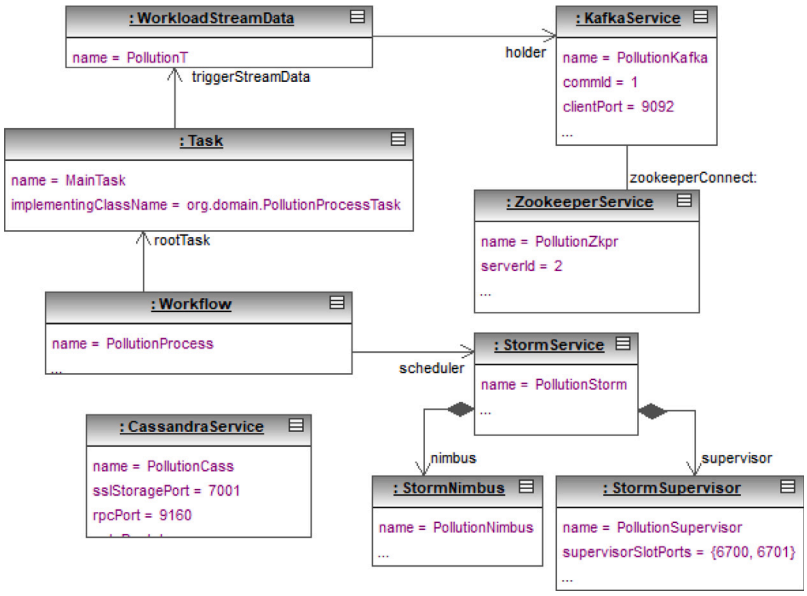

Fig. 9. PlatformService and workflow instances for the pollution use case.

their corresponding instances are added to the model, configured and related with the corresponding elements already present in this one: the `KafkaService` instance, `PollutionKafka`, is assigned to the `holder` attribute of the `PollutionT` topic, since Kafka is responsible for its management, whereas the `StormService` instance, `PollutionStorm`, is assigned as `scheduler` of the `PollutionProcess` workflow, since its `mainTask` must be deployed on Storm. Fig. 9 represents, using a UML object diagram, these instances of the pollution model and how they are related.

Once the design model of the application has been defined, it is time to add the deployment-related modelling elements, i.e., defining the deployment platform. In this case, a hybrid deployment scenario is considered, where Storm and Cassandra are deployed on physical edge nodes, while Kafka and Zookeeper are deployed on containers managed by Kubernetes. As a consequence of this fact, as shown in Fig. 10, the `CassandraService` and `StormService` instances introduced before are related through their `host` attribute to their corresponding `ProcessingNodeCluster` instances, `CassandraCluster` and `StormCluster`, which must be added to the model, and which gathers the corresponding `PhysicalProcessingNode` instances: the node for Cassandra, `CassandraNode`, and two nodes for deploying Storm, `NimbusNode` and `SupervisorNode`.

On the other hand, Kafka and Zookeeper are deployed on containers on the cloud managed by Kubernetes, hence, their corresponding instances in the model are related to a single `KubernetesCluster` instance through their `host` attribute, as it is depicted in Fig. 11.

To include in the model the rest of information required to support the configuration and deployment process, an instance of `NodeDeploymentConf` or `KubernetesDeploymentConf` must be added to each `PlatformService` instance defined in the model, according to where it is deployed. The instances added in this example are represented also in Figs. 10 and 11, showing only part of their attributes.

This model would be used as input of the tool that automatically generates the set of configuration files and scripts required to deploy and launch the complete application in the deployment platform. For minimising the complexity of the deployment of services in Kubernetes, the new version of RAI4DeploymentTool relies on Helm [47], a package manager for Kubernetes, so the set of files generated for deploying Kafka and Zookeeper follow the rules defined by Helm. To give an idea of the amount on files required to deploy an application like this, the number of generated files for this specific deployment model sums a total of six configuration files and seven scripts, which are generate and distributed by the tool on the different nodes of the platform:

- CassandraNode receives two Cassandra configuration files and the script that launches the CassandraService instance, `PollutionCass`, on it.
- SupervisorNode receives one configuration file and the script required to launch the `PollutionSupervisor` instance on it.
- NimbusNode receives one configuration file and three scripts to launch, respectively, the `PollutionNimbus` and `PollutionUI` Storm instances, and to trigger the `PollutionProcess` workflow execution on the `PollutionNimbus` instance.
- LocalNode receives two configuration files and four Helm scripts: two of them required to launch `PollutionKafka` and `PollutionZkpr` instances on Kubernetes, and two more to remotely create the `PollutionT` topic on the Kafka instance managed by Kubernetes.

