Journal Pre-proof

A big data-centric architecture metamodel for Industry 4.0

Patricia López Martínez, Ricardo Dintén, José María Drake, Marta Zorrilla

PII: S0167-739X(21)00215-6
DOI: https://doi.org/10.1016/j.future.2021.06.020
Reference: FUTURE 6127

To appear in: *Future Generation Computer Systems*

Received date : 6 August 2020
Revised date : 4 May 2021
Accepted date : 15 June 2021

Please cite this article as: P.L. Martínez, R. Dintén, J.M. Drake et al., A big data-centric architecture metamodel for Industry 4.0, *Future Generation Computer Systems* (2021), doi: https://doi.org/10.1016/j.future.2021.06.020.

A big data-centric architecture metamodel for Industry 4.0

Patricia López Martínez, Ricardo Dintén*, José María Drake, Marta Zorrilla

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

Abstract

The effective implementation of Industry 4.0 requires the reformulation of industrial processes in order to achieve the vertical and horizontal digitalisation of the value chain. For this purpose, it is necessary to provide tools that enable their successful implementation. This paper therefore proposes a data-centric, distributed, dynamically scalable reference architecture that integrates cutting-edge technologies being aware of the existence of legacy technology typically present in these environments. In order to make its implementation easier, we have designed a metamodel that collects the description of all the elements involved in a digital platform (data, resources, applications and monitoring metrics) as well as the necessary information to configure, deploy and execute applications on it. Likewise, we provide a tool compliant to the metamodel that automates the generation of configuration, deployment and launch files and their corresponding transference and execution in the nodes of the platform. We show the flexibility, extensibility and validity of our software artifacts through their application in two case studies, one addressed to preprocess and store pollution data and the other one, more complex, which simulates the management of an electric power distribution of a smart city.

Keywords: Data-centric architecture, Data-intensive applications,

*Corresponding author

Email addresses: `lopezpa@unican.es` (Patricia López Martínez), `ricardo.dinten@unican.es` (Ricardo Dintén), `drakej@unican.es` (José María Drake), `marta.zorrilla@unican.es` (Marta Zorrilla)

Model-based development, Industry 4.0, Big data, Metamodel

1. Introduction

The term “Industrie 4.0” stands for the fourth industrial revolution, the next stage in the organisation and control of the entire value stream along the life cycle of a product [1]. The Industry 4.0 deals with the transformation of the traditional pyramid model of automation to a network model of interconnected services, combining Operational Technology (OT) with Information Technology (IT) [2]. Its aim is to achieve the required operation with a qualitative improvement in the automation and optimisation of the industrial processes as well as meeting the rising and global demand of customized products. This new industrial model is based on the ubiquity and connectivity of data, people, processes, services and Cyber-Physical Systems (CPS) that exchange and exploit the information generated at each level of the architecture (cyber-physical, intermediation and application levels) in order to get a decentralized production and adaptable to changes in real time [3].

Considering the importance of adopting new ICT-enabled (Information and Communication Technologies) production methods for industries, in 2016 Germany published the Reference Architecture Management Industrial 4.0 (RAMI 4.0) [4] and United States, the Industrial Internet Reference Architecture (IIRA) [5] with the aim of facilitating their adoption and continuous evolution. Both provide viewpoints or architectural axis to guide manufacturers in the building, implementation and operation of industrial systems but none of them offer tools for dealing with their conception, design and instantiation. Later, Salkin et al. [6] pointed out the need of a use case framework to enable a successful implementation of Industry 4.0 and exposed real use cases to exemplary how to integrate and connect new technologies to develop smart products and processes. However, they either offered a reference architecture that can be directly instantiated by the industrial environment. Nevertheless, they pointed out the design principles that it should comply: agility, interoperability, virtualisation, decentralisation, real-time data management, service orientation and integrated business processes.

Big data and cloud computing are currently considered as key enablers that can lead to the building of the future industrial ecosystem by interlinking the cyber and physical worlds. The main reason is that these are able to manage the variety, velocity, volume and criticality of data that the industrial environment generates by means of highly distributed and scalable architectures that can be dynamically dimensioned to process workloads in real time [2]. Data indeed is the fundamental resource to promote Industry 4.0 and concepts like common "data buses" connecting factory environments have already been identified as the single most important enabler of novel I4.0 paradigms [7],[8]. That is why the metamodel for describing a reference architecture for Industry 4.0 proposed in this paper follows a highly flexible data-centric architecture in order to easily integrate disruptive big data and state-of-the-art technologies. The key characteristic of a data-centric architecture is that the data are the central asset and their storage and governance are the first step of the process, which precede the creation of any application or service [9]. Next, applications designed as data workflows, also known as data-intensive applications [10], are defined. These are characterized by manipulating data from one or more producers (data sources) and generating some output into some kind of data sink to be later consumed by other processes.

Although there are currently many frameworks for processing big data, for instance, tools from the Apache ecosystem, most of them were designed for their use in social networks. Therefore, their use and setting must be adapted to work in complex environments in which there are a great number of heterogeneous hardware resources, industrial internet of things and specific software running under strict real time constrains that must coexist with these new cutting-edge technologies and that will condition their deployment. As Al-Gumaei et al. [11] pointed out in their survey, general big data platforms should adopt more IIoT applications and platforms, while existing industrial cloud platforms should add big data frameworks to their portfolio.

To contribute in the development of Industry 4.0, this paper proposes:

- The description of a reference architecture for Industry 4.0, named RAI4.0, supported on a data-centric architecture and on a set of services that provide their management and the platform monitoring.
- The definition of a metamodel that collects the description of all elements involved in an RAI4.0-compliant digital platform (data, resources, workloads and metrics) as well as the necessary information to configure, deploy and execute workloads (applications) on it.
- A model-based tool compliant to the RAI4.0 metamodel that automates the deployment of RAI4.0 systems by automating the generation of configuration, deployment and launch files and their corresponding transference and execution in the nodes of the platform.

This paper is organized as follows. Section 2 relates scientific works about data architectures and modelling approaches for dealing with the requirements specification, the design and the building of complex systems applied to Industry 4.0. Section 3 describes the proposed RAI4.0 reference architecture for Industry 4.0. Section 4 explains the RAI4.0 metamodel that represents the concepts defined in the reference architecture. Section 5 summarizes how the configuration and deployment tool operates and the advantages that this provides. Section 6 presents two case studies to validate our purpose. The first one addressed to the read and storage of pollution data and the second one, to the management of electric power consumption of a smart medium-size city. Section 7 discusses the advantages and current limitations of our proposal and compares our reference architecture with other ones found in the literature. Likewise, we comment the strategy designed to include relevant issues such as data quality and security aspects in our proposal. Finally, Section 8 summarizes the main contributions of this work and comments open future lines.

2. Industry 4.0, big data architectures and model-based tools

Industry 4.0 is a relative new term that arose in Germany to refer to the digital transformation of the industry, also known as “Intelligent Factory” or “Industrial Internet” [12]. In the first place, it is important to highlight that this digitalisation is not only the application of disruptive technology, such as robotisation or automation, to production and service processes, but also encompasses a fundamental aspect that is the application of the Internet of Things (the infrastructure of interconnection among devices), the sensorization and the analysis of massive data from each one of the elements that take part in the productive processes to simulate what-if scenarios and make smarter decisions. These decisions are mainly aimed at gaining efficiency, better adapting production and supply to demand, integrating the entire value chain of the company, decentralizing decision making and predicting results accurately. In short, it is a revolutionary shift that will lead to changes in the consumption, labour market [13] and design of manufacturing processes [14].

Industry 4.0 urges traditional systems to evolve towards the creation of ecosystems enabling more flexible production processes through connecting systems and sharing data [2]. ICT sector provides many data-related technologies to progress in the effective implementation of Industry 4.0 such as (i) CPS, (ii) Industrial Internet of Things (IIoT), (iii) cloud solutions & decentralized services, and (iv) big data & stream processing technologies for processing large amounts of production data in real time [8]. But their adoption is still limited. According Mabkhot et al. [14], the smart factory system is still only a vision.

In the literature, we found several works that test and apply these cutting-edge technologies to failure minimization [15], smart farming [16], building predictive models for providing smart energy services [17], raising home users’ awareness for sustainable energy consumption [18] or improving the productivity by combining data from different collaborative systems and using them in the design, production, and product delivery processes [19][20] to name a few. But, as Mabkhot et al. pointed out [14], one of the biggest challenges in Industry

4.0 nowadays is to achieve a high level of sharing and interchanging information between all the components involved: products, production infrastructure and processes, control systems and reactive real-time applications. It must bear in mind that, in industrial environments, data sources are numerous and the complexity of their interoperability is increased when various management levels, suppliers and consumers use them. Furthermore, there is still non-solved security and authentication issues [21, 22]. Therefore, it is essential to provide architectural proposals that orchestrate all these elements and provide manufacturers with methodological tools to address their transition towards Industry 4.0 as the one offered in this paper.

2.1. Design principles for Industry 4.0

In order to design our architecture, we have taken the requirements from state-of-the-art reviews of the ongoing research on the Industry 4.0 phenomenon. These, in general, highlight its key design principles and technology trends without specifying a concrete architectural solution.

In [23], Hermann et al identify six design principles that are crucial for businesses to embrace in order to achieve the full benefits of Industry 4.0 technology. These design principles are - interoperability; virtualisation; decentralisation, real-time capability; service orientation; and modularity which are derived from the following Industry 4.0 components: CPS, IoT, internet of services and smart factory. This list is extended in [24] adding other principles not so linked to an implementable architectural solution but general issues such as horizontal and vertical integration, product personalisation or corporate social responsibility. In other recent work [25], the authors, in addition to the previous ones, include other characteristics, pointing out as principles the following ones: efficiency, integration, flexibility, decentralisation, customisation, virtualisation, security, is holistic, service-oriented, ubiquitous, collaborative, modular, robust, uses real-time information, makes data-optimized decisions, balances work life, and is autonomous and intelligent.

Next, we highlight those we consider that are more related and restrict the building of an implementable, general and agnostic regarding to technology solution. The architecture must be distributed and scalable, that means, must allow decentralisation not necessarily physical, but logical. This must combine the use of resources available in the cloud and in the edge computing to meet real time restrictions and utilize virtualisation techniques with the aim of facilitating the scaling. Likewise, the architecture proposed must facilitate the collaboration between machines, processes, systems and people through the exchange of information in real time, the triggering of actions, and decision-making and carrying out actions based on the information obtained from the environment. This leads to the need of using a data bus or another data-centric solution on which applications built under a modular and decoupled software architecture run with very few interdependencies between them to be easily adjusted. The construction of these modules under the service-oriented architecture [26] favours the establishment of new collaborative business models [27] (horizontal integration).

In this regard, Wingerath et al.[28] proposed kappa and lambda as general architectural solutions to deal with real-time stream processing. For instance, Arantes et al. [29] used a kappa architecture to develop a data analysis prototype tool for Industry 4.0 using SysML and Model-driven Engineering. Likewise, Raptis et al. [7] pointed out that data buses are the most important key enablers as a consequence of the crucial role that data play in the integration of the two worlds, the physical and cyber ones. Finally, IIRA [5] includes the concept of layered data buses in its reference architecture to enable the exchange and the sharing data, both data in motion and data at rest. In particular, IIRA suggests the layered data bus because it is a common architecture across IIoT systems in multiple industries since it provides low-latency, secure, peer-to-peer data communications across logical layers of the system. It is most useful for systems that must manage direct interactions between applications in the field, such as control, local monitoring and edge analytics [16]. Central to the data bus is a data-centric publish-subscribe communications model. Applications on the data bus simply "subscribe" to data they need and "publish" information they produce. Messages logically pass directly between the communicating nodes. The fundamental communications model implies both discovery—what data should be sent—and delivery—when and where to send it.

In short, these references allow us to endorse that our data-centric architecture meets the I4.0 requirements for most industrial scenarios.

2.2. Model-based development for Industry 4.0

Since the beginning of the fourth industrial revolution, researches and productive sector have worked in order to propose methodological approaches and tools to deal with the design and development of complex systems. Model-based development is a well-established discipline to support conception, design, assessment and development of software. Regarding its use and application in Industry 4.0 or its enabling technologies, we find particularly interesting the survey carried out by Wortmann et al. [30]. They identify model-based systems engineering (MBSE) as a key enabler for the development of complex systems, such as the ones typical from Industry 4.0, and hence they analyse the state of the art of MBSE for the smart factory through a systematic mapping study. Among other conclusions, they pointed out the majority of papers contribute methods and concepts to solve particular challenges of Industry 4.0 as we also checked before our related work search. Next, we described these works according to the stage of lifecycle that address: the requirement specification phase or the development stage.

In the first group, Petrasch and Hentschke [31] defined an Industry 4.0 process modeling language (I4PML) that extends the OMG's BPMN (Business Process Model and Notation) standard and describes a method for the specification of new components, e.g. IoT devices or actuation and sensing tasks that are present in the Industry 4.0 applications. With a similar purpose, Suri et al. [32] designed a model-based approach for creating and communicating business strategies (mission, goals, and tactics) and bridging the gap between the business strategies and corresponding operational processes using Business Motivation Model (BMM) and Business Process Modeling and Notation (BPMN) respectively. In addition, Manoj Kannan et al. [33] proposed a standardized development of information systems compliant with the manufacturing Reference Architecture Model for Industry 4.0 (RAMI 4.0) [4]. And, Durão et al. [34] presented a preliminary integrated component data model in UML for specifying the features of cyber-physical production systems and smart product that must be stored and exchanged with other components and assemblies along the entire production system. Finally, although not specific for Industry 4.0 but associated to an enabler technology, Lassoued et al. [35] proposed the Cloud Workflow Service Meta-Model (CWS2M) by extending BPMN 2.0 to support the modeling of cloud workflow services and offer the separation of organizational concerns from their technical achievement in the specification process.

Regarding contributions linked to the development stage, we found the followings. The authors of [36] proposed a MDE tool for application code generation in Hadoop framework. Santurkar et al. [37] defined a domain-specific language, named Stormgen, for specifying Storm topologies in order to make developers the programming of data streaming processes easier. Pérez-Palacín et al. [10, 38] developed a powerful UML profile specifically tailored to support the design, assessment and continuous deployment of data intensive applications in private or public clouds. Arantes et al. [39] built a MDE solution that allows modelling analytic applications with data sent by the cyber-physical systems through Kafka to be processed by Spark and Mlib following a kappa architecture.

As far as we know, there are no proposals that address the definition of an implementable reference architecture as the one described in this paper that considers data as central element of the digital platform, on which a set of global, heterogeneous and decentralized services are orchestrated. Nevertheless, we have found some interesting proposals such as [10] that might be integrated in our solution by means of model transformations. Although it may need to be extended since it is exclusively oriented to the development of data-intensive applications (DIA) deployed on the cloud.

3. Reference architecture for Industry 4.0

This section describes our digital platform reference architecture, named RAI4.0 (Reference Architecture for I4.0). This aims to delimit and establish the strategies with which this ecosystem is organized, and to qualify and relate the information that is managed, the tasks that process this information, the hardware and software resources that support its transfer, storage and processing as well as the monitoring agents that allow the configuration and management of the system as a whole. This is designed under the design principles mentioned in Sect. 2.1 and, in particular, to meet the following Industry 4.0 requirements:

- The architecture must be operated in a reactive and decentralized mode in order to handle data streams that are generated in the environment.
- The distribution, heterogeneity and scalability of the computational resources required to meet the functional and non-functional requirements of the applications deployed in the environment are conceived as a set of services with the aim of keeping a unique strategy for its management.
- The monitoring of the environment is an essential task since its maintenance and management rely on adaptive and dynamic strategies defined from the levels of use of the resources and the state of operation.

Therefore, the proposed architecture is based on three key principles: Data as a service (DaaS), Platform as a Service (PaaS) and Monitoring as a Service (MaaS). These terms are taken from the terminology of the cloud computing domain [40] although their meaning is not exactly the same. They have in common that the "as a service" term means that the use of the resources must be independent of how these were implemented; however, their location cannot be opaque due to the intrinsic characteristics of an industrial environment where there are a huge heterogeneity of devices installed and connected physically that need to operate complying with real-time restrictions, generally latency, which determines how services should be implemented and where these should be deployed.

Our proposal, currently, allows the design of the industrial environment as well as its management when the system is operational. The reference architecture is defined by means of an abstract and technology-independent model, but this paper also includes a specific implementation based on a set of the Apache project technologies [41] to make the use and scope of our proposal more tangible and easier to understand.

3.1. Data as a Service (DaaS): Data-centric software architecture

At its highest level of abstraction, the industrial environment is defined by the description and characterisation of the data that it manages (see Figure 1). The analysis of the required data and their governance is the starting point of the design of the digital platform. The software components and the hardware resources that comprise the platform are just elements that must be recruited and scaled in order to produce, process or consume the defined global data.

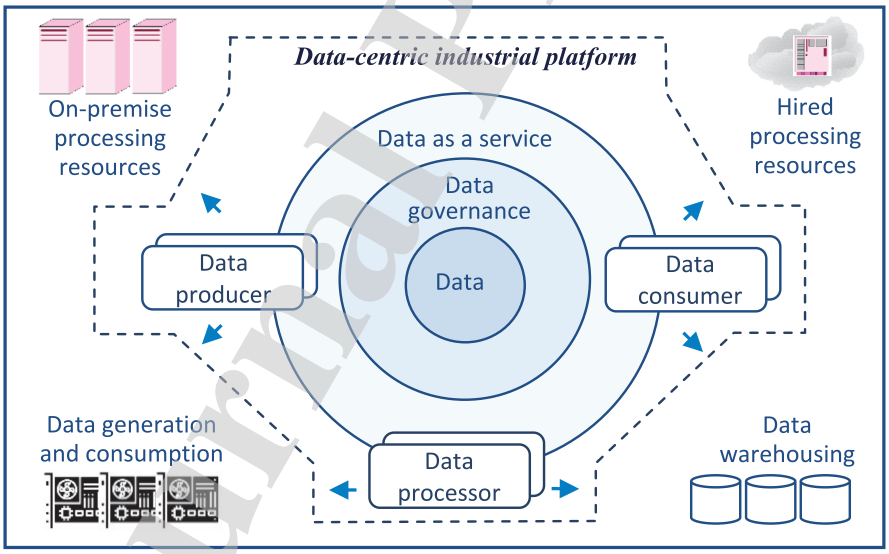

Data-centric industrial platform
On-premise processing resources
Hired processing resources
Data as a service
Data governance
Data
Data producer
Data consumer
Data processor
Data generation and consumption
Data warehousing

Figure 1: Data-centric industrial platform

Data are a strategic asset [42] and thus policies, rules and requirements that industries establish must be considered in the design and configuration of the platform. Governance defines metadata that must be linked to data in the production phase. This is essential for the platform to operate decentralized and autonomously. Metadata are responsible for collecting:

- The nature, semantics and quality of the data required by the agents that process them.
- The estimation of the volume, velocity and variety of the data that have to be managed.
- The security requirements against external risks (authentication, integrity, confidentiality and availability), reliability requirements against system failures and other possible kind of non-functional requirements.
- The metrics to be measured in order to reconfigure the resources dynamically.

Data from industrial environments are organized in topics. These represent flows of instances that describe the same type of information and are managed with the same criteria of persistence, durability, availability, security, etc. Topics are registered in the production platform and their associated metadata constitute the global information available for those services that operate with their instances. The set of topics registered in the system are used to define the workload of the system.

Topic instances are registered as immutable time series. These are written by the producer once and read several times by all consumers subscribed to the topic (publisher-subscriber communication model). The instances' lifecycle is declared in the metadata associated to the topic with policies typically based on volume (number of instances) or time period (temporal window). Instances are distributed among different partitions to meet the requirements of scalability and level of concurrency allowed for their processing. The number of partitions, their location and criteria for the distribution of instances in the partitions are also defined in the metadata associated to each topic.

The applications that provide the functionality in the environment are conceived as workflows of processing tasks that operate following a reactive strategy. A workflow is executed 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, as shown in Figure 2a. The activation of the remaining workflow tasks is triggered according to the synchronization dependencies (pipeline, branch, fork, merge and join) established in the workflow. The workflow can create and record private topics in order to implement control flow mechanisms among tasks. Their lifecycle must coincide with the one of the workflow that creates it.

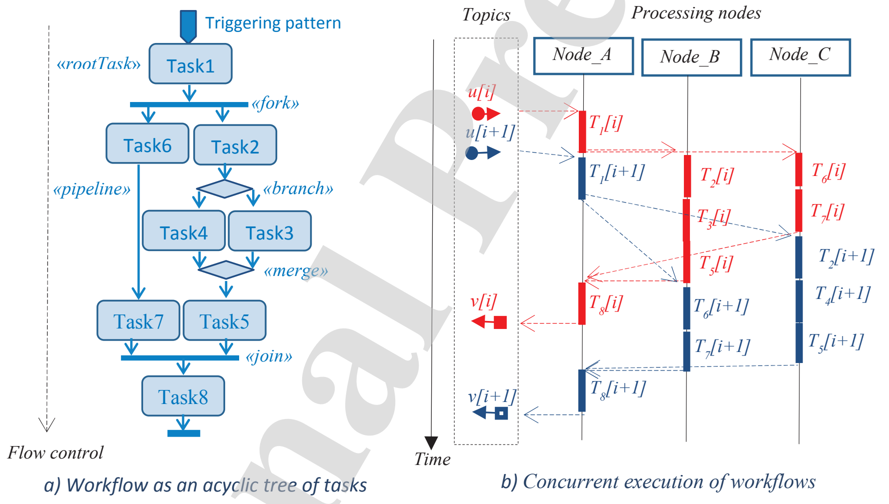


Figure 2: Workflow as a set of processing tasks

Processing tasks are implemented as decoupled software components that are declared as consumers of the topics that trigger them and as producers of the topics that they generate. The instantiation of each task inside a workflow can be replicated in multiple processing nodes under the premise that the code associated to that task must be transferred to the node where data are persisted with the aim of minimizing latency, and not the opposite, data are moved to the node where the task is instantiated. As shown in Figure 2.b, the execution of the tasks of a workflow that correspond to the same occurrence of the triggering pattern can be distributed among different nodes of the platform and, at the same time, tasks corresponding to different triggering events can be concurrently executed in the system. The number of replicas of each task and the level of concurrency with which these can be executed are consequence of both the partitioning strategy and the distributed scheduling policy defined in the topic.

3.2. Platform as a Service (PaaS): distributed, heterogeneous and scalable platforms

The heterogeneity of the computational resources is an intrinsic feature of industrial environments. Its computational capacity is comprised of three folds: i) a great proportion of embedded systems (dew computing), whose general hardware connectivity requirements and operative conditions make them compatible with only those specific resources established by their manufacturers (cyber-physical systems, trains, etc.); ii) a relevant number of resources on the fog aimed at performing the daily operation of the industrial environment (data and dispatching centres, data warehouses, communication centres, etc.), and finally, iii) an increasing number of resources hired on demand on the cloud (cloud computing).

In order to meet Industry 4.0 requirements, a digital platform needs that its resources are interconnected and exchange information, its computational power can be dynamically used according to the needs of each service and its excess computational capacity is orchestrated through a set of middleware services that make it available to the platform.

In the proposed referenced architecture, the computational resources of the platform are organized in physical or virtual nodes and communication networks. The former are responsible for data distribution and processing tasks and, the latter, for sending information among them. We also define the cluster as a set of nodes that provide access to the computational resources of the platform.

As depicted in Figure 3, the agents which manage the data flows have access to the computational resources through the interfaces of the middleware services. Likewise, the configuration of each service establishes the operation modes and use of the resources and, accordingly, the behaviour of the platform.

The reference architecture thus defines the general functionality and the role of the set of available computational services. Each specific implementation must specialize the aspects that depend on the underlying technologies such as the service interface and the configuration parameters. In this paper, as previously mentioned, a platform based on Apache services is defined.

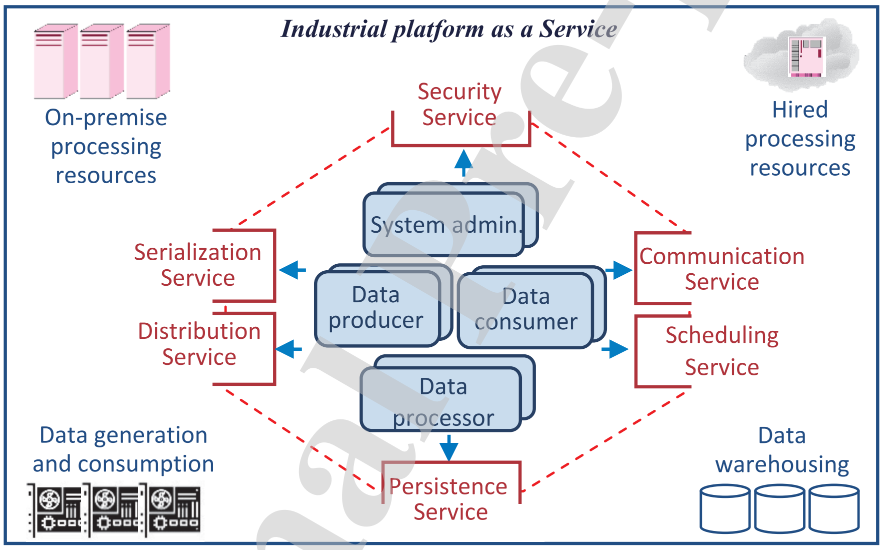


Figure 3: Industrial platform as a service

The services currently defined for accessing and managing the platform resources are:

- **Serialization service**: It implements the serialization of the topic instances required for their transference through the platform. This is independent of the programming languages of the agents that manage the topics. The serialization service defines a language to formalize the structure of the information (schema) and a binary format to codify the serialized information. The schema is associated to topic metadata, being this available during the read/write operations of any instance of the topic from/to the environment. Apache AVRO [43] service is used in our Apache-based implementation of the platform.

- **Distribution service**: It provides secure access to the global information shared by all the resources in the distributed system. This provides both the metadata that qualify each topic and the configuration and coordination data of all distributed services available in the platform. Zookeeper [44] service is used in our proposal. This records the information in an immutable way and assigns a key to each resource with which is possible to identify its successive updates. Furthermore, this offers a synchronization mechanism for notifying the changes in the information that handles to those agents subscribed. Likewise this assigns a key to each update which makes the identification of its successive values easier.

- **Communication service**: Distributed service that registers the topics in the digital platform and sends topic instances among the agents of the environment following a publisher/subscriber strategy. This service is also in charge of: i) manage the lifecycle of the topic instances according to the policy established in the topic metadata; ii) define the partitions for distributing data streams among the available nodes; and, iii) guarantee the balanced distribution of the data among all subscribers. Apache Kafka [45] is the tool selected as communication service in our industrial platform.

- **Scheduling service**: Distributed service that schedules the execution of the processing tasks in the nodes of the cluster in which the required data are available (stored). Three schedulers are offered in our platform: the former is Apache SPARK [46], which dynamically schedules the execution of the processing tasks of a workflow in the nodes that this manages; the second is STORM [47], a real time stream processing which, unlike Spark, operates at tuple level achieving lower latencies, but present other properties that can limit its use, such as, no preserving instance order or guaranteeing strict processing [28]; the latter is a static approach based on delegating the instantiation of the processing tasks to the code responsible of executing the workflow, which is activated by means of a notification mechanism provided by Kafka.

- **Persistence service**: This provides the distributed persistent storage for those instances whose lifecycle exceeds the one supported by the communication service, generally based on volume (number of instances) or time period. The agents that process the instances of a topic are responsible for deciding when these require to be persisted. Big data platforms must integrate both NewSQL and NoSQL data management systems (DBMS) [48] in order to meet the requirements of consistency, availability and partitioning tolerance specified for each topic. Our reference architecture therefore includes three services: MemSQL [49], an in-memory relational DBMS which prioritizes consistency; Apache Cassandra [50], a NoSQL column-family store based on partitioning that distributes data among nodes and dynamically scale; and finally, Neo4j [51], a graph-based DBMS that makes traversal queries easier.

- **Security service**: The distributed security service is in charge of guaranteeing authentication, integrity, confidentiality and availability. Security policies are dynamic and data-oriented, therefore these must be defined in topic metadata, that means, who, how and under which restrictions the instances of the topic can be accessed. In the current implementation, Zookeeper provides mechanisms to manage the access to global shared information and Kafka enables the interchange of topic instances. Nevertheless, both applications rely on a service-oriented strategy, being still their adaptation to a data-centric security strategy pending [52].

3.3. Monitoring as a Service (MaaS): decentralized monitoring system

The proposed platform is decentralized in order to make the horizontal and vertical scaling easier. Any decentralized platform must count with both a mechanism that maintains updated information about, at least, data availability and resources utilisation and, another one that notifies agents about the main changes in the state of the system, so that they can self-reconfigure to meet the specified requirements.

The monitoring service consists of two elements. The former is the distributed monitoring service. This is instantiated in the platform as a set of active brokers in charge of: i) gathering information about the state of the components and services installed in the platform; ii) maintaining updated information about the global state of the system, making the global information available to the components and services that require it and iii) notifying changes of state to those agents interested on them. The latter is a set of monitoring agents associated to both the software components and the brokers of the platform services, which, in turn, provide information (metrics) about their state to the monitoring service. When the components to be monitored are legated, these measures have to be collected by means of other indicators such as the number of I/O messages sent through the network or the use of resources during their execution.

The brokers of the monitoring service are configured and instantiated when the nodes where they are installed are incorporated to the platform. They must be set up taking the impact of the monitoring process on the network traffic and the memory required for storing all the collected data into account. On the other hand, the monitoring agents are configured and instantiated as part of the instantiation of the component or service which they are associated with.

Our digital platform uses Prometheus toolkit [53] as a monitoring tool. This technology provides both the monitoring server, which collects and stores metrics, and a set of monitoring agents, called exporters, which extract and send different kind of metrics to the monitoring server. Prometheus provides a great variety of already implemented exporters for well-known operating systems, databases, etc. Likewise, this offers guidance and libraries to develop customized exporters.

4. RAI4.0 metamodel

This section details our RAI4.0 metamodel, which defines the modelling elements required for describing systems designed according to the RAI4.0 reference architecture. Models compliant to this metamodel can be used by developers to conceive, design, configure and deploy their systems. An example of a tool that uses RAI4.0-compliant models as support for the deployment process of a system is explained in the next section.