To provide some insights into the model definition and deployment process, Fig. 12 demonstrates how a user would utilise the metamodel and deployment tool to create this hybrid application. PollutionHybridKubernetes.xmi file contains the model of the application, which, as

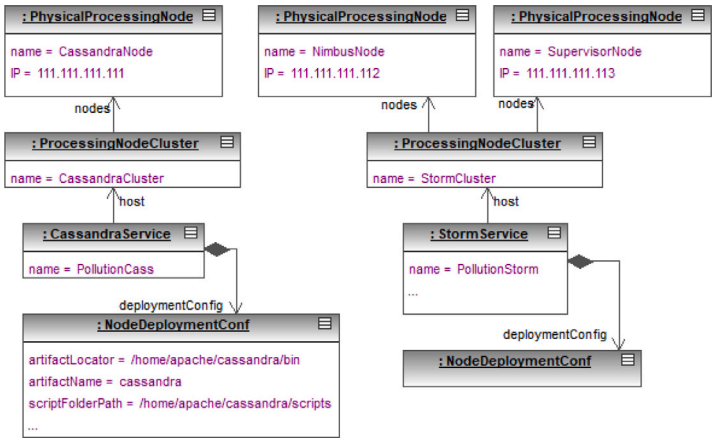

Fig. 10. Deployment configuration for the Cassandra and Storm services.

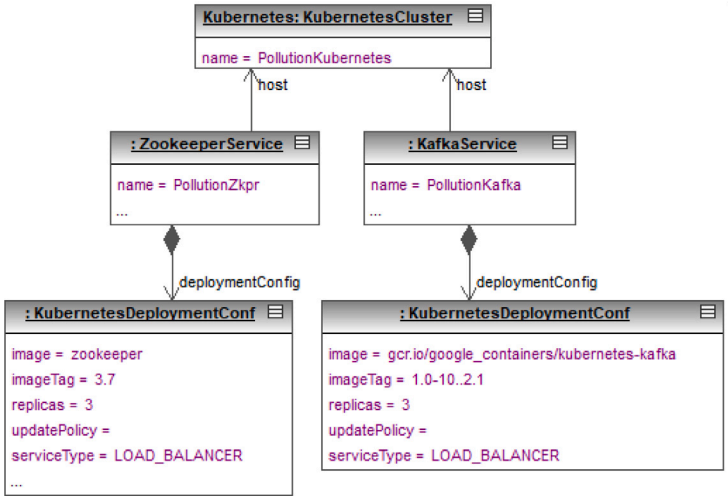

Fig. 11. Deployment configuration for the Kakfa and Zookeeper services.

said in Section 3.2 and shown in the top part of Fig. 12, it is edited using the Ecore Sample Reflective Editor. The figure also shows how configuration properties are assigned to the elements of the model through the Properties tab (the figure is focused on the Zookeeper instance of the model). Once the model is finished, the deployment tool is triggered. As it was explained in Section 3.2, the application invokes the `deployAndLaunch()` method of the root class, `ComputationalSystem`, whose code is partially shown in the figure. This method generates the configuration files required by each deployable component and the scripts required to trigger their execution. As an example, the generated files for the Zookeeper instance are shown in the bottom of the figure. The `deployAndLaunch()` method itself sends these files, via scp, to their corresponding hosts and executes the scripts, via ssh, to trigger the execution of the application.

5.1. Deployment variations of IoT model

As an evidence of the reutilisation capacity that is provided by the proposed metamodel and the benefits of using the associated deployment tool to automate the whole process, we compare different deployment scenarios for this same case study. All of them share the same functionality, i.e. the same design shown in Fig. 9, but each one differs from the others in the place of deployment, on-premise or cloud, the mode of deployment, physical or virtualised, or the number of nodes used per service. The considered scenarios are the following:

- The first scenario (base or reference case) is the simplest, with all the `PlatformService` instances (one Zookeeper, one Kafka, one Cassandra, one Storm Nimbus and one Storm supervisor) deployed on a single physical node.
- The second one is a distributed and scaled version of the base scenario, where each `PlatformService` instance is deployed on its own physical node, and besides, some services (Kafka and Zookeeper) are scaled up to three nodes.
- The third scenario is a distributed version of the base scenario but deployed on a hybrid environment, where some services (Storm and Cassandra) are deployed on physical processing nodes whereas Kafka and Zookeeper are deployed on AWS EC2 virtual machines.
- The last scenario is the hybrid deployment explained in the previous section, using Kubernetes to containerise Kafka and Zookeeper, while Storm and Cassandra remain deployed on physical nodes.