We rely on metamodeling [54] because it is a software strategy that allows us to describe the semantics and attributes of the elements that comprise the big data platform as well as the relationships among them.

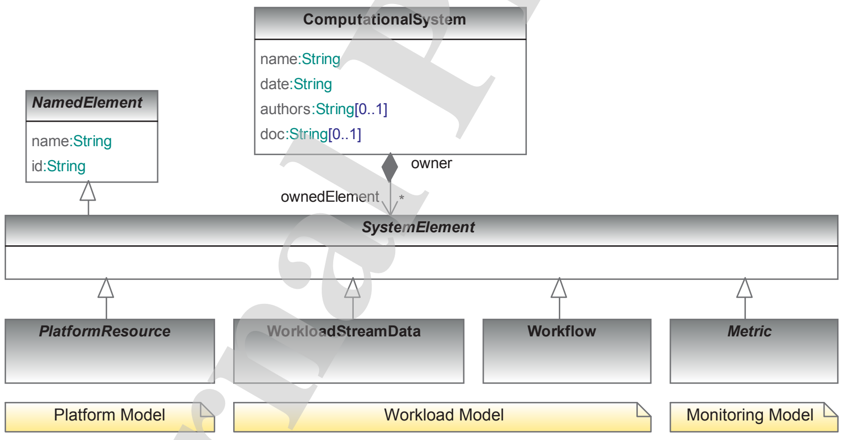


Figure 4: Root elements of the RAI4.0 Metamodel

Each model compliant to the RAI4.0 metamodel is organized around a root element, instance of the `ComputationalSystem` class (see Figure 4). This acts as container of the three sections that comprise the digital platform: the physical and virtual resources and the middleware services (Platform model); the workload of the system, that means, the set of data streams (or topics) that flow in the environment along with the workflows (set of processing tasks) that produce and/or consume these topics and (Workload model); finally, the metrics that must be measured during the execution of the system (Monitoring model). Four root classes have been defined for modelling the elements corresponding to each section:

- `PlatformResource`: this gathers the hierarchy of classes aimed to describe the different resources and services that comprise the digital platform.
- `WorkloadStreamData`: this characterizes the topics or data streams globally defined in the environment.
- `Workflow`: this describes distributed processing flows, i.e., distributed applications that can be added to the environment and executed following a reactive strategy.
- `Metric`: this collects the hierarchy of elements used to specify the indicators to be measured at runtime.

The rest of elements defined are explained along the following subsections.

4.1. Platform model

Figure 5 depicts the classes that inherit from `PlatformResource` and hence, they 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 available in the environment (dew or edge computing) (`PhysicalProcessingNode`) or virtual resources hired on the cloud (`VirtualProcessingNode`). This identifies each node with its IP address and describes some properties such as the number of processors, the amount of memory available or the external IP address in case of virtual machines. PlatformResource elements can be grouped in `ResourceCluster` elements, if needed. `NodeCluster` is a cluster formed only by `ProcessingNode` elements.

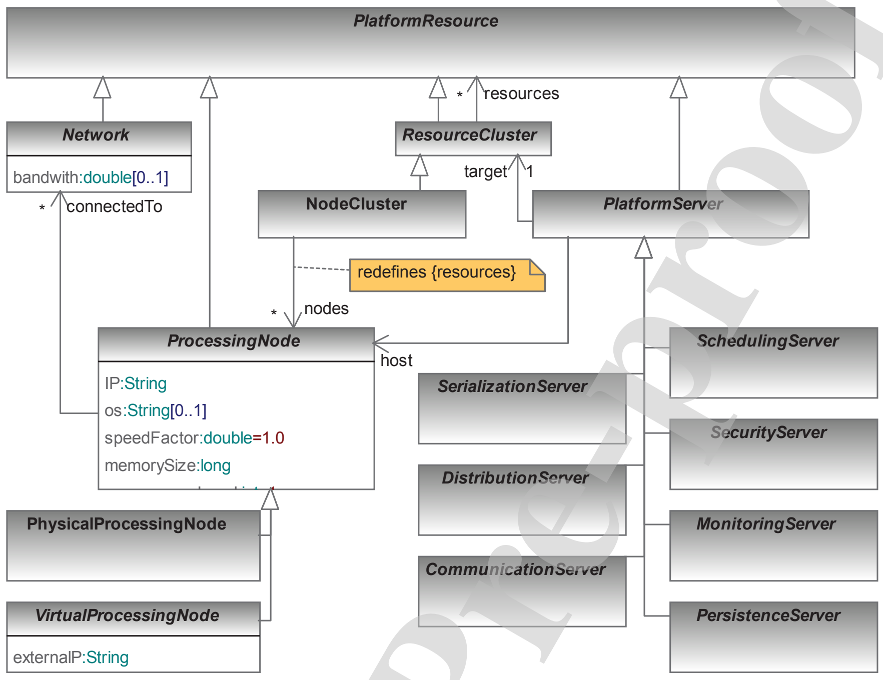


Figure 5: Platform resources hierarchy

`Network` characterizes the effect of the communications' network and the communications' service that manages it. At this level of abstraction, this only includes the bandwith attribute.

`PlatformServer` is a 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. At this level of abstraction only the processing node in which the broker is installed (`host`) and the cluster it serves (`target`) are defined as attributes of the class. As shown in Figure 5, this class is, in turn, specialized in seven abstract classes, one for each type of service described in Section 3: `SerializationServer`, `CommunicationServer`, `DistributionServer`, `SchedulingServer`, `SecurityServer`, `MonitoringServer` and `PersistenceServer`.

As can be seen in Figure 5, all these classes that constitute the essential elements of the platform model are defined as abstract classes, so they must be specialized to incorporate the semantics and specific information of the technologies choosen for their implementation. For the Apache-based implementation proposed in the paper, a Kafka-based communication service (`KafkaServer`), a Zookeeper-based distribution server (`ZookeeperServer`), an Apache Storm scheduling server (`StormServer`) and a Cassandra persistence server (`Cassandra-Server`) are defined in the metamodel (see Figure 6). Likewise, a Prometheus-based monitoring server (`PrometheusServer`) was added. As shown in Figure 6, each class collects the attributes with the specific information required for the configuration and deployment of the corresponding services. This leads to the definition of a great amount of attributes per class due to the inherent complexity of each element. Most of them have default values assigned, so the user only has to override those that need to be adjusted on each deployment scenario.

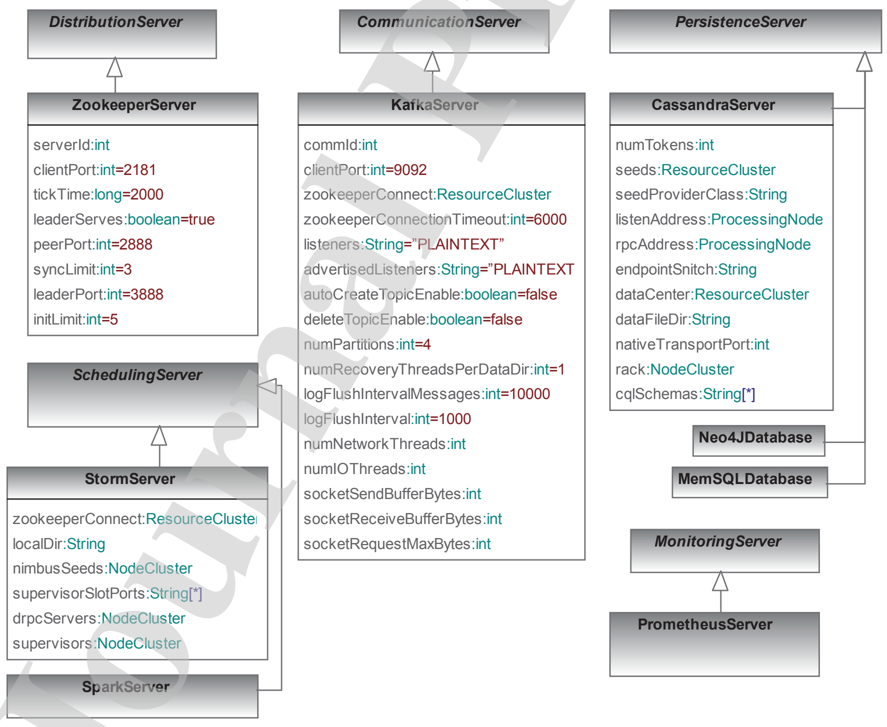


Figure 6: Specialization of PlatformServer classes for Apache-based implementation

4.2. Workload model

As said previously, two elements contribute to the definition of the workload of the system: the set of topics that flows through the environment and the set of processing workflows that uses (produces/consumes/transforms) these topics.

The RAI4.0 reference architecture places the topics globally defined in the system as central elements of the system architecture. They are not simply declarative elements, but they have an effective implementation in the platform: they must be registered, configured and deployed on it. The information registered of each topic is essential for the functionality and behaviour of all the services and components of the system. The root class that represents global topics is `WorkloadStreamData`. As shown in Figure 7, this inherits from a root abstract class called `StreamData`, which defines attributes common to any type of topic, such as the size (`messageSize`), the number of partitions in which the topic instances are distributed (`numPartitions`) or the number of copies available of each topic instance (`numReplication`). The `WorkloadStreamData` class may need some kind of extension to include characteristics dependent on the underlying technologies, as it happens in our case, where as shown in Figure 7, the `KafkaWorkloadStreamData` class is defined to include those aspects specific to the way Kafka manages topics (basically, the `holder` attribute is redefined to reference to a `KafkaServer` instance).

There are another kind of topics that can be defined in the model, whose purpose is to support the flow control of the tasks executed by the same workflow. They are modelled by means of the class `WorkflowStreamData`. This also inherits from `StreamData`. All topics are managed in the same mode but `WorkloadStreamData` instances are directly defined as an aggregation of the computational system (as part of its workload, see Figure 4), whereas `WorkflowStreamData` instances are defined in their respective workflows as can be observed in Figure 8.

`WorkflowStreamData` is, in turn, specialized in two classes: i) `FlowStreamData` that describes the data flow generated by a workflow and used to manage the internal flow of control among the internal tasks of this by means of the source

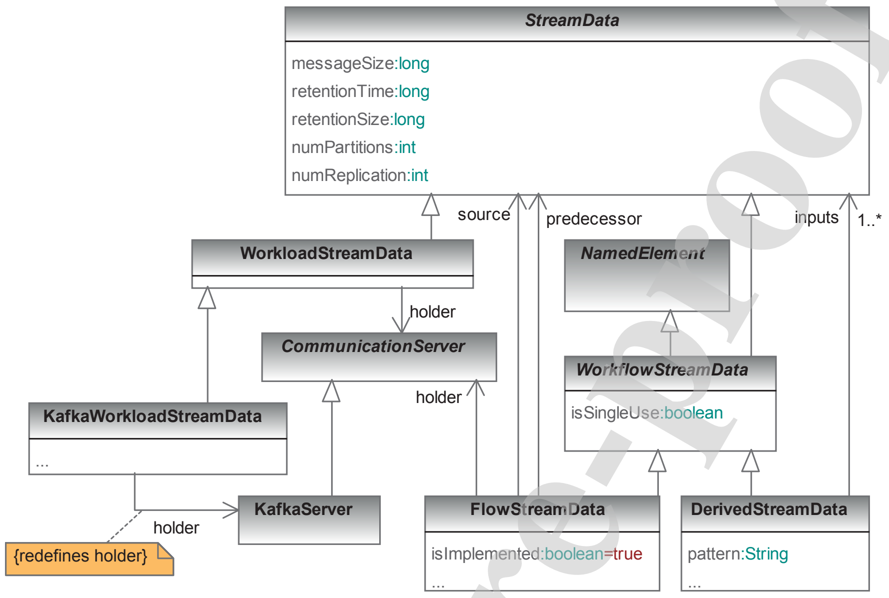


Figure 7: Class hierarchy for describing topics or data flows

and predecessor attributes; ii) `DerivedStreamData` that represents a virtual data flow that results from applying a filter function (pattern) to one or more data flows (inputs).

On the other hand, the processing tasks that exploit the topics available in the environment are organized in `Workflow` elements, whose main structure is shown in Figure 8. The execution of a workflow is triggered in response to the occurrence of a topic (`rootTask.triggerTopic`), which can be of `WorkloadStreamData` or `DerivedDataStream` type. Next, a set of processing tasks (`ownedTasks`) related by the control flow are executed in the system. 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. 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 topic that triggers a task can be a derived topic, thus complex control flows using branching, forking, joining, merging, etc. patterns must be managed inside the workflow. The service responsible for the assignment of resources and the launching of the execution of the tasks is assigned to the scheduler attribute of the workflow class.

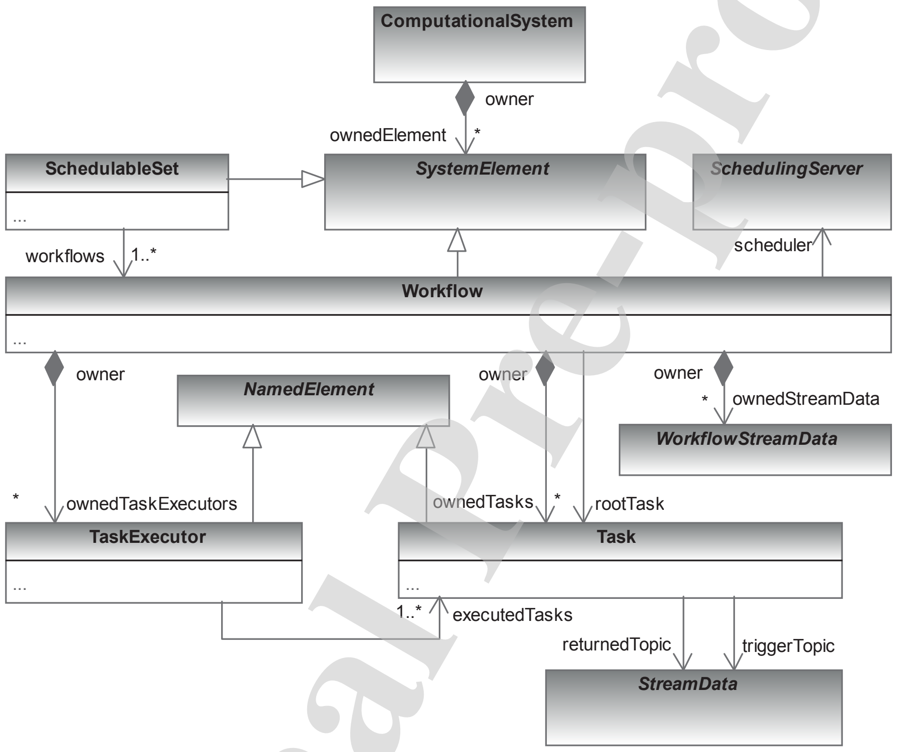


Figure 8: Workflow description

580 Finally, the `TaskExecutor` element represents a software artifact that encapsulates the code of a set of tasks (`executedTasks`) of the workflow. It is only used in the case the static scheduling approach explained in Section 3 is being applied, in which the deployment and instantiation of the tasks must be addressed explicitly during the workflow instantiation. On the contrary, when
585 a dynamic strategy, e.g. based on Storm is applied, the scheduling service is responsible for the deployment of the tasks.

4.3. Monitoring model

Monitoring the state, behaviour and performance of the system relies on the definition of the set of metrics to be collected and the execution of two complementary processes. The former is the monitoring agent responsible for obtaining the measures. To this end, it must be instantiated together with the component that must be tracked. The latter aims to gather, integrate and register all the information provided by the different monitoring agents deployed in the system for its subsequent analysis. This process is supported by a distributed monitoring service (`MonitoringServer` in the metamodel).

Figure 9 shows the main classes defined in RAI4.0 metamodel to support the monitoring strategy. The metrics to be measured are modelled through classes that inherit from the `Metric` class and are part of the `ComputationalSystem`. At this abstraction level, the attributes that can be defined are two: the element to which the metric is applied (`target`) and the agent in charge of providing the measure (`meter`). The `Meter` class describes the information required for the configuration, deployment and launching of the agents in charge of collecting the measures required to evaluate the metrics. At this level of abstraction only the association with the corresponding `MonitoringServer` must be defined.

Extensions of `MonitoringServer` and `Meter` classes will be defined for each specific technology used for the monitoring task by adding the corresponding required attributes. Our implementation uses Prometheus technology, therefore the following classes are added to the metamodel:

- `PrometheusServer` class, which models the Prometheus server broker.

- `PrometheusMeter` class, which acts as root class for all the kind of measuring agents that can be used. This includes the common attributes to all of them. Since Prometheus uses a polling strategy for the communication between the server and its agents, these attributes are the port through which the agent provides the measures to its corresponding Prometheus server (`monitoringPort`) and the frequency with which the server reads the specified measures (`monitoringTime`).

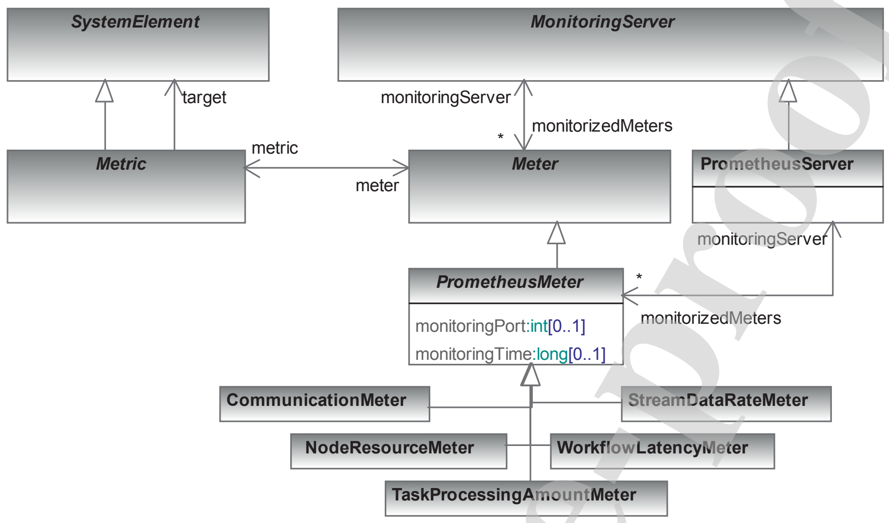


Figure 9: Classes used to describe the monitoring model based on Prometheus

- A set of classes that extends `PrometheusMeter` with the aim of modelling the different kind of monitoring agents currently supported in the platform. For instance, the `NodeResourceMeter` class represents a monitoring agent based on the NodeExporter [55] provided by Prometheus, which allows to extract measures about utilisation and memory consumption of a Linux-based processing node.

As a consequence of the fact that there is a wide, almost unlimited, variety of metrics that can be potentially defined for a computational system, each new kind of metric requires the definition of an extension of the `Metric` class. In the current version of the metamodel, see Figure 10, the defined metrics are restricted to aspects related to timing behaviour and resources utilisation. Other metrics related to energy consumption, reliability, availability, etc. can be included if there are monitoring services that provide the required measures. For each specialized `Metric`, its corresponding target and meter attributes must be redefined to refer to the correct type of element according to the nature of the metric. For example, the `ProcessingNodeUtilization` metric must be associated to a `ProcessingNode` as target element and must be measured by a

NodeResourceMeter, which represents an agent with capacity to measure about the utilisation of resources in a node.

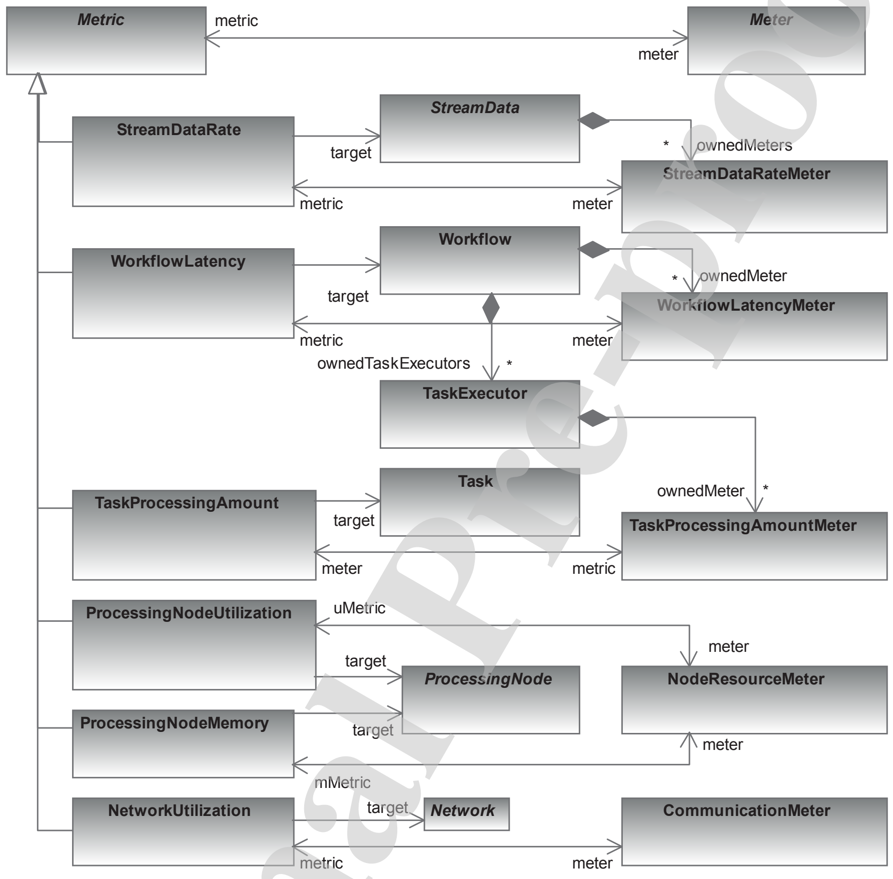


Figure 10: Classes hierarchy for representing metrics and monitoring agents

4.4. Instantiation model

One of the principles that guides the RAI4.0 reference architecture is that all its elements, services and workflows, are decoupled components, i.e., they can be independently deployed and instantiated, being each component responsible of inquiring the information required to be installed to the environment services. From that moment, these operate reactively, that means, they remain suspended until the corresponding activation event occurs (the reception of a message through a port in the case of the platform services or the occurrence of an instance of a topic in the case of a workflow) and then, they execute their corresponding code and offer as a result new events that transmit, in turn, new data or control flow topics.

Therefore, all the deployable components are distributed as self-contained software artifacts that encapsulate their corresponding code and are available in the nodes where they can be potentially instantiated. So, they can be invoked once or more times according to the system description. The information required to configure and launch these software artifacts is technology-dependent, but some common structural elements are gathered in the RAI4.0 Metamodel. Figure 11 shows the deployable components, which are: the services of the platform (instances of `PlatformServer`), the `Workflow` elements and their corresponding (optional) `TaskExecutor` elements, the `WorkloadStreamData` and the `Meter` elements. All of them inherit the common attributes and functionality that is necessary to instantiate the component from the `SystemComponent` class. Some of these attributes are the location and name of the corresponding software artifact (`artifactLocator` and `artifactName`, respectively), the arguments that could be required to be provided at instantiation time (`arguments`) or the path of the directory where the generated configuration files and launch scripts must be stored in the launching node (`configDir` and `scriptsDir`), among others.

Next, we describe the tool developed to take advantage of this metamodel in twofolds: i) to design and configure applications under a data-centric architecture and ii) to automate the deployment and execution of these applications.

5. Deployment tool for the configuration and deployment of RAI4.0 systems

One of the advantages of model-based development is the possibility of building tools that support enterprise designers, software developers and IT managers in the conception, design, assessment or deployment of complex systems. Among

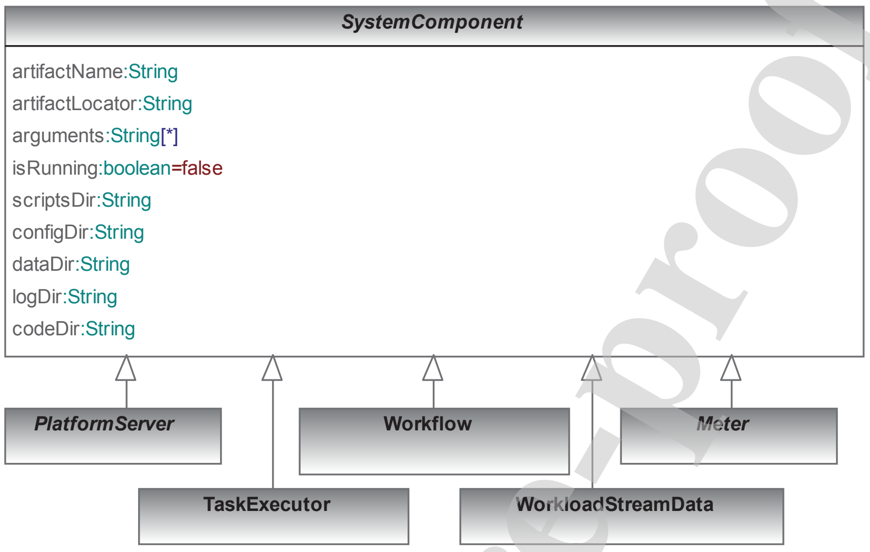


Figure 11: Deployable elements in the RAI4.0 metamodel

all of them, we began with the development of a deployment tool due to the complexity involved in the configuration, deployment and launching of the initial digital platform as well as in its reconfiguration and in the installation/uninstallation of the workflows.

In order to point out the value of our RAI4.0 deployment tool and show its usefulness, next we summarize the steps typically involved in the execution of a new workflow or service in the platform:

1. Create or modify the configuration files of the workflow/service itself and other involved resources.
2. Register new topics.
3. Define the monitoring metrics to read during the execution of the workflow/service and generate the infrastructure required to collect the corresponding measures.
4. Launch the software artifacts that implement the workflow/service and the metrics infrastructure.

As can be observed, there are many elements to be configured, each one with several parameters to be established. Thus, a tool that automates the configuration and deployment of applications designed according to the RAI4.0 metamodel will help to reduce errors and save time and money. 690

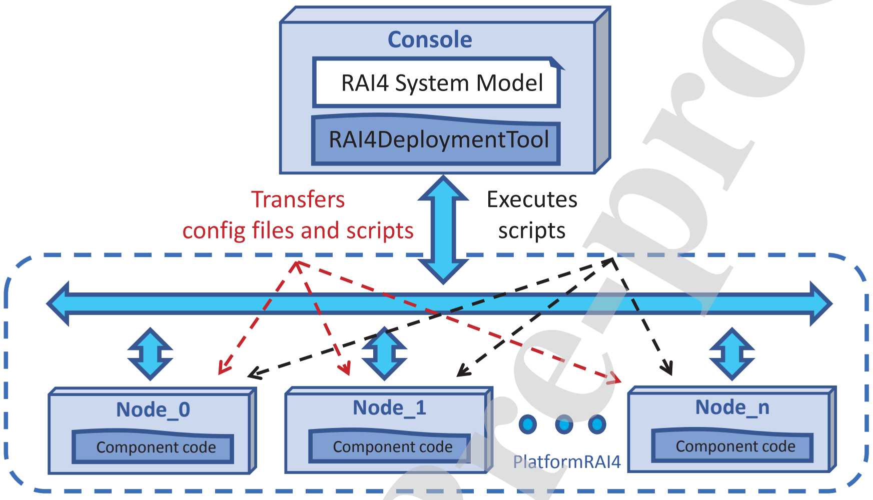


Figure 12: Elements that participate in the configuration and deployment process

Figure 12 shows the main elements that participate in the process of configuring, deploying and launching an RAI4.0-compliant system. The RAI4.0 Deployment tool processes the model of the system (compliant to the RAI4.0 Metamodel) and, as a first step, generates all the required configuration files and launch scripts. Then, this sends these files to their corresponding 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, that means, the RAI4.0 Deployment tool 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. 695 700

RAI4.0 Deployment tool requires certain information to operate, this is gathered in attributes and methods that are defined in some of the classes of the metamodel as can be observed in Figure 13.

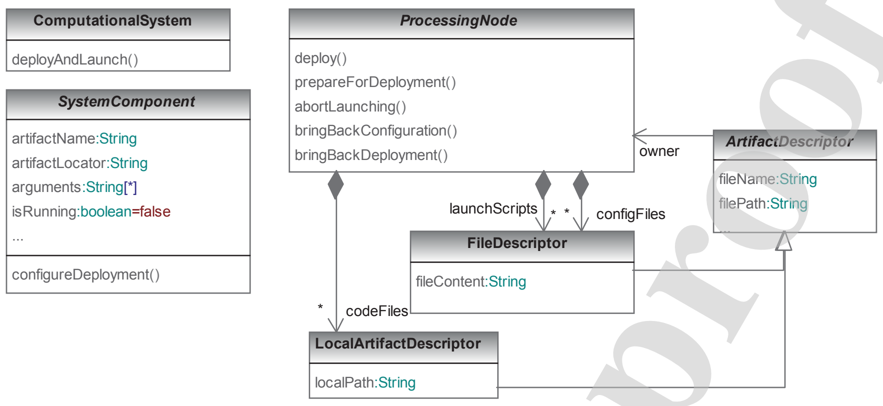


Figure 13: Attributes and methods included in the metamodel to support the deployment tool

The root class of the model, `ComputationalSystem`, defines a method called `deployAndLaunch` that launches the whole deployment and launching process. The implementation of this method, modelled in the sequence diagram depicted in Figure 14, entails the following steps:

- Invocation of the `prepareForDeployment` method on each `ProcessingNode` element contained in the model. This method initializes the `configFiles` and `launchScripts` attributes, which are used to store all the configuration information and scripts generated during the configuration process by all the components that need be executed on the node.
- Invocation of the `configureDeployment` method on each deployable component. This method is inherited from the `SystemComponent` class, which represents elements that can be independently deployed in the system. 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 physical 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. As an example of this overriding process, the implementation of the `configureDeployment`