The number of instances declared in the model and the number of files generated by the tool in order to deploy each scenario are shown in Table 2 whereas the XMI files corresponding to them are available in git.[3]

Next we describe the process to go through all the different scenarios just making some tweaks to the models:

[3] https://github.com/istr-uc/RAI4DeploymentTool.

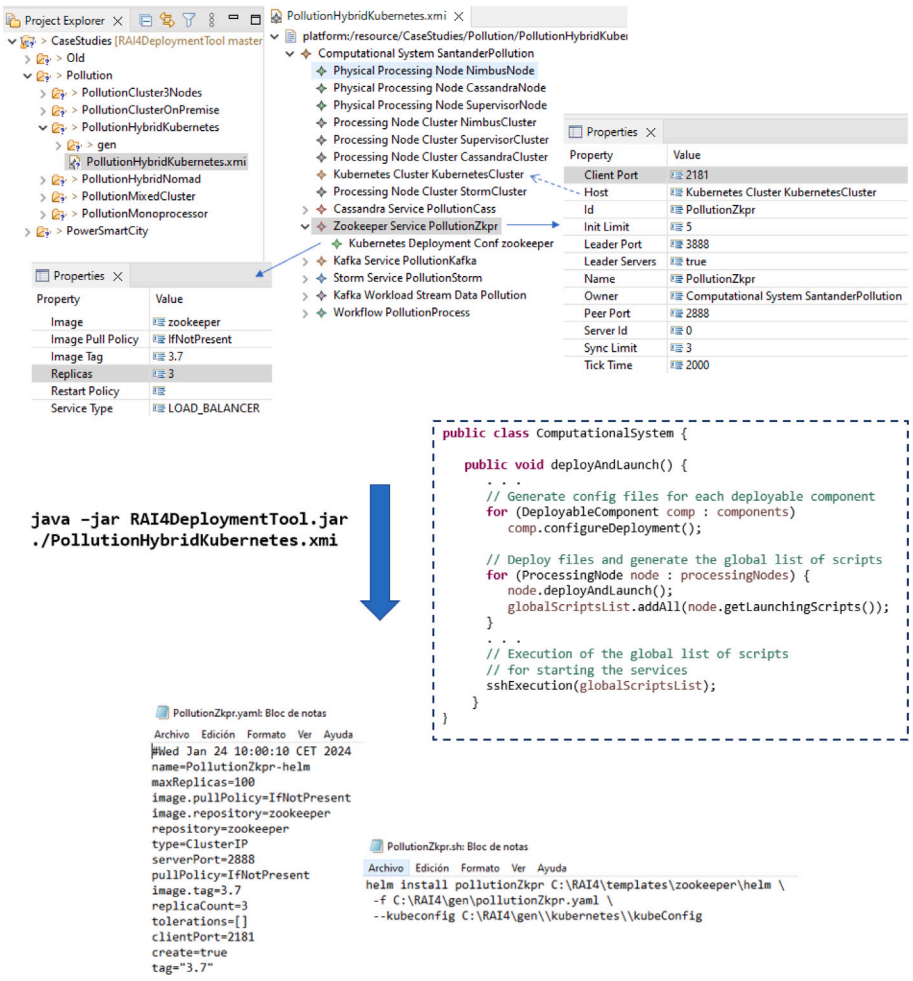

Fig. 12. Model definition, file generation and deployment process for the Zookeeper service from the pollution case study.

- Creating the first model requires defining the following instances: one `ComputationalSystem`, one `PhysicalProcessingNode` and its corresponding `ProcessingNodeCluster`, one instance for each `PlatformService` (which makes a total of six, including the Storm nimbus and worker), two for the workflow (`Workflow` and `Task`) and one `WorkloadStreamData` for the topic. After describing those instances, we add the corresponding `NodeDeploymentConf` instance to each `PlatformService` and to the `Workflow`, and we are ready to deploy the first scenario using our deployment tool.
- To create the model for the second scenario, we add to the first model the additional `PhysicalProcessingNode` instances with their corresponding `ProcessingNodeCluster` instances and change the `host` reference of each `PlatformService` to refer to those clusters (the clusters for Storm and Cassandra will have only one node, while the ones for Kafka and Zookeeper will contain three). These changes, that should not take more than five minutes to a user familiarised with the metamodel, allow us to configure another deployment easily and quickly.
- The third model can be obtained from the second one, only by changing the `PhysicalProcessingNode` instances corresponding to Kafka and Zookeeper to `VirtuaProcessingNode` instances and removing the extra nodes used to scale them before, without requiring any other change in the platform services.
- Finally, since the last deployment scenario is a little more complex than the previous ones, it is also the most time consuming but we can say that for a user familiarised with the metamodel it can take around ten minutes. In this case, the model of the second scenario would be taken as starting point. We would need to define a new `KubernetesCluster` instance and make the `KakfaService` and `ZookeeperService` instances refer to it as their `host`. This would require to delete from the model the `ProcessingNodeCluster` instances used before to deploy Kafka and Zookeeper. Besides, the new `KubernetesDeploymentConf` instances should be added to the model to configure the deployment of these two platform services on Kubernetes. If another orchestration technology would be used, Nomad or Docker Swarm, only the `KubernetesCluster` and `KubernetesDeploymentConf` instances should be replaced by the ones corresponding to the chosen technology.

As we can observe, the greater the complexity of the deployment, either by increasing the number of nodes or by virtualising the services, the number of instances in the model and the configuration/script files to write and send to the respective nodes grow slightly but the consumed development time is very low. Even more, thanks to the use of RAI4DeploymentTool, the process of generating and sending these files and executing the services on their corresponding nodes is almost insignificant compared with doing it by hand, which saves time and money to IT teams when reconfiguring the platform. Furthermore, the use of this tool avoids the edition of configuration files and the launching of each service manually, which reduces errors in the assignment of values to configuration properties as well as the ordered launching of services. Therefore, we consider that this tool is very useful in the test stage of the application, when issues such as latency, throughput and response time of the system must be measured in different scenarios

Table 2
Evaluation/model complexity of the different deployment scenarios.

No. Scenario	Description	No. instances	No. generated files
1	Single node on premise	21	14
2	Distributed and scaled cluster on premise	34	25
3	Hybrid cluster (on premise and cloud)	30	15
4	Hybrid cluster (on premise and containers)	27	16

in order to choose the best setting for meeting the requirements of the application.

6. Discussion

As a result of the experience gained in the development and deployment of data-intensive applications, we can highlight the advantages and open issues of our model-based reference architecture and deployment tool.

Firstly, it provides a systematic method for dealing with the design and deployment of these applications. The metamodel allows designers to address their conception and development in two phases: first, to carry out a high-level design where the services, data streams, workflows and metrics that constitute the applications are described, focusing their efforts on reasoning about the concrete problem to solve being aware of the rest of applications in the environment, in particular those with which they must interoperate; and, subsequently, to focus on a second stage dedicated to the selection and evaluation of technologies and development of artifacts, for which different implementation options are offered, also specifying the essential parameters for their configuration. Moreover, the extension incorporated in this work on the hybridisation and containerisation of applications makes the metamodel even more versatile and powerful, as it can cover all current deployment alternatives in distributed and scalable environments. This feature is absolutely essential for the proposal to be used by the productive sector.

As the first and second stages are decoupled, once the first one is finished, different teamworks can be in charge of developing the software artifacts that implement the defined processing tasks while system administrator experts determine the deployment configuration and describe it in the model, thus being documented.

Thanks to the deployment tool provided, the modelled system can be deployed and running within minutes. Moreover, administrators can have different pre-designed configurations, so that before final deployment, they can run performance benchmarks, compare and choose the most satisfactory configuration. This task, which is tedious if done manually, can be automated with our strategy.

Although dealing with the complexity of the first design is the main contribution of the metamodel, the ease of integration of other applications in the platform should not be overlooked, i.e., incorporating new topics and workflows, as well as monitoring metrics, among others. By reasoning about an already created model, the designer takes in mind the rest of the processes that coexist in the digital platform, thus favouring the sharing of resources and interoperability and avoiding inopportune collisions.

Nevertheless, our tool still has aspects for improvement, some of them already in development phase. First, it would be very useful for novice administrators that the tool could offer suitable configurations for expected workloads based on the historic log of the platform. That means, monitoring KPIs were used as input of a predictor that specifying a load state of platform would suggest a configuration per each service. Second, the interface for model definition and tool usage could be more user-friendly. Therefore, we are currently working on a new version of the environment, built as a stand-alone Eclipse Rich Application, that integrates the deployment tool as an automatic model-to-text transformation, and a graphical editor for the models. This will be available for the second semester of 2024. Third, support for Nomad is being implemented in the deployment tool due to its increasing use for containering simple applications, specially in edge nodes. Finally, as a consequence of this upgrade of the RA, another issue to solve is the migration of models described with the previous version of RAI4.0.