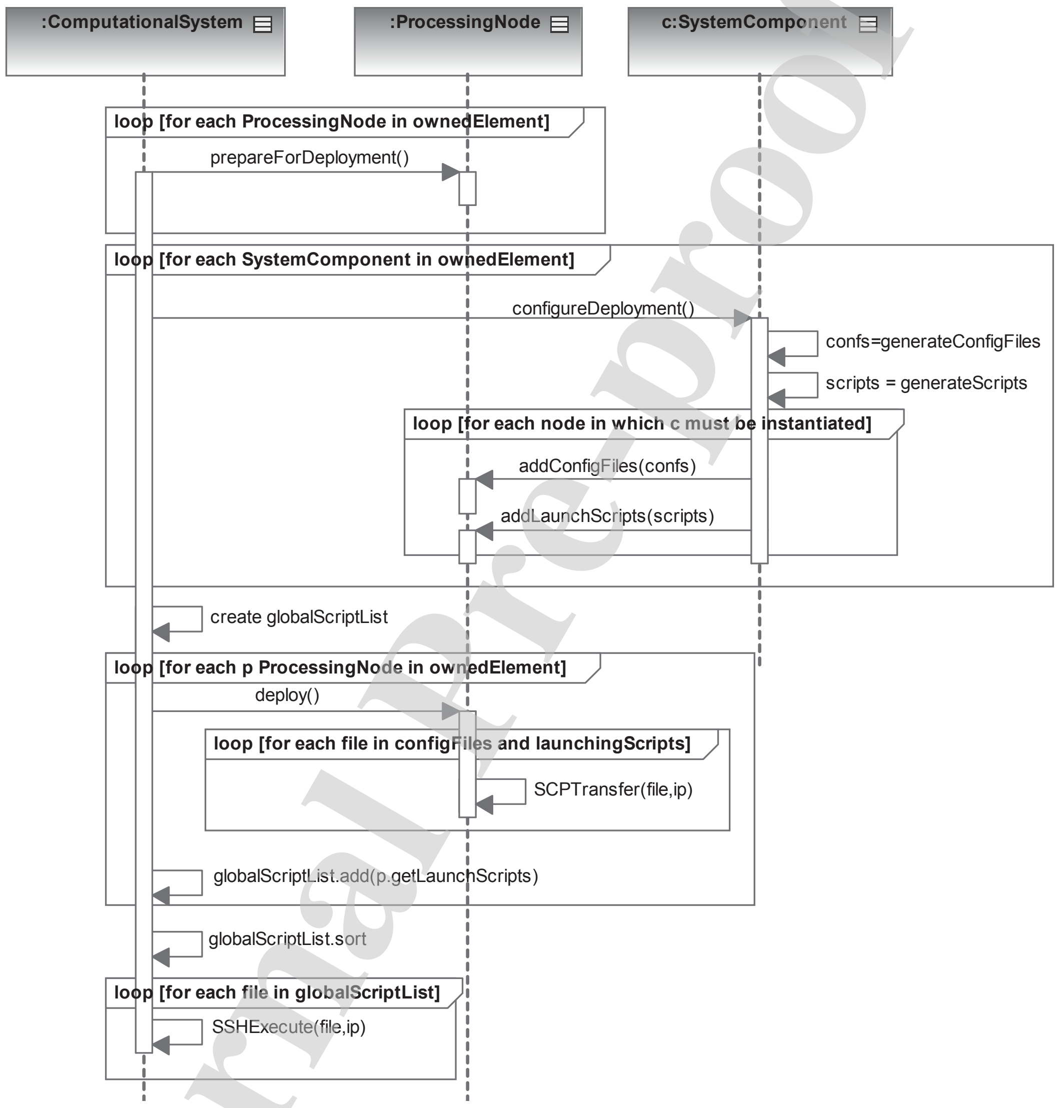
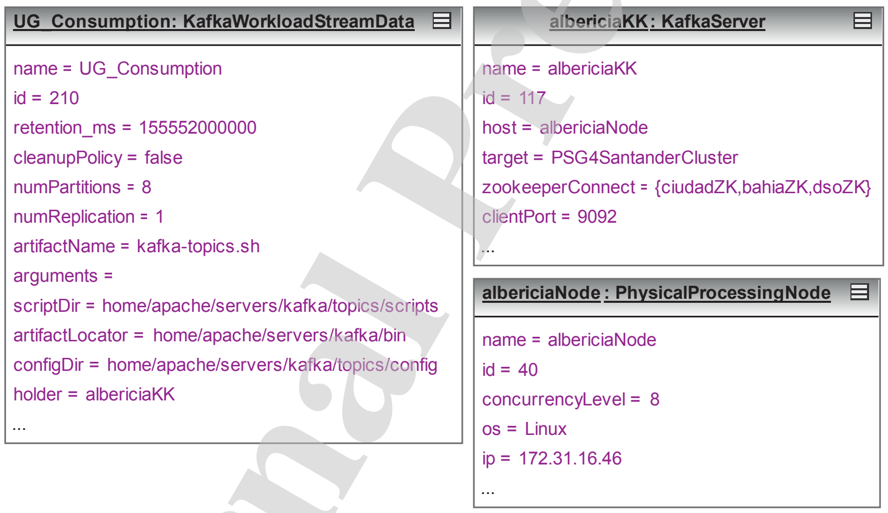

:ComputationalSystem
:ProcessingNode
c:SystemComponent
loop [for each ProcessingNode in ownedElement]
prepareForDeployment()
loop [for each SystemComponent in ownedElement]
configureDeployment()
confs=generateConfigFiles
scripts = generateScripts
loop [for each node in which c must be instantiated]
addConfigFiles(confs)
addLaunchScripts(scripts)
create globalScriptList
loop [for each p ProcessingNode in ownedElement]
deploy()
loop [for each file in configFiles and launchingScripts]
SCPTransfer(file,ip)
globalScriptList.add(p.getLaunchScripts)
globalScriptList.sort
loop [for each file in globalScriptList]
SSHExecute(file,ip)

Figure 14: Model of the deployAndLaunch method of the ComputationalSystem class

method in a `KafkaWorkloadStreamData` element is explained. Figure 15 gathers one instance of a `KafkaWorkloadStreamData` of the case study exposed in Section 6. The instantiation of this topic requires the execution 725 of the following command in the Kafka server where this topic must be registered:

```
home/apache/servers/kafka/bin/kafka−topics.sh −−create −−bootstrap
    −server XXX.XX.XX.46:9092 −−partitions 8 −−replication−factor
    1 −−topic UGConsumption
```

This command is embedded in a script file which is generated by the `configureDeployment` method. The `WorkloadStreamData` instance has access to all the information needed to generate the command, such as the name of the topic, the number of partitions, or the port of the KafkaServer used for the instantiation (the IP address of the underlying processing node is accessible through the host attribute). If any information is missing, a `ConfigurationException` is thrown.



UG_Consumption: KafkaWorkloadStreamData
name = UG_Consumption
id = 210
retention_ms = 155552000000
cleanupPolicy = false
numPartitions = 8
numReplication = 1
artifactName = kafka-topics.sh
arguments =
scriptDir = home/apache/servers/kafka/topics/scripts
artifactLocator = home/apache/servers/kafka/bin
configDir = home/apache/servers/kafka/topics/config
holder = albericiaKK
...

albericiaKK: KafkaServer
name = albericiaKK
id = 117
host = albericiaNode
target = PSG4SantanderCluster
zookeeperConnect = {ciudadZK,bahiaZK,dsoZK}
clientPort = 9092
...

albericiaNode : PhysicalProcessingNode
name = albericiaNode
id = 40
concurrencyLevel = 8
os = Linux
ip = 172.31.16.46
...

Figure 15: Example of the configuration and deployment information required to register a topic

- Invocation of the `deploy` method on each `ProcessingNode` element. This method transfers, using SCP, the configuration files stored in the `config-Files` attribute and the scripts corresponding to the `launchScripts` attribute to the corresponding physical node. This method can raise `Deploy-mentException` in case an error is found in the transfer of a file. If this happens, the process is stopped and the operator is informed of the error. The tool is responsible for returning the system to its initial state.

- Remote execution, using SSH, of the set of scripts previously transferred to the physical nodes, which will launch the execution of the corresponding components using the configuration data also transferred in the previous step. This method can raise the `LaunchingException`, being the tool also responsible for returning to the initial state and notifying the user the error. 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.

The current implementation of the RAI4.0 Deployment tool is based on Eclipse/EMF [56]. The RAI4.0 Metamodel has been formalized using Ecore and the tool has been written in Java. This is triggered as a common Java application, using the RAI4.0 model as input argument. In a future, it could be integrated in an Eclipse Rich Application through a contextual menu associated to the models compliant to the RAI4.0 Metamodel. Both the tool implementation and the metamodel are available at [57].

6. Case studies

Two cases studies are included in the paper to show the applicability of the proposal. The former is a simple academic case, which was used as first proof of concept. It models a real-time application that processes pollution data of a city. The latter one is a much more complex and complete Industry 4.0 case study, which emulates the management of the electric power distribution of a city using a smart grid platform.

6.1. Pollution case study

This application aims to analyse and process a data stream from pollution sensors installed in a smart city. Data are published in Kafka, processed by Storm and persisted in Cassandra for building prediction models in the future.

It must be remembered the reference architecture organizes the system in three independent but complementary sections: the workload of the system, the platform resources available for execution, and the metrics monitoring. So, the first step to elaborate such a model consists in defining and qualifying the topics that flow in the environment. In this case, there is a single topic, called `Pollution`, which gathers the ingested pollution data. Then, the workflows that produce, consume or transform the information (topic instances) must be added to the model. The configuration and deployment information 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. Figure 16 depicts a schema of the behaviour of the `PollutionProcessing` workflow, triggered by the `Pollution` topic. 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 seconds). 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.

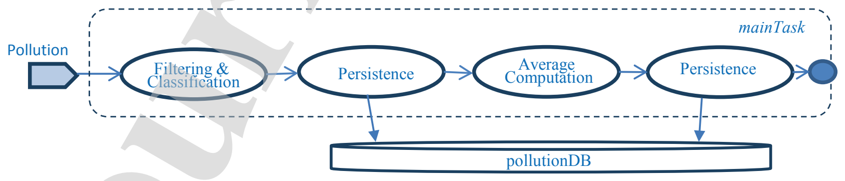


Figure 16: PollutionProcessing workflow schema

The second step addresses the definition of the platform resources. 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, their corresponding instances are added to the model, configured and related with the corresponding elements already present in the model: the `KafkaServer` instance is assigned to the `holder` attribute of the `Pollution` topic, since Kafka is responsible for its management, and the `StormServer` instance is assigned as scheduler of the `PollutionProcessing` workflow, since its `mainTask` is going to be deployed as an Storm topology. Figure 17 represents the main instances of the pollution model and how they are related.

For sake of simplicity, the monitoring section is omitted in this case.

Next, we must specify the deployment platform. In order to show the reutilisation and automation benefits the proposed metamodel and tool provide, we describe three different deployment scenarios for this same application:

- First, a monoprocessor deployment scenario. With that purpose, a single instance of `PhysicalProcessingNode`, called `MainNode`, is added to the model and characterized according to the node properties (IP, speed, etc.). Then, this node is referenced from all the middleware services instances through their corresponding`host` attribute, as it was shown in Figure 17. Once we have the final model, it can be used as input of the deployment tool, which will generate all the required configuration files and create and execute all the required launch scripts, in the correct order (for example, to launch a Kafka server, Zookeeper must be already available). In this specific case, the tool will generate four configuration files (one for each service) and six scripts: the one for registering the Pollution topic on Kafka, the one for launching the workflow on Storm and one script for launching each server (Kafka, Zookeeper, Storm and Cassandra). The files generated by the tool for this example (and for the following ones) are also available in the tool repository [57], under the `caseStudies` folder.

- Second, a distributed deployment in which each service is deployed in a

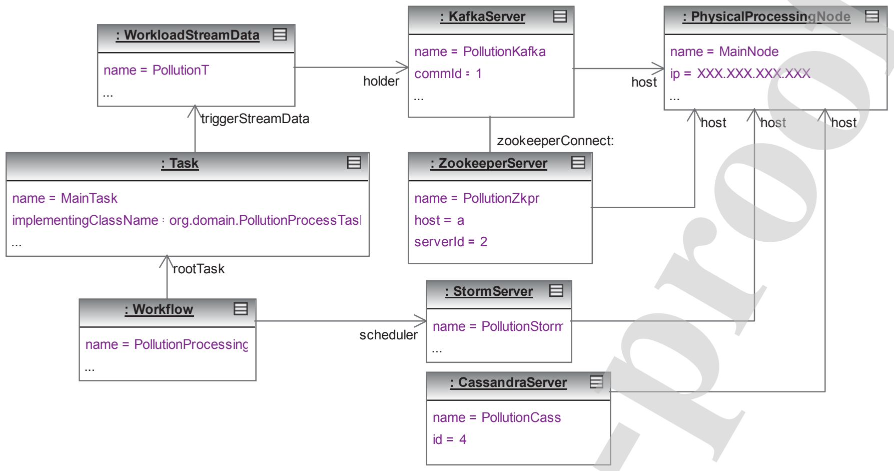


Figure 17: Instances of the Pollution example on a monoprocessor scenario

different node. As it is shown in Figure 18 the only modification required in the model in order to get a distributed deployment scenario consists in adding the corresponding four `PhysicalProcessingNode` instances, and change the `host` reference of each server instance to the node in which it will be deployed. The rest of the model requires no modifications. The number of configuration files and scripts generated for this setting is the same as the previous one, but each one is sent and executed in its corresponding node.

- Finally, a cloud distributed deployment. To launch this same application on the cloud instead of on premise platform, the only change required in the model consists in transforming the previous `PhysicalProcessingNode` instances into `VirtualProcessingNode` instances and assigning them their corresponding external IP addresses. Implementing this change in this same case took less than seven minutes to a test student. The number of files and scripts remains unchanged regarding the two previous cases.

Hence, this simple case study evidences the reutilisation capacity that is provided by the metamodel proposed and the benefits of using the associated

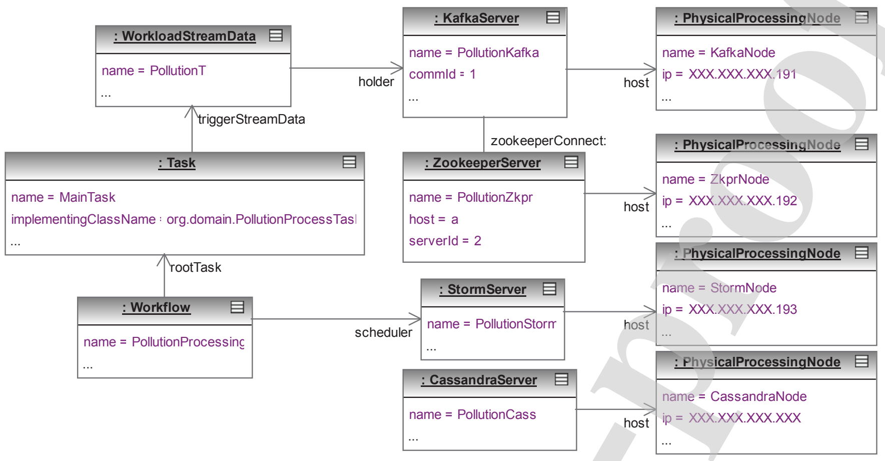


Figure 18: Instances of the Pollution example on a distributed scenario

deployment tool to automate the process and save time. Suffice it to point out that changing from a monoprocessor to a distributed deployment, only a slight modification of the model is required. It is enough with defining the new hosts and changing the host attribute in the corresponding servers, which is insignificant in comparison with having to write the corresponding scripts and configuration files on each node and launch the scripts by hand.

6.2. Electric power consumption management on a PowerSmartCity

The second case gathers the management of an electric power distribution of a city. This is exposed in order to show the modelling capacity that the metamodel provides and justify the need of a deployment tool that automates the process of setting and launching the system.