Another criticism that could be made is the eternal trade-off between designing a generic tool or a specific one, i.e. the need of choosing between having the best configuration for a specific technology versus the ease of being able to decide freely. We have bridged this trade-off by allowing specialisation of the platform services according to their underlying technologies. Another concern is that the complexity of our RA grows with the addition of new services, such as the upcoming ones related to automating the generation, publishing and updating of machine learning models (MLOps) [48], leading to an increasing learning curve for its use. Hence, we are evaluating how to mitigate this effect. One possibility could involve outlining steps in the Continuous Model-driven Engineering (CMDE) concept, where the target of Continuous Delivery is an MDE artifact created through collaboration within an MDE ecosystem, as proposed by Garcia and Cabot [49]. In this context, DevOps principles evolve into ModDevOps, supporting a seamless continuum of models from design to runtime and vice versa, that means, with data gathered from the environment to drive dynamic reconfiguration at runtime or finding improved designs [50].

Finally, we have no real evidence of companies using our model directly, as the experiments have been developed by the authors through collaboration agreements in which the company receives the required implemented solution, generated according to our strategy.

7. Conclusions and future lines

Few years ago, cloud computing was the solution for massive data management and computing. However, the real-time constrains and the specific requirements of cyber–physical systems present in industrial environments have led to design and deploy data-intensive applications that must be easily reconfigured so that the services and resources used meet the functional and non-functional requirements of each application. Frequently, latency and throughput are restricted to certain limits, making it necessary to reduce delays and consequently deploy processing tasks close to where the data is produced (edge computing). In addition, it is highly desirable that tasks and services are isolated and containerised in order to make their deployment easier, more flexible and free from incompatibilities. These aspects led us to extend our RAI4 metamodel to support containerised and hybrid deployment settings.

RAI4 is a model-based tool that helps software developers and IT teams to design and deploy modern data intensive applications on big data architectures. This is defined at two levels: a technology-independent, extensible and reusable model and another specialised one that collects the most popular cutting-edge technologies. This involves the following advantages: on the one hand, this metamodel allows practitioners to design their applications bearing in mind all the essential elements in data-intensive applications and on the other hand, to configure different deployments and to measure their performance by means of the monitoring metrics implemented in their own solution. This tool can therefore be classified in the group of DevOps technologies which help teams streamline their development and delivery processes, reduce errors and delays, and increase their ability to respond to changing requirements and business needs [12].

In our experience, by using simple models to describe the design and the deployment of complex applications and by automating the deployment starting from such models, we have achieved considerable gains both in terms of reasoning on and documenting our digital platform, as well as in terms of time saved when deploying and redeploying with different configurations. Likewise, the time required for the integration of new applications or services is reduced since parts of the model can be reused.

Ongoing work primarily focuses on enhancing the deployment tool by supporting new technologies, such as Nomad, and improving usability with a graphical editor for deployment models. Additionally, a model-driven approach is being implemented, enabling files and scripts generation through a model-to-text transformation. Likewise, we continue designing new case studies, for instance, predictive maintenance [51] or smart houses [52].

Disclaimer

Finally, we would point out that the software identified in this paper is used in order to check the completeness and capability of implementation of our reference architecture in different use cases. This does not imply their recommendation or that these products are necessarily the best available for that purpose.

CRediT authorship contribution statement

Ricardo Dintén: Writing – review & editing, Software, Investigation, Conceptualization. **Patricia López Martínez:** Writing – review & editing, Visualization, Software, Methodology. **Marta Zorrilla:** Writing – review & editing, Writing – original draft, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Code publicly available at: https://github.com/istr-uc/RAI4DeploymentTool.

Acknowledgements

This work was partially supported by MCIN/ AEI /10.13039/501100011033/ FEDER "Una manera de hacer Europa" under grant PID 2021-124502OB-C42 (PRESECREL) and the predoctoral program "Concepción Arenal del Programa de Personal Investigador en formación Predoctoral" funded by Universidad de Cantabria and Cantabria's Government (BOC 18-10-2021).

References

[1] R.Y. Zhong, X. Xu, E. Klotz, S.T. Newman, Intelligent manufacturing in the context of industry 4.0: a review, Engineering (ISSN: 2095-8099) 3 (5) (2017) 616–630, http://dx.doi.org/10.1016/J.ENG.2017.05.015, https://www.sciencedirect.com/science/article/pii/S2095809917307130.

[2] S. Wang, J. Wan, D. Li, C. Zhang, Implementing smart factory of industrie 4.0: An outlook, Int. J. Distrib. Sens. Netw. 12 (1) (2016) http://dx.doi.org/10.1155/2016/3159805.

[3] N. Bicocchi, G. Cabri, F. Mandreoli, M. Mecella, Dynamic digital factories for agile supply chains: An architectural approach, J. Ind. Inf. Integr. 15 (2019) 111–121, http://dx.doi.org/10.1016/j.jii.2019.02.001.

[4] R. Heidel, M. Hoffmeister, M. Hankel, U. Döbrich, The Reference Architecture Model RAMI 4.0 and the Industrie 4.0 Component, VDE Verlag, 2019, URL https://www.vde-verlag.de/buecher/624990/the-reference-architecture-model-rami-4-0-and-the-industrie-4-0-component.html.

[5] Industrial Internet Consortium, Industrial internet reference architecture v1.9, 2019.

[6] Fraile, Sanchis, Poler, Ortiz, Reference models for digital manufacturing platforms, Appl. Sci. 9 (20) (2019) http://dx.doi.org/10.3390/app9204433.

[7] M. Hermann, T. Pentek, B. Otto, Design principles for industrie 4.0 scenarios, in: 2016 49th Hawaii International Conference on System Sciences, HICSS, 2016, pp. 3928–3937, http://dx.doi.org/10.1109/HICSS.2016.488.

[8] C.E. Belman-López, J.A. Jiménez-García, S. Hernández-González, Análisis exhaustivo de los principios de diseño en el contexto de industria 4.0, Rev. Iberoam. Autom. Inform. Ind. 17 (4) (2020) 432–447, http://dx.doi.org/10.4995/riai.2020.12579.

[9] R. Tolosana-Calasanz, J. Ángel Bañares, J.-M. Colom, Model-driven development of data intensive applications over cloud resources, Future Gener. Comput. Syst. 87 (2018) 888–909, http://dx.doi.org/10.1016/j.future.2017.12.046.

[10] C. Pahl, A. Brogi, J. Soldani, P. Jamshidi, Cloud container technologies: A state-of-the-art review, IEEE Trans. Cloud Comput. 7 (3) (2019) 677–692, http://dx.doi.org/10.1109/TCC.2017.2702586.

[11] M. De Donno, K. Tange, N. Dragoni, Foundations and evolution of modern computing paradigms: Cloud, IoT, edge, and fog, IEEE Access 7 (2019) 150936–150948, http://dx.doi.org/10.1109/ACCESS.2019.2947652.

[12] A.V. Jha, R. Teri, S. Verma, S. Tarafder, W. Bhowmik, S.K. Mishra, B. Appasani, A. Srinivasulu, N. Philibert, From theory to practice: Understanding DevOps culture and mindset, Cogent Eng. 10 (1) (2023) 2251758, http://dx.doi.org/10.1080/23311916.2023.2251758, arXiv:https://doi.org/10.1080/23311916.2023.2251758.

[13] P. Ataei, A. Litchfield, The state of big data reference architectures: A systematic literature review, IEEE Access 10 (2022) 113789–113807, http://dx.doi.org/10.1109/ACCESS.2022.3217557.

[14] P.L. Martínez, R. Dintén, J.M. Drake, M.E. Zorrilla, A big data-centric architecture metamodel for industry 4.0, Future Gener. Comput. Syst. 125 (2021) 263–284, http://dx.doi.org/10.1016/j.future.2021.06.020.

[15] R. Qasha, Z. Wen, J. Cała, P. Watson, Sharing and performance optimization of reproducible workflows in the cloud, Future Gener. Comput. Syst. 98 (2019) 487–502, http://dx.doi.org/10.1016/j.future.2019.03.045.

[16] D. Oppenheimer, A. Ganapathi, D.A. Patterson, Why do internet services fail, and what can be done about it? in: Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems - Volume 4, USITS '03, USENIX Association, USA, 2003, p. 1.

[17] D. Corral-Plaza, I. Medina-Bulo, G. Ortiz, J. Boubeta-Puig, A stream processing architecture for heterogeneous data sources in the Internet of Things, Comput. Stand. Interfaces 70 (2020) 103426, http://dx.doi.org/10.1016/j.csi.2020.103426.

[18] F. Carcillo, A. Dal Pozzolo, Y.-A. Le Borgne, O. Caelen, Y. Mazzer, G. Bontempi, SCARFF: A scalable framework for streaming credit card fraud detection with spark, Inf. Fusion 41 (2018) 182–194, http://dx.doi.org/10.1016/j.inffus.2017.09.005.