Figure 19 depicts schematically the elements that comprise a general electric power distribution network of a medium-size city. They are organized as a redundant and real-time reconfigurable power distribution network in order to be robust against the failure of any of its elements.

All elements in the industrial environment, see Figure 20, are controlled and managed by different types of computational resources interconnected through

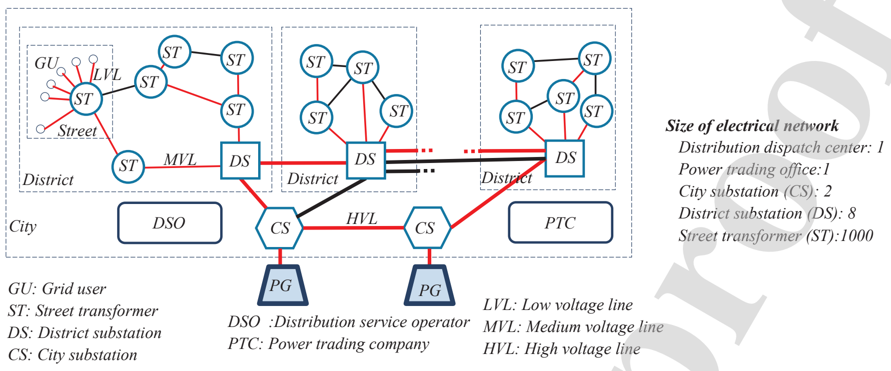


Figure 19: Structure of the electric power distribution network used in our case study

communication networks. The computational power is very high but also heterogeneous. In this example, there are 250.000 smart meters installed at users' homes and 1.000 nodes installed at street level distribution transformers. They together provide a very flexible and efficient control of the accesses to the electric power network, and, at the same time, generate very valuable information at real-time. These are embedded processors (dew computing) with capacity for generating and consuming data, however they cannot support the installation of brokers that access to the middleware services. On the other hand, there are 12 servers installed for the control of the district substations, the city substations and the distribution dispatch centre (fog computing) which are mainly dedicated to the supervision and control of the electric equipments. Although they can also offer some of their spare computational capacity to process tasks of data analysis. In addition, if it were necessary to increase this capacity, the platform could hire external computing resources (cloud computing).

Next, the specification of this case study based on our metamodel is described. We follow the same steps as in the previous example. First of all, we define and qualify the topics that flow in the environment. Table 1 lists these topics. Each topic is described and registered in the platform by means of a set of metadata, which constitutes the basis for defining the resources that must be assigned for its management. For instance, Table 2 details the `UG_Consumption`

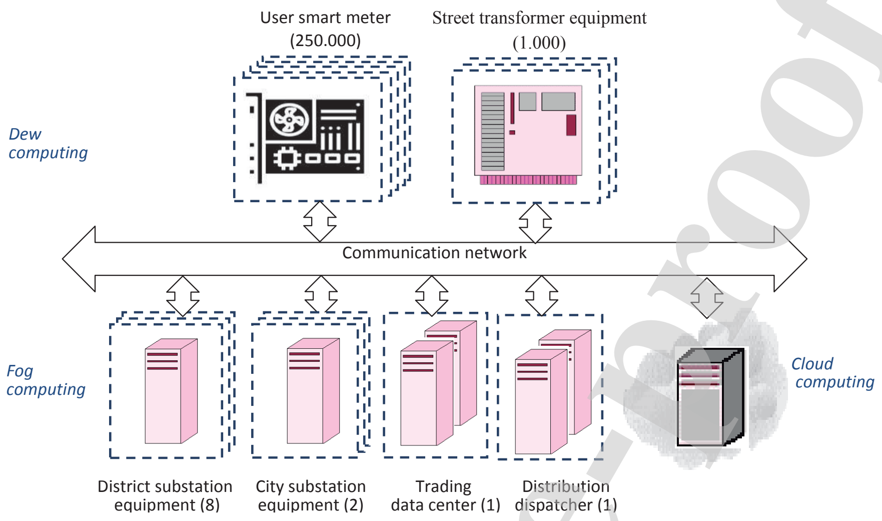

User smart meter (250.000)
Street transformer equipment (1.000)
Dew computing
Communication network
Fog computing
Cloud computing
District substation equipment (8)
City substation equipment (2)
Trading data center (1)
Distribution dispatcher (1)

Figure 20: Power smart grid elements

topic, which is formalized by means of a `KafkaWorkloadStreamData` instance. Following this schema, the remaining 20 topics of the example should be defined.

880 Then, we describe the workflows that produce, consume or transform the information (topic instances) available in the platform. Figure 21 depicts a schema of the behaviour of one of them, the `DailyConsumption` workflow, whose aim is the daily reading of the consumed electricity and the calculation of the profile of hourly consumption of each user grid. This workflow is triggered by the 885 occurrence of instances of the `UG_Consumption` topic. The workflow has two processing tasks, `powerTask` and `profileTask`, executed on pipeline. `PowerTask` registers the energy consumed by the client in the `powerBD` database, whereas `profileTask` writes down the user's consumption profile in the `profileBD` database. The second task reads the 24 instances of the `UG_Status` topic (in890 puts) generated each day, which contain the mean consumed power by each client. Just in case there were data missing, the task would estimate this consumption using historical data from the `profileBD` database. The control flow between the two tasks is performed through the private topics of the workflow named `powerStored` and `profileStored`.

Table 1: Topics defined in PowerSmartGrid

Source	**Downstream topics**	**Source**	**Upstream topics**
Dispatch Center	UG_Configuration	District Substation	DS_Identification
	ST_Configuration		DS_Status
	DS_Configuration		DS_Alarm
	CS_Configuration	Street Transformer	ST_Identification
	UG_Constraints		ST_Status
	TS_Connectivity		ST_Alarm
	DSConnectivity	User Grid	UG_Identification
	CS_Connectivity		UG_Status
City Substation	CS_Identification		UG_Consumption
	CS_Status		DS_Alam
	CS_Alarm		

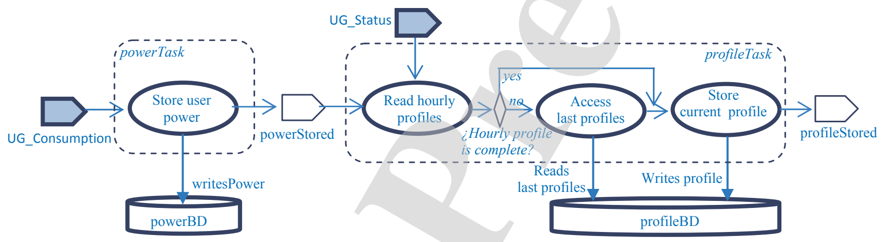


Figure 21: Reactive task flow of DailyConsumption workflow

We proceed now to identify the physical and virtual resources along with the configuration of the underlying middleware services. Likewise, we associate each resource and service with the metrics to be measured and the instrumentation required for this purpose. Figure 22 represents an excerpt of the complete model, with the instances included in it to describe the information required for the configuration and deployment of the services installed in one of the processing nodes, named `cityNode`. Two brokers are installed on it, a Kafka server (`cityKK`) and a Zookeeper server (`cityZK`). For simplicity, only these two instances are shown in detail (attributes whose default values are not overriden are not shown). We can observe, for instance, how the host attribute of both instances refers to the `cityNode` instance. This partial model includes also the `NodeResourceMeter` instance, `cityMeter`, which represents the monitoring

Table 2: Description of UG_Consumption topic

Name	**UG_Consumption**
Description	It describes the energy (Kw/h) consumed by a UG in an official day. It is necessary for the billing process.
Source	User Grid
AVRO schema	{"namespace": "psg4c", "type": "record", "name": "UG_Consumption", "fields": ["name": "pcId", "type": "long", "name": "clientID", "type": "long", "name": "startCount","type": "long", "name": "fixedEnergy","type": "float", "name": "optionalEnergy","type": "float", "name": "reactiveEnergy","type": "float"]}
Confidence level	Maximum (if data exist, it is absolutely reliable)
Availability level	Maximum (if data is not available, alarm is raised)
Confidence level	TRADING + OWNER + MAINTENANCE
Generation period	86400000 (1 day)
Generation offset	86400000 (1 day)
Persistence time	3456000000 (40 days)
Sharding level	8 (one partition for district)
Replication level	3 (supports one node failed)

agent associated to the `cityNode` node and the metric `cityUtilization` of the `ProcessingNodeUtilization` class.

Figure 23 summarizes the instances involved in modelling the configuration and deployment of the `DailyConsumption` workflow. Again, only the `dailyConsumption` workflow instance is shown with all its assigned attributes. The remaining instances involved are only named: the `dsRackScheduler` scheduler (of `NodeScheduler` type), which indicates that a static scheduling approach is applied; the `TaskExecutor` instance, called `dailyConsumptionExec`, which is required to instantiate the tasks marked by the scheduling strategy chosen; the two Tasks, `powerTask` and `profileTask`; the two internal topics to support the control flow, `powerStored` and `profileStored`; and finally, the instances corresponding to the monitoring agent used to measure the workflow latency, `dailyConsumptionMeter`, and its corresponding metric, `dailyConsumption-`

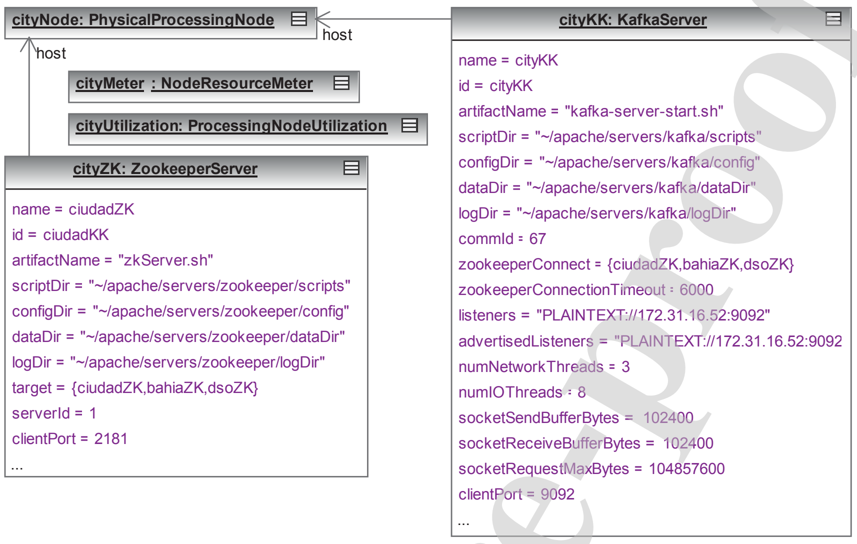

cityNode: PhysicalProcessingNode
host
host
cityMeter : NodeResourceMeter
cityUtilization: ProcessingNodeUtilization
cityZK: ZookeeperServer
name = ciudadZK
id = ciudadKK
artifactName = "zkServer.sh"
scriptDir = "~/apache/servers/zookeeper/scripts"
configDir = "~/apache/servers/zookeeper/config"
dataDir = "~/apache/servers/zookeeper/dataDir"
logDir = "~/apache/servers/zookeeper/logDir"
target = {ciudadZK,bahiaZK,dsoZK}
serverId = 1
clientPort = 2181
...
cityKK: KafkaServer
name = cityKK
id = cityKK
artifactName = "kafka-server-start.sh"
scriptDir = "~/apache/servers/kafka/scripts"
configDir = "~/apache/servers/kafka/config"
dataDir = "~/apache/servers/kafka/dataDir"
logDir = "~/apache/servers/kafka/logDir"
commId = 67
zookeeperConnect = {ciudadZK,bahiaZK,dsoZK}
zookeeperConnectionTimeout = 6000
listeners = "PLAINTEXT://172.31.16.52:9092"
advertisedListeners = "PLAINTEXT://172.31.16.52:9092
numNetworkThreads = 3
numIOThreads = 8
socketSendBufferBytes = 102400
socketReceiveBufferBytes = 102400
socketRequestMaxBytes = 104857600
clientPort = 9092
...

Figure 22: Configuration and deployment model of the "cityNode" processing node

Latency.

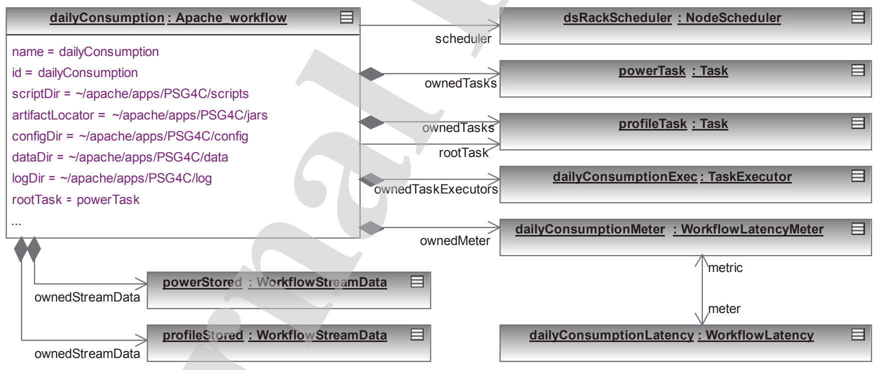

dailyConsumption : Apache_workflow
name = dailyConsumption
id = dailyConsumption
scriptDir = ~/apache/apps/PSG4C/scripts
artifactLocator = ~/apache/apps/PSG4C/jars
configDir = ~/apache/apps/PSG4C/config
dataDir = ~/apache/apps/PSG4C/data
logDir = ~/apache/apps/PSG4C/log
rootTask = powerTask
...
scheduler
ownedTasks
ownedTasks
rootTask
ownedTaskExecutors
ownedMeter
dsRackScheduler : NodeScheduler
powerTask : Task
profileTask : Task
dailyConsumptionExec : TaskExecutor
dailyConsumptionMeter : WorkflowLatencyMeter
metric
meter
dailyConsumptionLatency : WorkflowLatency
ownedStreamData
powerStored : WorkflowStreamData
ownedStreamData
profileStored : WorkflowStreamData

Figure 23: DailyConsumption workflow model

On the one hand, this case study allows us to validate that our reference architecture is flexible enough and support the modelling of data-intensive applications [58] meeting the design features pointed out by [6]: decentralisation, real-time, data management, service orientation, virtualisation and integration in business process. On the other hand, this case study shows that the number of elements to set up and deploy in a digital platform under a third generation architecture [59] is large and complex. For this example, the tool generates a total of 99 files (available in [57]): a registration script for each one of the 21 topics defined, an script for launching each `TaskExecutor` inside the workflows, and a script for instantiating each monitoring agent and each middleware service defined, along with their corresponding configuration files. This proves that the task of writing coherent configuration files and launch files is overwhelming, time-consuming, costly to reconfigure, and prone to errors if these are performed manually. Our proposal makes this task easier by collecting all the information in a model compliant to the RAI4.0 Metamodel and using this as input of the RAI4.0 Deployment tool.

7. Discussion section

This section discusses the advantages and current limitations of our proposal and compares our reference architecture with other ones found in the literature. Likewise, we comment the strategy designed to include relevant issues such as data quality and security aspects in our proposal. In addition, we mention our current advances and future works.