[19] R. Sahal, J.G. Breslin, M.I. Ali, Big data and stream processing platforms for industry 4.0 requirements mapping for a predictive maintenance use case, J. Manuf. Syst. 54 (2020) 138–151, http://dx.doi.org/10.1016/j.jmsy.2019.11.004.

[20] A. Salis, A. Marguglio, G. De Luca, S. Razzetti, W. Quadrini, S. Gusmeroli, An edge-cloud based reference architecture to support cognitive solutions in process industry, Procedia Comput. Sci. 217 (2023) 20–30, http://dx.doi.org/10.1016/j.procs.2022.12.198, URL https://www.sciencedirect.com/science/article/pii/S1877050922022761, 4th International Conference on Industry 4.0 and Smart Manufacturing.

[21] P. Ataei, A. Litchfield, NeoMycelia: A software reference architecturefor big data systems, in: 2021 28th Asia-Pacific Software Engineering Conference, APSEC, 2021, pp. 452–462, http://dx.doi.org/10.1109/APSEC53868.2021.00052.

[22] S. Nadal, V. Herrero, O. Romero, A. Abelló, X. Franch, S. Vansummeren, D. Valerio, A software reference architecture for semantic-aware big data systems, Inf. Softw. Technol. 90 (2017) 75–92, http://dx.doi.org/10.1016/j.infsof.2017.06.001, URL https://www.sciencedirect.com/science/article/pii/S0950584917304287.

[23] A. Margara, G. Cugola, N. Felicioni, S. Cilloni, A model and survey of distributed data-intensive systems, 2022, http://dx.doi.org/10.48550/ARXIV.2203.10836.

[24] D. Pérez-Palacín, J. Merseguer, J.I. Requeno, M. Guerriero, E.D. Nitto, D.A. Tamburri, A UML profile for the design, quality assessment and deployment of data-intensive applications, Softw. Syst. Model. 18 (6) (2019) 3577–3614, http://dx.doi.org/10.1007/s10270-019-00730-3.

[25] C. Blanco, D.G. Rosado, Ángel Jesús Varela-Vaca, M.T. Gómez-López, E. Fernández-Medina, Onto-CARMEN: Ontology-driven approach for cyber–physical system security requirements meta-modelling and reasoning, Internet Things 24 (2023) 100989, http://dx.doi.org/10.1016/j.iot.2023.100989, URL https://www.sciencedirect.com/science/article/pii/S2542660523003128.

[26] O. Bentaleb, A.S.Z. Belloum, A. Sebaa, A. El-Maouhab, Containerization technologies: taxonomies, applications and challenges, J. Supercomput. 78 (1) (2022) 1144–1181, http://dx.doi.org/10.1007/s11227-021-03914-1.

[27] M.U. Ilyas, M. Ahmad, S. Saleem, Internet-of-things-infrastructure-as-a-service: The democratization of access to public internet-of-things infrastructure, Int. J. Commun. Syst. 33 (16) (2020) e4562, http://dx.doi.org/10.1002/dac.4562, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/dac.4562, e4562 dac.4562.

[28] Q. Zhang, L. Liu, C. Pu, Q. Dou, L. Wu, W. Zhou, A comparative study of containers and virtual machines in big data environment, 2018, arXiv:1807.01842.

[29] D. Merkel, Docker: Lightweight linux containers for consistent development and deployment, Linux J. 2014 (239) (2014).

[30] I. Diamanti, Container adoption benchmark survey, report, 2018.

[31] A. Polvi, Coreos is building a container runtime, rkt, 2014, URL https://coreos.com/blog/rocket/. (Accessed 14 May 2023).

[32] Canonical Ltd, Linux containers - LXC - documentation, 2023, URL https://linuxcontainers.org/lxc/documentation/. (Accessed 14 May 2023).
[33] Q. Zhang, L. Liu, C. Pu, Q. Dou, L. Wu, W. Zhou, A comparative study of containers and virtual machines in big data environment, 2018, CoRR abs/1807.01842, arXiv:1807.01842, URL http://arxiv.org/abs/1807.01842.
[34] I. Jaju, Maximizing DevOps Scalability in Complex Software Systems: Maximizing DevOps Scalability in Complex Software Systems (Master's thesis), Upsala Universitet, 2023.
[35] K. Hightower, B. Burns, J. Beda, Kubernetes: Up and Running Dive into the Future of Infrastructure, first ed., O'Reilly Media, Inc., 2017.
[36] F. Soppelsa, C. Kaewkasi, Native Docker Clustering with Swarm, Packt Publishing, 2017.
[37] HashiCorp, Nomad by HashiCorp, 2022, URL https://www.nomadproject.io/.
[38] M. Straesser, J. Mathiasch, A. Bauer, S. Kounev, A systematic approach for benchmarking of container orchestration frameworks, in: Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering, ICPE '23, Association for Computing Machinery, New York, NY, USA, 2023, pp. 187–198, URL https://doi.org/10.1145/3578244.3583726.
[39] M. Wurster, U. Breitenbücher, M. Falkenthal, et al., The essential deployment metamodel: a systematic review of deployment automation technologies, SICS Softw.-Inensiv. Cyber-Phys. Syst. 35 (2020) 63–75, http://dx.doi.org/10.1007/s00450-019-00412-x.
[40] F. Paraiso, S. Challita, Y. Al-Dhuraibi, P. Merle, Model-driven management of docker containers, in: 2016 IEEE 9th International Conference on Cloud Computing, CLOUD, 2016, pp. 718–725.
[41] N. Petrovic, M. Tosic, SMADA-Fog: Semantic model driven approach to deployment and adaptivity in fog computing, Simul. Model. Pract. Theory 101 (2020) 102033, http://dx.doi.org/10.1016/j.simpat.2019.102033, Modeling and Simulation of Fog Computing.
[42] M. Gusev, What makes dew computing more than edge computing for internet of things, in: 2021 IEEE 45th Annual Computers, Software, and Applications Conference, COMPSAC, 2021, pp. 1795–1800, http://dx.doi.org/10.1109/COMPSAC51774.2021.00269.
[43] D. Steinberg, F. Budinsky, M. Paternostro, M. E., EMF: Eclipse Modeling Framework 2.0, second ed., Addison-Wesley Professional, 2009.
[44] W. Wingerath, F. Gessert, S. Friedrich, N. Ritter, Real-time stream processing for big data, Inform. Technol. 58 (4) (2016) 186–194, http://dx.doi.org/10.1515/itit-2016-0002.
[45] M.R. Delavar, A. Gholami, G.R. Shiran, Y. Rashidi, G.R. Nakhaeizadeh, K. Fedra, S.H. Afshar, A novel method for improving air pollution prediction based on machine learning approaches: A case study applied to the capital city of Tehran, ISPRS Int. J. Geo Inf. 8 (2) (2019) 99, http://dx.doi.org/10.3390/ijgi8020099.
[46] G. Díaz, H. Macià, V. Valero, et al., An intelligent transportation system to control air pollution and road traffic in cities integrating CEP and colored Petri nets, Neural Comput. & Appl. 32 (2020) 405–426, http://dx.doi.org/10.1007/s00521-018-3850-1.
[47] Helm Authors and The Linux Foundation, Helm, 2024, URL https://helm.sh/.
[48] O. Debauche, J.B. Nkamla Penka, M. Hani, A. Guttadauria, R. Ait Abdelouahid, K. Gasmi, O. Ben Hardouz, F. Lebeau, J. Bindelle, H. Soyeurt, N. Gengler, P. Manneback, M. Benjelloun, C. Bertozzi, Towards a unified architecture powering scalable learning models with IoT data streams, blockchain, and open data, Information 14 (6) (2023) http://dx.doi.org/10.3390/info14060345, URL https://www.mdpi.com/2078-2489/14/6/345.
[49] J. Garcia, J. Cabot, Stepwise adoption of continuous delivery in model-driven engineering, in: J.-M. Bruel, M. Mazzara, B. Meyer (Eds.), Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment, Springer International Publishing, Cham, 2019, pp. 19–32.
[50] B. Combemale, M. Wimmer, Towards a model-based DevOps for cyber-physical systems, in: J.-M. Bruel, M. Mazzara, B. Meyer (Eds.), Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment, Springer International Publishing, Cham, 2020, pp. 84–94.
[51] R.D. Herrero, M.E. Zorrilla, An I4.0 data intensive platform suitable for the deployment of machine learning models: a predictive maintenance service case study, in: F. Longo, M. Affenzeller, A. Padovano (Eds.), Proceedings of the 3rd International Conference on Industry 4.0 and Smart Manufacturing (ISM 2022), Virtual Event / Upper Austria University of Applied Sciences - Hagenberg Campus - Linz, Austria, 17-19 November 2021, in: Procedia Computer Science, vol. 200, Elsevier, 2021, pp. 1014–1023, http://dx.doi.org/10.1016/J.PROCS.2022.01.300.
[52] M.E. Zorrilla, Á. Ibrain, Bernard, an energy intelligent system for raising residential users awareness, Comput. Ind. Eng. 135 (2019) 492–499, http://dx.doi.org/10.1016/j.cie.2019.06.040.