7.1. RAI4.0 architecture vs other alternatives

First of all, we point out that RAI4.0 is a reference architecture designed under the premises of sharing and taking advantage of the huge data volumes generated in industrial environments with the aim of getting the vertical and horizontal integration of the value chain, being aware of the heterogeneity of devices and protocols that cohabit in these complex systems and the restrictive functional requirements such as latency to which they are subject to.

One of the main advantages of our proposal is that it has been designed for being generic, technologically agnostic, distributed, scalable and flexible for integrating heterogeneous services with different kind of requirements. This provides developers with a common architecture to build data intensive applications whose specific requirements, in particular, those related to real time, must be solved by deploying the solution using an enough number of nodes for carrying out the computational processes. With the aim of avoiding communication and transfer delays, and thus, easing the fulfilment of the timing requirements, these nodes must be placed near those that store the data to be processed (edge computing). For instance, data from the monitoring and control of a robot usually need to be processed on the nearest node in order to meet latency requirements [60] and thus, these should never be stored or processed on the cloud.

The fact of having designed the reference architecture following a model-based strategy is another strength of our proposal. This makes its extension, reusability and customization simple and easy. For instance, we are currently extending our reference architecture to include the possibility of deploying the digital platform by means of containers using Docker or Kubernetes [57].

As previously mentioned, our architectural proposal is based on a shared data bus which facilitates the collaboration between machines, processes, systems and people through the exchange of information in real time. Other architectural proposals utilised in the industry are based on the three-layer pattern (edge, platform and business) or on the mediated edge connectivity and management pattern [5]. In the first one, the edge level collects data from cyber-physical devices to transfer it to the platform level, which, in turn, provides management functions for data assets and offers services related to its analysis, whereas the business level provides interfaces to end users and implements applications and decision support systems. That means, it divides the solution functionally and there is no a global sharing, integration and reuse of data in the environment. We remark two proposals under this approach: the conceptual model proposed by Ungurean et al. [61] exclusively addressed to IIoT applications and an extension of the three-layer pattern architecture with special focus on providing security based on blockchain technologies defined by Sittón-Candanedo et al. [62]. Both of them are explained at high abstraction level, not providing a detailed description of the elements that comprise an implementable digital platform. The second pattern mentioned pursues the local connectivity of devices at the edge and their isolation except for a gateway that connects to a wide area network. Its main benefit is to reduce the complexity of IIoT systems, so that they can scale both in number of managed devices and in networks. Thus its focus is mainly at physical level. This pattern has been used by Marino et al. [63] with the aim of designing a new networked architecture for interconnected logistics hubs and by developing a cloud-based Physical Internet (PI) framework and platform.

Other proposals found in the literature for addressing the industrial revolution 4.0 are based on the concept of digital twins. A digital twin is a digital representation of a real-world entity or system that continuously learns and updates itself from multiple sources to represent its near real-time status, working condition or position. Aheleroff et al. [64] adopted and extended RAMI [4] for the integration of digital twins in their reference architecture model. As RAMI, this is a conceptual reference, this means, it does not provide technical guidelines to implement digital twins as a notably difference with respect to our proposal. Even more, our reference architecture could be used to design and deploy digital twins, being this another possible use case. The importance of this concept has led ISO to develop a new ISO 23247 standard, still under development, which will include the specification of a reference architecture with functional views [65], thus this will not delve into implementation aspects.

Another relevant enabling technology in Industry 4.0 is the knowledge extraction from data. In this regard, we also found frameworks that guide developers to carry out analytical processes such as Carvalho et al. [66]. This kind of applications are considered as another possible use case of our proposal and therefore these can be configured and deployed with our tool. In this moment, we are working in a data stream mining prototype aimed at predictive maintenance.

Regarding the deployment tool built, we recognize that currently one possible limitation is that the deployment configuration is established based on the developers' expertise. Nevertheless, the information provided by the RAI4.0 model could be also used to apply early performance analysis in order to make decisions about the most appropriate deployment platform. To meet this aim, specific tools should be added to the RAI4 environment. Two approaches could be adopted for the development of these tools: i) defining them as extensions of the MAST environment [67, 68], developed in our research group or, ii) using tools provided by external frameworks, such the one described in [10, 38]. The former provides a set of tools for the analysis of real time distributed applications based on analytical and simulation techniques. It is a proved open-source software that is in continuous improvement and evolution and which relies on a platform metamodel very similar to our proposal. However, it is not addressed to cloud-based platforms, so its integration with RAI4 may require some extensions of its metamodel or tools. On the other hand, the latter facilitates the usage of frameworks already adapted to cloud computing however, it would require a greater effort of comprehension and integration due to the unfamiliarity with the underlying metamodels, which may use a different architectural approach. In any case, the integration of both alternatives with RAI4 is equally feasible. Both would require the definition of the corresponding model-to-model transformations and their integration in the current environment.

Another concern that IT staff in head of the production environment could have is how to detect the needs of reconfiguration of the digital platform deployed. For this end, our proposal is to add monitoring artifacts to the platform, which generate notifications and alarms about problems of performance or other issues. Next, IT staff could design a new deployment model with RAI4.0 Deployment Tool based on the previous one, being only needed to modify the elements or parameters required for solving the detected issues.

In our honest opinion, the use of this framework consistently in a company can facilitate the coherent development of data intensive applications, the reutilization and sharing of resources and the creation, use, enrichment and knowledge extraction of data needed for the decision-making.

7.2. Security, privacy and data quality issues

Security, privacy and data quality among other non-functional requirements are highly relevant and non-trivial issues that must be taken into account in any digital platform. Our reference architecture relies on both a metadata repository that stores all the features associated to each topic (or data source) registered in the environment and a security service following the strategy defined in [69], responsible for providing secure access to the global information shared by all the resources in the distributed system.

The security aspects that the RAI4.0 reference architecture will first incorporate are those related to the data available in the environment and that have been recorded by the publishers and are received by the subscribers. Four security issues will be initially addressed:

- Authenticity: The agents that access the environment (administrators, publishers or subscribers) are authenticated by the environment, and depending on their identity, they will have differentiated capabilities to access topics.

- Confidentiality: data registered in the environment can only be read by authorized subscribers in accordance with the confidentiality permissions established in the topics and the security credentials of the agents.

- Integrity: the instances of the topics that are registered in the environment are only introduced by authorized publishers, which guarantee the integrity characteristics established in the integrity policy defined in the topic. Likewise, the environment guarantees the integrity of the instances in the replication and transfer processes between the environment and the agents.

- Availability: Access to the data registered in the environment will be done in accordance with the availability policy established in the topic and the access credentials of the agent that enters in.

Other security issues [21, 5] such as protection against ciber-attacks will be studied once we have developed and deployed our security service. On the other hand, the use of artificial intelligence and analytical techniques involves the concern to safeguard the privacy of personal data [70]. For this purpose, specific metadata will be defined on topics in order to limit their use at a certain degree of granularity and to be anonymised.

Quality is another relevant aspect to bear in mind in an I4.0 digital platform [71]. It must be ensured that the data sources (topics) ingested in the environment present adequate levels of quality for their intended use. Our reference architecture relies on registering quality metadata of each data source, measured by means of quality services as the one proposed by Rivas et al. [72]. There are different data quality dimensions, among which are included (but are not limited to) the following ones: Uniqueness, Accuracy, Consistency, Completeness, Timeliness, Currency and Validity (ISO 25012). Each one requires different type of measurements to be assessed. For example, measuring completeness requires analyzing whether there is a value (any value) in a field, whereas measuring validity requires comparing existing format, patterns, data types, ranges, values, and more to the defined set of what is allowable [73].

Data stewards in collaboration with developers will be responsible for defining quality rules and the minimum thresholds that each pair data source - workload must fulfil. Publishers and workloads of our digital platform thus, will have to take into account and verify the fulfillment of these thresholds in real time and, in case of non-compliance, developers must establish the actions to be carried out (i.e. notify warnings topics, generate alarms, stop the processes affected, etc.).

8. Conclusions

The real implementation of Industry 4.0 supposes a revolutionary shift both in its conception and its design. Industrial processes must be radically transformed from traditional hierarchical models to a network model of interconnected services through the sharing and interchange of data. Data, indeed, are the central element of the architecture [7]. There are two main aspects to bear in mind, the volume, velocity, variety and criticality of data generated in the environment and the functional and non-functional constraints of the industrial processes which are generally subjected to real time restrictions, service availability, security and dynamic integration of heterogeneous devices [2]. The lack of a reference architecture that guided the implementation of the level of information defined in RAMI 4.0 [4] or its equivalent, the IIRA Functional information domain [5] led us to design and propose the RAI4.0 reference architecture as well as a metamodel that would help developers in the design of a flexible, easily extendible, scalable and distributed computational environment for Industry 4.0. This is the goal of our RAI4.0 Metamodel, which describes the different elements of the architecture, establishes the relationships among them and guide in the building of industrial data intensive applications. Furthermore, the metamodel also provides the necessary information to configure, deploy and execute workloads (applications) on a digital platform. In this case, only for applications built with Apache framework tools. Finally, we also offered a model-based tool compliant to this metamodel which automates the generation of configuration, deployment and launch files, their transference and execution in the nodes of the platform. This provides two main advantages: time-saving as a consequence of the model reuse and error-reduction in the deployment process and subsequent reconfiguration tasks.

All these software artifacts have been validated with several case studies, two of them are described in the paper. The first one, a simple and academic data streaming application that reads pollution levels in different areas of a city provided by mobile sensors that are ingested through Kafka and processed by Apache Storm to finally be persisted in Cassandra. This is deployed on a premise server, on a premise cluster and on the cloud. And the second one, a more complex empirical case study addressed to the management of the electric power consumption of a smart medium-size city deployed on a premise cluster on the edge.

Ongoing work mainly focuses on the development of a data-centric security service and the proposal of a data governance framework for our RAI4.0 reference architecture [74]. Another aspects we are addressing are the extension of the metamodel to containerize applications and deploy them on Amazon Web Services and in providing developers with tools that support them in other software development stages. We are also interested in improving the deployment tool, by providing it as an Eclipse plug-in that offers templates for easing the definition of the deployment models or even a graphical edition tool.

9. 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.

10. Acknowledgements

This work has been funded in part by the Spanish Government and FEDER funds (AEI/FEDER, UE) under grant TIN2017-86520-C3-3-R (PRECON-I4).

References

[1] K.-D. Thoben, S. Wiesner, T. Wuest, "industrie 4.0" and smart manufacturing – a review of research issues and application examples, International Journal of Automation Technology 11 (1) (2017) 4–16. `doi:10.20965/ijat.2017.p0004`.

[2] N. Velásquez, E. Estevez, P. Pesado, Cloud computing, big data and the industry 4.0 reference architectures, Journal of Computer Science and Technology 18 (03) (2018) e29. `doi:10.24215/16666038.18.e29`.
URL `http://journal.info.unlp.edu.ar/JCST/article/view/1151`

[3] V. Alcácer, V. Cruz-Machado, Scanning the industry 4.0: A literature review on technologies for manufacturing systems, Engineering Science and Technology, an International Journal 22 (3) (2019) 899 – 919. doi:https://doi.org/10.1016/j.jestch.2019.01.006.
URL http://www.sciencedirect.com/science/article/pii/S2215098618317750

[4] P. I. 4.0, Reference architectural model industrie 4.0, https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/rami40-an-introduction.html, accessed 30 Dec 2019 (2018).

[5] I. I. Consortium, Industrial internet reference architecture v1.9, http://www.iiconsortium.org/IIRA.htm, accessed 30 April 2019 (2019).

[6] C. Salkin, M. Oner, A. Ustundag, E. Cevikcan, A Conceptual Framework for Industry 4.0, Springer International Publishing, Cham, 2018, pp. 3–23. doi:10.1007/978-3-319-57870-5_1.
URL https://doi.org/10.1007/978-3-319-57870-5_1

[7] T. P. Raptis, A. Passarella, M. Conti, Data management in industry 4.0: State of the art and open challenges, IEEE Access 7 (2019) 97052–97093. doi:10.1109/ACCESS.2019.2929296.

[8] 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, Journal of Manufacturing Systems 54 (2020) 138 – 151. doi:https://doi.org/10.1016/j.jmsy.2019.11.004.
URL http://www.sciencedirect.com/science/article/pii/S0278612519300937

[9] Data-centric architecture: A model for embracing the machine age, https://www.stratio.com/whitepaper-data-centric-architecture/, accessed 30 Dec 2019 (2019).

[10] D. Pérez-Palacín, J. Merseguer, J. I. Requeno, M. Guerriero, E. Di Nitto, D. A. Tamburri, A uml profile for the design, quality assessment and deployment of data-intensive applications, Software and Systems Modeling 18 (6) (2019) 3577–3614. doi:10.1007/s10270-019-00730-3. URL https://doi.org/10.1007/s10270-019-00730-3

[11] K. Al-Gumaei, K. Schuba, A. Friesen, S. Heymann, C. Pieper, F. Pethig, S. Schriegel, A survey of internet of things and big data integrated solutions for industrie 4.0, in: 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA), Vol. 1, 2018, pp. 1417–1424. doi:10.1109/ETFA.2018.8502484.

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

[13] J. Min, Y. Kim, S. Lee, T.-W. Jang, I. Kim, J. Song, The fourth industrial revolution and its impact on occupational health and safety, worker's compensation and labor conditions, Safety and Health at Work 10 (4) (2019) 400 – 408. doi:https://doi.org/10.1016/j.shaw.2019.09.005. URL http://www.sciencedirect.com/science/article/pii/S2093791119304056

[14] M. M. Mabkhot, A. M. Al-Ahmari, B. Salah, H. Alkhalefah, Requirements of the smart factory system: A survey and perspective, Machines 6 (2018) 2–23. doi:https://doi.org/10.3390/machines6020023.

[15] S. Ahmad, A. Badwelan, A. M. Ghaleb, A. Qamhan, M. Sharaf, Analyzing critical failures in a production process:is industrial iot the solution?, Wireless Communications and Mobile Computing (2018). doi:10.1155/2018/6951318.

[16] I. Sittón-Candanedo, R. Alonso, J. Corchado Rodríguez, S. Rodríguez, R. Casado-Vara, A review of edge computing reference architectures and a new global edge proposal, Future Generation Computer Systems 99 (05 2019). doi:10.1016/j.future.2019.04.016.

[17] V. Marinakis, H. Doukas, J. Tsapelas, S. Mouzakitis, A. Sicilia, L. Madrazo, S. Sgouridis, From big data to smart energy services: An application for intelligent energy management, Future Generation Computer Systems (05 2018). doi:10.1016/j.future.2018.04.062.

[18] M. E. Zorrilla, Á. Ibrain, Bernard, an energy intelligent system for raising residential users awareness, Computers & Industrial Engineering 135 (2019) 492–499. doi:10.1016/j.cie.2019.06.040.
URL https://doi.org/10.1016/j.cie.2019.06.040

[19] A. Grieco, P. Caricato, D. Gianfreda, M. Pesce, V. Rigon, L. Tregnaghi, A. Voglino, An industry 4.0 case study in fashion manufacturing, Procedia Manufacturing 11 (2017) 871 – 877, 27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM2017, 27-30 June 2017, Modena, Italy. doi:https://doi.org/10.1016/j.promfg.2017.07.190.
URL http://www.sciencedirect.com/science/article/pii/S2351978917303980

[20] S. Tiwari, H. Wee, Y. Daryanto, Big data analytics in supply chain management between 2010 and 2016: Insights to industries, Computers & Industrial Engineering 115 (2018) 319 – 330. doi:https://doi.org/10.1016/j.cie.2017.11.017.
URL http://www.sciencedirect.com/science/article/pii/S0360835217305508

[21] D. S. Terzi, R. Terzi, S. Sagiroglu, A survey on security and privacy issues in big data, in: 2015 10th International Conference for Internet Technology

and Secured Transactions (ICITST), 2015, pp. 202–207. doi:10.1109/ICITST.2015.7412089.

[22] N. Tariq, M. Asim, F. Al-Obeidat, M. Zubair Farooqi, T. Baker, M. Hammoudeh, I. Ghafir, The security of big data in fog-enabled iot applications including blockchain: A survey, Sensors 19 (8) (2019) 1788. doi:10.3390/s19081788.
URL http://dx.doi.org/10.3390/s19081788

[23] 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.

[24] M. Ghobakhloo, The future of manufacturing industry: a strategic roadmap toward industry 4.0, Journal of Manufacturing Technology Management 29 (2018) 910–936.

[25] C. B. Lopez, J. J. García, S. H. González, Análisis exhaustivo de los principios de diseño en el contexto de industria 4.0, Revista Iberoamericana de Automática e Informática industrial 0 (0) (2020). doi:10.4995/riai.2020.12579.
URL https://polipapers.upv.es/index.php/RIAI/article/view/12579

[26] P. Angulo, C. C. Guzmán, G. Jiménez, D. Romero, A service-oriented architecture and its ict-infrastructure to support eco-efficiency performance monitoring in manufacturing enterprises, International Journal of Computer Integrated Manufacturing 30 (1) (2017) 202–214. arXiv:https://www.tandfonline.com/doi/pdf/10.1080/0951192X.2016.1145810, doi:10.1080/0951192X.2016.1145810.
URL https://www.tandfonline.com/doi/abs/10.1080/0951192X.2016.1145810

[27] S. Wiesner, K.-D. Thoben, Requirements for models, methods and tools supporting servitisation of products in manufacturing service ecosystems,

International Journal of Computer Integrated Manufacturing (2016) 1–11doi:10.1080/0951192X.2015.1130243.

[28] W. Wingerath, F. Gessert, S. Friedrich, N. Ritter, Real-time stream processing for big data, Information Technology 4 (58) (2016) 186–194.

[29] M. Arantes, R. Bonnard, A. P. Mattei, P. de Saqui-Sannes, General architecture for data analysis in industry 4.0 using sysml and model based system engineering, in: 2018 Annual IEEE International Systems Conference, SysCon 2018, Vancouver, BC, Canada, April 23-26, 2018, 2018, pp. 1–6. doi:10.1109/SYSCON.2018.8369574.
URL https://doi.org/10.1109/SYSCON.2018.8369574

[30] A. Wortmann, B. Combemale, O. Barais, A systematic mapping study on modeling for industry 4.0, in: 2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS), 2017, pp. 281–291. doi:10.1109/MODELS.2017.14.

[31] R. Petrasch, R. Hentschke, Process modeling for industry 4.0 applications: Towards an industry 4.0 process modeling language and method, in: 2016 13th International Joint Conference on Computer Science and Software Engineering (JCSSE), 2016, pp. 1–5. doi:10.1109/JCSSE.2016.7748885.

[32] K. Suri, J. Cadavid, M. Alferez, S. Dhouib, S. Tucci-Piergiovanni, Modeling business motivation and underlying processes for rami 4.0-aligned cyber-physical production systems, in: 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2017, pp. 1–6. doi:10.1109/ETFA.2017.8247702.

[33] S. M. Kannan, K. Suri, J. Cadavid, I. Barosan, M. v. d. Brand, M. Alferez, S. Gerard, Towards industry 4.0: Gap analysis between current automotive mes and industry standards using model-based requirement engineering, in: 2017 IEEE International Conference on Software Architecture Workshops (ICSAW), 2017, pp. 29–35. doi:10.1109/ICSAW.2017.53.

[34] L. F. C. S. Durão, H. Eichhorn, R. Anderl, K. Schützer, E. de Senzi Zancul, Integrated component data model based on uml for smart components lifecycle management: A conceptual approach, in: A. Bouras, B. Eynard, S. Foufou, K.-D. Thoben (Eds.), Product Lifecycle Management in the Era of Internet of Things, Springer International Publishing, Cham, 2016, pp. 13–22.

[35] Y. Lassoued, S. Nurcan, Modeling contextualized flexible cloud workflow services: An mde based approach, in: 2017 11th International Conference on Research Challenges in Information Science (RCIS), 2017, pp. 44–55. doi:10.1109/RCIS.2017.7956516.

[36] A. Rajbhoj, V. Kulkarni, N. Bellarykar, Early experience with model-driven development of mapreduce based big data application, in: 2014 21st Asia-Pacific Software Engineering Conference, Vol. 1, 2014, pp. 94–97. doi:10.1109/APSEC.2014.23.

[37] S. Santurkar, A. Arora, K. Chandrasekaran, Stormgen - a domain specific language to create ad-hoc storm topologies, in: 2014 Federated Conference on Computer Science and Information Systems, 2014, pp. 1621–1628. doi:10.15439/2014F278.

[38] M. Guerriero, S. Tajfar, D. A. Tamburri, E. Di Nitto, Towards a model-driven design tool for big data architectures, in: Proceedings of the 2Nd International Workshop on BIG Data Software Engineering, BIGDSE '16, ACM, New York, NY, USA, 2016, pp. 37–43. doi:10.1145/2896825.2896835.
URL http://doi.acm.org/10.1145/2896825.2896835

[39] M. Arantes, R. Bonnard, A. P. Mattei, P. de Saqui-Sannes, General architecture for data analysis in industry 4.0 using sysml and model based system engineering, in: 2018 Annual IEEE International Systems Conference (SysCon), 2018, pp. 1–6. doi:10.1109/SYSCON.2018.8369574.

[40] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, A view of cloud computing, Commun. ACM 53 (4) (2010) 50–58. doi:10.1145/1721654.1721672.
URL http://doi.acm.org/10.1145/1721654.1721672

[41] The apache software foundation, http://www.apache.org/ (accessed 30 April 2019).

[42] D. Wang, Building value in a world of technological change: Data analytics and industry 4.0, IEEE Engineering Management Review 46 (2018) 32–33.

[43] The apache avro project: a data serialization system, http://avro.apache.org (accessed 30 April 2019).

[44] J. F., R. B., ZooKeeper: Distributed process Coordination, O,Reilly, 2014.

[45] Apache Kafka project: A distributed streaming platform, http://kafka.apache.org/ (accessed 30 April 2019).

[46] Apache Spark: A fast and general engine for large-scale data processing, http://spark.apache.org/,) (accessed 30 Dec 2019).

[47] Apache Storm: A fast and general engine for large-scale data processing, https://storm.apache.org/,) (accessed 30 Dec 2019).

[48] K. Grolinger, W. A. Higashino1, A. Tiwari, M. A. Capretz, Data management in cloud environments: Nosql and newsql data stores.journal of cloud computing: Advances, systems and applications 2013, 2:22, Journal of Cloud Computing: Advances, Systems and Applications (2013) 2–22doi:10.1186/2192-113X-2-22.

[49] Memsql:tutorials overview., https://docs.memsql.com/tutorials/v5.8/tutorials-overview/ (accessed 30 April 2019)).

[50] Apache Cassandra, http://cassandra.apache.org/ (accessed 30 April 2019).

[51] I. Robinson, J. Webber, E. Eifrem, Graph Databases, O'Reilly, 2013.

[52] A. Furfaro, A. Garro, A. Tundis, Towards security as a service (secaas): On the modeling of security services for cloud computing, in: 2014 International Carnahan Conference on Security Technology (ICCST), 2014, pp. 1–6. `doi:10.1109/CCST.2014.6986995`.

[53] Prometheus overview, `https://prometheus.io/docs/introduction/overview/` (accessed 30 April 2019).

[54] M. Brambilla, J. Cabot, M. Wimmer, Model-Driven Software Engineering in Practice, Morgan & Claypool publishers, 2012.

[55] Prometheus exporters, `https://github.com/prometheus/node_exporter` (accessed 30 April 2019).

[56] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks, EMF: Eclipse Modeling Framework 2.0, 2nd Edition, Addison-Wesley Professional, 2009.

[57] RAI4 deployment tool and metamodel, `https://github.com/istr-uc/RAI4DeploymentTool` (accessed 20 July 2020).

[58] M. Kleppmann, Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems, O'Reilly, 2017.

[59] K. Kwang, Third platform shift triggers enterprise software evolution, `http://www.zdnet.com/article/third-platform-shift-triggers-enterprise-software-evolution/`, accessed 30 April 2019 (5 2013).

[60] Y. Chen, Q. Feng, W. Shi, An industrial robot system based on edge computing: An early experience, in: USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18), USENIX Association, Boston, MA, 2018. URL `https://www.usenix.org/conference/hotedge18/presentation/chen`

[61] I. Ungurean, N. C. Gaitan, A software architecture for the industrial internet of things—a conceptual model, Sensors 20 (19) (2020). doi:10.3390/s20195603.
URL https://www.mdpi.com/1424-8220/20/19/5603

[62] I. Sittón-Candanedo, R. S. Alonso, J. M. Corchado, S. Rodríguez-González, R. Casado-Vara, A review of edge computing reference architectures and a new global edge proposal, Future Generation Computer Systems 99 (2019) 278 – 294. doi:https://doi.org/10.1016/j.future.2019.04.016.
URL http://www.sciencedirect.com/science/article/pii/S0167739X1930264X

[63] C. Salvadori, f. marino, p. castoldi, S. Bocchino, V. Dao, I. Seitanidis, Iot enabling pi: towards hyperconnected and interoperable smart containers, 2019.

[64] S. Aheleroff, X. Xu, R. Y. Zhong, Y. Lu, Digital twin as a service (dtaas) in industry 4.0: An architecture reference model, Advanced Engineering Informatics 47 (2021) 101225. doi:https://doi.org/10.1016/j.aei.2020.101225.
URL http://www.sciencedirect.com/science/article/pii/S1474034620301944

[65] G. Shao, M. Helu, Framework for a digital twin in manufacturing: Scope and requirements, Manufacturing Letters 24 (2020) 105 – 107. doi:https://doi.org/10.1016/j.mfglet.2020.04.004.
URL http://www.sciencedirect.com/science/article/pii/S2213846319301312

[66] G. G. de Carvalho Chrysostomo, M. V. B. de Aguiar Vallim, L. S. da Silva, L. A. Silva, A. R. de Aguiar Vallim Filho, A framework for big data analytical process and mapping—baprom: Description of an application in an industrial environment, Energies 13 (22) (2020). doi:10.3390/en13226014.
URL https://www.mdpi.com/1996-1073/13/22/6014

[67] M. González Harbour, J. J. Gutiérrez, J. M. Drake, P. López Martínez, J. C. Palencia, Modeling distributed real-time systems with mast 2, Journal of Systems Architecture 59 (6) (2013) 331 – 340. doi:https://doi.org/10.1016/j.sysarc.2012.02.001. URL http://www.sciencedirect.com/science/article/pii/S1383762112000033

[68] MAST: Modelling and analysis suite for real-time applications, https://mast.unican.es (accessed 03 February 2021).

[69] A. Furfaro, A. Garro, A. Tundis, Towards security as a service (secaas): On the modeling of security services for cloud computing, in: 2014 International Carnahan Conference on Security Technology (ICCST), 2014, pp. 1–6. doi:10.1109/CCST.2014.6986995.

[70] J. Koo, G. Kang, Y.-G. Kim, Security and Privacy in Big Data Life Cycle: A Survey and Open Challenges, Sustainability 12 (24) (2020) 1–1. URL https://ideas.repec.org/a/gam/jsusta/v12y2020i24p10571-d463896.html

[71] D. Williams, H. Tang, Data quality management for Industry 4.0: A survey, Data Quality 22 (2) (2020) 26–35.

[72] B. Rivas, J. Merino, I. Caballero, M. Serrano, M. Piattini, Towards a service architecture for master data exchange based on iso 8000 with support to process large datasets, Computer Standards & Interfaces 54 (2017) 94 – 104, sI: New modeling in Big Data. doi:https://doi.org/10.1016/j.csi.2016.10.004. URL http://www.sciencedirect.com/science/article/pii/S0920548916301192

[73] D. Plotkin, Data Stewardship: An Actionable Guide to Effective Data Management and Data Governance, Morgan Kauffmann, 2013.

[74] J. Yebenes, M. Zorrilla, Towards a data governance framework for third generation platforms, Procedia Computer ScienceThe 2nd International Conference on Emerging Data and Industry 4.0 (EDI40) (2019).

A big data centric architecture metamodel for Industry 4.0

- A reference architecture for Industry 4.0 applications.
- A metamodel, called RAI4 Metamodel, for the design and deployment of applications according to the proposed reference architecture.
- A model-based tool that enables configuring, deploying and launching applications designed according to the RAI4 metamodel.
- Two case studies, addressed to the management of electric power consumption of a smart medium-size city and to the analysis of pollution data obtained from sensors deployed in a smart city.

Patricia López Martínez received her PhD degree from the University of Cantabria (Spain) in 2010 with a thesis focused on the development of component-based real-time applications. She is an Associate Professor at the Software Engineering and Real-Time group (ISTR) of the University of Cantabria and she has been involved in several national and European research projects. Her main research interests are focused on applying software engineering approaches (model-based development, component-based development, etc.) to the development of real-time systems and recently to Industry 4.0.

Ricardo Dintén received the B.S. degree in Computer Science from the University of Cantabria, Spain in 2019. He is currently studying for a M.S. degree in Computer Science at the University of Cantabria and working as a researcher at the Software Engineering and Real-Time group (ISTR). His research interests are centered in Data-Intensive Applications, cloud computing and Industry 4.0.

José M. Drake received a PhD in Science from Seville University in 1976. He is currently retired but he has been Full Professor of the University of Cantabria and team head of the Software Engineering and Real-Time research group. Primary interests are in software engineering for distributed real-time systems. He has dealt with conceptual aspects, as specification and modelling of real-time systems, new paradigms of design, such as component-based systems and resource allocation, and specification and implementation of middleware to support these paradigms on distributed systems. In collaboration with industrial companies, he has developed a wide range of projects using real-time engineering for nuclear industry and electrical power control and monitoring.

Marta Zorrilla is an Associate Professor in Computer Science within Software Engineering and Real Time Group at the University of Cantabria (Spain). She has been involved in several national and European research projects together with other international research institutions. Her research interests are the database technologies, data mining and big data, currently applied to Industry 4.0. She is author of a database book and more than 60 works published in international journals, chapters and conferences. She is an active reviewer of several international journals and conferences such as Expert Systems with Applications, Decision Support Systems, International Journal of Information Technology & Decision Making, IEEE Transactions on Human-Machine Systems, among others.

Patricia López Martínez: Software, Conceptualization, Investigation, Writing - Review & Editing **Ricardo Dintén:** Software, Writing - Review & Editing, Investigation, **José María Drake:** Conceptualization, Methodology, **Marta E. Zorrilla:** Conceptualization, Resources, Writing – Original Draft, Project Administration

P Lopez

Ricardo Dinten

JM Drake

Marta Zorrilla

Declaration of interests

☒ 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.

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: