Knowledge-Based Systems 232 (2021) 107489

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

TITAN: A knowledge-based platform for Big Data workflow management☆

Antonio Benítez-Hidalgo[1], Cristóbal Barba-González, José García-Nieto, Pedro Gutiérrez-Moncayo, Manuel Paneque, Antonio J. Nebro, María del Mar Roldán-García, José F. Aldana-Montes, Ismael Navas-Delgado *

Dept. de Lenguajes y Ciencias de la Computación, ITIS Software, Universidad de Málaga, Málaga 29071, Spain

ARTICLE INFO

Article history:
Received 30 November 2020
Received in revised form 2 September 2021
Accepted 6 September 2021
Available online 10 September 2021

Keywords:
Big Data analytics
Semantics
Knowledge extraction

ABSTRACT

Modern applications of Big Data are transcending from being scalable solutions of data processing and analysis, to now provide advanced functionalities with the ability to exploit and understand the underpinning knowledge. This change is promoting the development of tools in the intersection of data processing, data analysis, knowledge extraction and management. In this paper, we propose TITAN, a software platform for managing all the life cycle of science workflows from deployment to execution in the context of Big Data applications. This platform is characterised by a design and operation mode driven by semantics at different levels: data sources, problem domain and workflow components. The proposed platform is developed upon an ontological framework of meta-data consistently managing processes and models and taking advantage of domain knowledge. TITAN comprises a well-grounded stack of Big Data technologies including Apache Kafka for inter-component communication, Apache Avro for data serialisation and Apache Spark for data analytics. A series of use cases are conducted for validation, which comprises workflow composition and semantic meta-data management in academic and real-world fields of human activity recognition and land use monitoring from satellite images.

1. Introduction

Big Data is usually described by the well-known V's (Volume, Velocity, and Variety, Variability, Veracity, and Value, among others) and different approaches generally focus on some of these features. A plethora of technologies (most of them Open Source projects, such as those from Apache Foundation) have emerged in the last years around Big Data to propose generic solutions that could underline any of these V's [1,2]. For example, for dealing with Volume, NoSQL databases (Apache Cassandra, MongoDB, Apache HBase, Neo4J among many more[2]) propose new approaches to the efficient management of large datasets. In terms of handling extensive scale data, there exist several proposals to leverage distributed computing in an efficient manner (such as Apache Spark), that ultimately speed up the data processing time. Velocity is not only taken into consideration in NoSQL databases (to ensure the ingestion of fast data streams), but also in the analysis of data streams (e.g., Apache Kafka, Flink, and Spark).

However, one of the main focuses nowadays is on providing Value out of all the available data. Getting this Value is not an easy task that could be solved with a single horizontal technology. Thus, there are also many approaches related to Big Data analytics (i.e., the use of advanced analysis techniques applied to large and diverse datasets) coming from heterogeneous vertical domains, such as transportation [3], healthcare [4], e-Science [5], and agriculture [6]. The goal is to provide solutions that can deal with several Big Data issues in an application using the technologies that best suit for different aspects of the problem.

The development of a solution that provides Value to a given Big Data analysis process is usually hard-coded in the application. With the explosion of data and the expectations to make use of it, organisations start automating data extraction processes, which would end up in data repositories from which extracting Value is a challenge. The use of semantics, including contextual information, is a promising approach to deal with data to improve the data analysis processes and to ensure the efficient reuse

☆ This work has been partially funded by the Spanish Ministry of Science and Innovation via Grant PID2020-112540RB-C41 (AEI/FEDER, UE) and Andalusian PAIDI program with grant P18-RT-2799. Funding for open access charge: Universidad de Málaga / CBUA.

* Corresponding author.
E-mail address: ismael@uma.es (I. Navas-Delgado).

[1] Supported by Grant PRE2018-084280 (Spanish Ministry of Science, Innovation and Universities).

[2] See https://hostingdata.co.uk/nosql-database/ for a long list of classified NoSQL databases.

https://doi.org/10.1016/j.knosys.2021.107489

of software components [7,8]. Thus, a current trend is the development of data-driven applications capturing, managing and injecting domain knowledge into their software components [7,9,10] to cope with the complexity of this kind of applications. Regardless of how semantics is built up and captured (elicitation of domain semantics), the use of domain semantics in Big Data applications can exploit the use of ontologies. That is to say, it uses a formal description of the concepts in the domain of interest and the relationships among them [11], to discover new knowledge (Value) and guide the analysis processes.

The use of semantics opens some exciting research challenges: enhancing data and analysis results quality; validation of the analysis processes (both during design and execution phases); traceability of the analysis results and capability of generating not only an explanation of results but also a "confidence level" which depends on, among others, data provenance and domain semantics [12]. In particular, Semantic Web technologies can be used to annotate data with domain knowledge and also to annotate features and transformation processes done by data analysis algorithms (e.g., algorithm parameters, input and output variables, and behaviours, among others).

In this context, we developed the BIGOWL ontology [7], which was the result of an ontology-driven approach to support knowledge management in Big Data analytic workflows. With the semantics, our ontology follows the tendency of the Big Data analysis for automating the selection of concrete modelling techniques such as deep learning or multi-objective, using service composition [13,14]. However, taking advantage of BIGOWL in real scenarios required much effort from the developer's side, requiring the manual annotation of the information and deploying the workflows in the execution environment. In this paper, we present TITAN (*seman***T***ics in b***I***g da***T***a* **AN***alytics*), a semantic-enhanced software platform for the accurate design, annotation, development, testing, deployment and execution of Big Data analytic workflows. In our proposal, BIGOWL is acting at the *core* of TITAN by providing the ontology scheme according to which semantic annotation of algorithmic components, data sources, operation constraints, and execution planning is performed. Thus, TITAN can take advantage of semantics not only in the workflow construction, recommendation or validation, but also by giving domain context to the analysis. Our framework distinguishes between programmer users, who develop new components and annotate them using the semantics, and business users, that use such components to perform the analysis. TITAN enhances their user experiences, so that the latter can benefit of all the framework semantic features without knowing anything about how to declare the semantics of the data or components developed by the prior. The main contributions of this platform are described next:

- Improvement and extension of BIGOWL [7] to enable the automatic deployment of workflows in Big Data environments. This includes providing validation mechanisms to ensure the quality of workflows from the design stages, before their execution, by taking advantage of the operational semantics of BIGOWL.
- Development of tools to support the domain contextualisation of data processed along with Big Data workflows, keeping a chain of data provenance not only for the data themselves, but also for the workflow's parameters and hyper-parameters and their components.
- Development of an Open Source Graphical User Interface[3] (GUI) for the design of Big Data analytic workflows and their execution in distributed environments by providing native support to data stream analysis through Apache Kafka [15] and Apache Spark [16]. The definition and documentation of how developers can include their algorithms in TITAN will boost the creation of a developers' community.
- Development of an API[4] to enable the development of new software tools, that can be independent of the aforementioned GUI.

The rest of the paper is structured as follows. In Section 2, background notes are provided for contextualisation and comparison of related work with TITAN. Section 3 describes the architecture of our semantic-enhanced software platform. Section 4 presents the methodology to develop new components. Section 5 depicts how to design a workflow in TITAN. Hereafter, in Section 6 three use cases are described to illustrate different workflow compositions with TITAN. In Section 7, a series of discussions are included. Finally, Section 8 includes concluding remarks and future work.

2. Background

This section includes background concepts in the Semantic Web field. Furthermore, a review of state of the art is provided to point out the main differences of the related works concerning the proposed approach.

2.1. Background concepts

- Ontology. Inline with [17], an ontology provides a formal representation of the real world. It defines a description of concepts in a concrete domain (classes or concepts), properties of each concept describing various features and attributes of the concept (properties) and restrictions on properties. Ontologies are defined in the W3C standard stack of the Semantic Web.[5] An ontology accompanied by a set of individual instances of classes composes a knowledge base and offer services to facilitate interoperability across multiple heterogeneous systems and databases.
- RDF. Resource Description Framework [18] is included in the W3C recommendation that defines a language for describing resources on the web. RDF defines resources in terms of triples, consisting of a subject, predicate and object. RDF Schema (RDFS) [19] defines vocabularies used in RDF descriptions.
- SPARQL is a query language for accessing RDF stores. It is the query language advocated by W3C [20] to work with RDF graphs [21], then allowing queries and web data sources identified by URIs.

2.2. Workflow analytic tools

In the last decades, Workflow Management Systems (WMSs) have grown in popularity within the scientific community. They can be classified into two distinct categories: task-driven WMSs and data-driven WMSs [22]. In the task-driven approach (Section 2.2.1), workflow task completion triggers the execution of the dependent tasks in the workflow. Nevertheless, in a data-driven approach (Section 2.2.2), the functions in the workflow are started when the input data are available, rather than task completion dependencies.

There exists a series of studies in which (Big) Data analytic tools (both task and data driven) are defined to design and execute data analytic workflows. These studies do not only support

[3] https://github.com/KhaosResearch/TITAN-GUI

[4] https://github.com/KhaosResearch/TITAN-API

[5] https://www.w3.org/standards/semanticweb/

the automation of repetitive tasks, but they can also manage complex analysis processes at various levels of detail and systematically capture provenance information for the derived data products. Some approaches are able to exploit meta-data to some degree to support users in the construction of workflows and their deployments. Below we compare some of these proposals, depending on the two described categories.

2.2.1. Task-driven workflow management systems

VisTrails [23], which has been developed at the University of Utah, is a system whose goal is to simplify and streamline the process of scientific data exploration. VisTrails can be combined with a wide range of tools, libraries, and visualisation systems and captures provenance for data products and workflow designs. Furthermore, VisTrails uses its own engine to manage the execution, allowing workflows to be run online or locally [24].

Following the idea of improving workflow provenance, in 2015 appeared YesWorkflow [25] that is a set of software tools that allow scientists annotating scripts (in Python, R, or Perl) with special comments to enhance the provenance of scientific results. This provenance can be queried to improve workflow trustability and re-usability.

In the field of open-source analytic tools, an interesting approach is KNIME (Konstanz Information Miner) [26], which provides a robust, modular, and highly scalable platform for workflow design and execution. It includes components for data loading, transformation, analysis, and visual exploration models. Furthermore, KNIME allows users to design data flows and data channels in a visual way, to run some selectively or all analytical components. KNIME is able to process and integrate diverse datasets. However, the workflow-oriented approach does not fit appropriately for processing streaming data. To address this issue, KNIME provides a streaming executor, which executes nodes concurrently and so the results of one node are instantly provided as input to the other connected ones.

Another open-source tool for managing workflows is Airflow [27], a lightweight workflow manager that allows designing and executing workflow as a Directed Acyclic Graph (DAG) of tasks. Both order and relationships between tasks are described in the DAG. Airflow has an architecture that allows the distribution of tasks to a high-handed number of workers and across multiple servers. Its user interface allows visualising pipelines, monitor progress, and troubleshoot issues.

Taverna is an open-source workflow tool suite [28], which combines distributed Web Services and local tools into complex analysis pipelines with a large number of users in the e-Science community. Taverna focus on connecting data sources to extract relevant information and produce a simple analysis of these data. Nevertheless, Taverna is not focused on the analysis of Big Data. Its workflow steps or *Processors* are implemented either as local Java classes or as Web Services that implement the WSDL interface. Between each processor, there are data and control links among them. The data links aim to establish dependencies between the output of a processor and the input of another one. A control link indicates that a processor can only begin its execution when another processor has successfully completed its execution [29]. Taverna is able to operate in different execution environments providing several possibilities of deployment. Taverna also has a desktop design GUI in order to facilitate the workflow design process.

In the context of tools that take into account semantic representations to enhance the user's experience for designing workflows, we can find the Workflow Instance Generation and Specialisation (WINGS) [8]. WINGS allows scientists to design computational experiments through semantic representations that describe constraints of the data and computational steps in the workflow. The core ontologies of WINGS are defined in the Web Ontology Language (OWL), while the rest (e.g., workflow templates, constrains) are defined in the Resource Description Framework (RDF). WINGS ensures that only the right components are used in workflows by checking the semantic constraints of the input and output types. Furthermore, WINGS provides web-based access and can run workflows by itself, or deploy them to external workflow management system such as Pegasus [30].

Following the idea of using semantics to assist data analysts in selecting and developing models in predictive big data analysis, Kumar et al. [31] proposed to employ Automatic Service Composition (ASC), where users can use service composition techniques to combine existing services as components. This framework generates an abstract workflow for predictive big data analysis and it is supported for semi-automated model selection using analytics ontology.

MINT [32] is a framework for model integration that uses semantic representations (based on the Geoscience Standard Names ontology [33]) to describe datasets and models. MINT provides functionality for guiding users in the discovery of models and data to suggest appropriate combinations of models as well as in the execution and visualisation of results.

Similar to Pegasus, Makeflow [34] is a command-line workflow engine that represents data-intensive workflows using a file format very similar to the one used by the Make tool [35]. In Makeflow, the input data of each activity must be explicitly specified for the workflow representation or the workflow description will be regarded as incorrect. Makeflow supports many different execution environments, including local execution or HTCondor [36].

A last proposal in task-driven WMS is BioMOBY [37], which uses semantic approaches to enable the validation of scientific workflows, defining data types for inputs and outputs of the components.

2.2.2. Data-driven workflow management systems

Besides MapReduce-based solutions like Apache Hadoop [38], Apache Yarn [39] or Apache Spark [40], several generalist data-oriented workflow management solutions have been developed in the recent years [41].

Pachyderm [42] is an open-source workflow system and data management framework. A Pachyderm workflow is denoted a *pipeline* and is organised around data repositories (nodes in a DAG) containing data. Pachyderm runs a pipeline on its data and waits asynchronously for new data to be processed. This tool is not limited to a given data format or programming language to process the data. Besides, Pachyderm is based on the idea that there are no folders (in the form of separate storage buckets) and all messages should reside in a single conceptual store. Pachyderm is built on top of a large number of software layers and runs on top of widely-used commercial cloud providers such as Amazon S3 [43], Microsoft Azure [44] or Google Cloud [45].

Likewise, Nextflow [46] enables workflows to be run either locally or in most cluster environments, and supports running software via either Docker [47] or Singularity [48] clients, which depends on the operating system. Nextflow approach, like Pachyderm, runs pipelines on its data. Nextflow provides complete integration with several versioning platforms such as GitHub, which enables the workflow to check for updates and pull data from a given repository.

Another tool is the interactive data-driven workflow engine for Grid-enabled resources, VLAM-G, [49]. It is a decentralised data flow driven workflow engine focused on e-science community. The design of VLAM-G is based on several core Grid services (resource management, data access, and resource information services). The engine consists of a Run-Time Environment for

workflow components and a Run-Time System Manager that controls and orchestrates the execution. Workflows are created by connecting components through data dependencies. The components can be special software developed for VLAM-G, web services, or interfaces to legacy software. VLAM-G allows monitoring run-time executions, as well as interactivity through parameters of the connected modules.

Provenance Management for Data-Driven Workflows, Karma2 [50], is another tool where the individual services that compose a workflow publish their own provenance to minimise performance overheads at the workflow-engine level. Karma collects the metadata provenance in the form of a workflow trace and records them in a central database server. That metadata comprises information about task nodes in workflows, operating independently of the workflow management system used. The information collected can be used to produce workflow execution traces that identify producer/consumer relationships for data exchange by these services.

2.3. Workflow management systems comparative analysis

In this section, we summarise the main features of the analysed proposals (see Table 1). The main features taken into consideration are:

- ***Approach***. This column indicates wherever the approach is data driven (D) or task (T) driven.
- ***Big Data support***. The support for Big Data technologies has been taken into account by discriminating three possible cases: (N) Big Data technologies are not included, (Y*) Big Data technologies are included and (Y) focused only in Big Data technologies.
- ***Data Stream support***. The analysis of streaming data is a key topic in many Big Data workflows, which can be either supported (Y) or not supported (N).
- ***Semantic-Based approach***. The use of semantics can be present in several aspects: (D) design support, (V) workflow validation, (P) deployment support, (L) Linked Data provision. In other cases, (N) when there is no support.
- ***Open Source***. Open Source approaches enable long term maintenance of the systems, avoiding the limitation of a specific funding frame. There are three options: closed source (N) , open source (Y) or unknown (U).
- ***Cost***. The cost of use can be: (F) free, (C) free community version and enterprise version underpayment, (E) only available after any payment or subscription method, or unknown (U).
- ***Languages Supported***. The programming languages supported for the development of new components is relevant to be able to engage as many developers as possible in the case of Open Source solutions.
- ***Platform***. This feature makes reference to the way the proposal can be used: (O) online version available, (S) standalone version available, (C) cloud support.

As a summary, Table 1 outlines the main features of the related work with regards to the knowledge-based approach proposed here. These features consist of specifying whether the existing systems focus on their orientation to Big Data analysis streaming processing support, provide online version, programming language, use of semantics in different steps of the study, or the licence of the framework. Then, it is possible to identify the actual contributions of the proposed semantic model beyond state of the art, as follows:

- TITAN is conceived to conduct analysis in Big Data and/or streaming environments. Similar to other frameworks in the literature, it is oriented to general analysis procedures (i.e., not limited to Big Data).
- TITAN is a semantic-enhanced platform with features resembling task-driven approaches (e.g., workflow design) and data-driven approaches (such as the component connection through data dependencies).
- Although there exist frameworks that use semantics during the analysis process, no one of them is driven by an ontology as operational core. This fact allows guiding and annotating all the analysis processes from design to execution and improves all the steps thanks to the use of the domain semantics. Furthermore, ontologies provide a formal way of describing the knowledge domain, based on first-order logic, which can be exploited to infer new knowledge and ensure the annotations' consistency.
- The proposed approach is validated on three real-world use-cases consisting of classical data mining, streaming Big Data processing with a deep learning algorithm, and remote sensing image analysis using a machine learning algorithm.

3. TITAN architecture

TITAN is a semantic-enhanced workflow management system (WMS) with several architectural levels to deal with the design of new components until workflows' execution:

- ***Metadata framework***. Use of ontologies to enable a set of advanced features.
- ***Software architecture***. TITAN provides features from task-driven WMS and data-driven WMS.

3.1. Metadata framework

Our main target is the design of a component-based platform in which semantics can be used to represent different aspects of Big Data analytic workflows, such as data, tasks, components, algorithms, and problem domain knowledge. In essence, our platform allows composing complex workflows by the specification of a series of operations that need to be applied to the input data. However, TITAN includes not only Big Data components but also traditional elements for building data analysis workflows.

Thus, explicit knowledge in the domain and how the different components transform the data can guide a smart design of workflows. Each task is formally described using semantics to ensure compatibility among components through semantic reasoning. Typically, domain knowledge is captured in the analysis of the problem and manually integrated during the algorithm design phase. In the best case, it is included as a set of rules that modify the general behaviour of the analytic algorithms. BIGOWL is aligned to domain ontologies to integrate domain knowledge in the workflow design process. Furthermore, this ontology provides TITAN with the formal representation of the available components, classified in four main groups: data sources, data processing, data analysis and data sinks. Semantics is formally represented in our platform employing three levels of abstraction: BIGOWL ontology, RDF graphs (i.e., individuals describing the workflows), and domain ontologies.

BIGOWL describes workflows and entities containing software components. Tasks represent how these components are instantiated (parameter configuration) to be executed. Some components are processing (transformation) components, including algorithms. With the aim of describing components, this ontology contains four main classes: First, *Data Collection* for annotating the initial data to serve as the input of the following components

Table 1
Comparison of the main features of the platform with regards to other solutions, based on their approach, Big Data technologies, streaming data, use of semantics, Open Source, use cost, language support and where are available the platforms.

Framework	Approach	Big data support	Data stream support	Semantic-based approach	Open source	Cost	Languages supported	Platform
VisTrails	T	N	N	N	Y	F	Python	S
YesWorkflow	T	N	N	D	Y	F	Java & Python	S
KNIME	T	Y*	Y	L	N	E	Java	SC
Airflow	T	Y*	N	N	Y	F	Python	SC
Taverna	T	N	Y	N	Y	F	Java	S
WINGS	T	N	N	DV	Y	F	Java	S
Pegasus	T	Y*	N	DV	Y	F	Java	S
MINT	T	N	N	DV	Y	F	Python	S
Makeflow	T	Y	N	D	Y	F	None	S
BioMOBY	T	N	N	V	Y	F	Java	S
ASC	T	Y	N	D	U	U	U	U
Pachyderm	D	Y*	Y	N	N	E	Go	SC
Nextflow	D	Y*	Y	N	Y	F	Python	S
VLAM-G	D	Y*	N	N	U	U	U	U
Karma2	D	N	N	D	U	U	U	U
TITAN	TD	Y*	Y	DVPL	Y	F	Java & Python & R & C++	SOC

(e.g., meta-data about a CSV file); Second, *Data Processing*, that defines the behaviour of the processing stage (e.g., clean dataset, call an external API, etc.); Third, *Data Analysis* represents all the analytic operations (e.g. KNN, SVN, Bayesian Network, etc.); finally, *Data Sink*, whose instances describe the process of persisting data in databases or visualisation. To sum up, the BIGOWL core classes and hierarchy are depicted in Fig. 1.

RDF graphs describe specific data, algorithms, components, parameters, tasks, and workflows. These graphs consist of BIGOWL instances (class and property individuals). If new data types and/or new object properties are required to describe a specific workflow, the ontology can be extended accordingly.

A domain ontology contains terms that represent a particular knowledge domain. Domain knowledge is one of the crucial factors to get great success in coping with Big Data analytic as it improves this process. For example, suppose a classification analysis where the dataset contains continuous response variables and either continuous dependants, a reasoner could use this knowledge to recommend the use of linear regression algorithms for classification analysis because when a dataset with continuous variables is classified must be used linear regression.

With the help of semantics, we can reuse different algorithms, workflows, and data, as well as discover new knowledge employing semantic reasoning. For example, reasoning on domain knowledge is used in TITAN to detect inconsistencies and operational mistakes in the workflow. Besides, semantic is used in TITAN to allow the manual annotation of input data using domain ontologies, letting users indicate, for example, the semantic meaning of a column, a row or even a cell in a tabular dataset. Through this step, users can relate a domain ontology with its input data. The semantic knowledge represented as RDF triples is then stored in the platform's own RDF storage. Finally, semantic annotations help to maintain and generate provenance information for the domain of workflows and use cases [51–53].

3.2. Software architecture

TITAN consists of several software modules working together using BIGOWL as the pivotal element to ensure reusability and flexibility. It can be deployed on-premises using Docker, to quickly set up the entire platform on any server with minimal configuration. Additionally, an online deployment is freely available at https://titan.khaos.uma.es/.

TITAN considers a workflow as a set of tasks and links among them following an open modular design. The tasks are instances of the components with specific values for their input, output, and

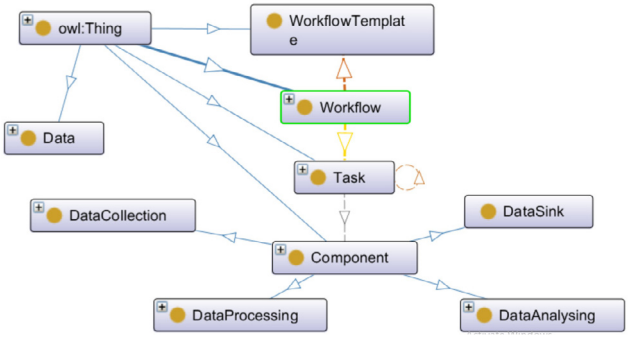

Fig. 1. Overview of the BIGOWL ontology. Continuous arrows refer to subclasses, whereas dotted ones refer to properties. BIGOWL provides core classes for defining any kind of workflow's elements as an RDF graph.

parameters. Therefore, this design allows re-using components in different workflows.

The motivation behind TITAN architecture is to provide a flexible way to integrate new technologies in the platform as it grows. Some critical design decisions were made at this point: first, to not restrict the components to be tied to a particular programming language; second, the component's input and output data must be easily associated with its corresponding semantics; finally, the definition, deployment, and execution of workflows should be assisted by a graphical tool. The current architecture of the TITAN platform is depicted in Fig. 2.

A ***graphical tool*** (web app) is provided to compose workflows interactively by drag-and-drop components from a catalogue. This tool allows users to interact with their data and workflows in their workspace. This web app is divided in different regions (Fig. 3). In the leftmost area, a list of available components (or catalogue) is shown. In the middle of the screen, we can find the canvas, where components can be dragged and dropped and connected to compose a workflow. The rightmost area is reserved for displaying the selected component's parameters (which can be updated by the user). Finally, a horizontal menu is located at the top of the screen. The components' catalogue is dynamically provided by BIGOWL, which is stored in an RDF triple store (*RDF storage*). In first releases, Stardog [54] was used as RDF database. Nevertheless, the limitations on the community version guided us to include a second choice for the RDF store: Virtuoso Community Edition. This way, the TITAN platform can be deployed with either database.

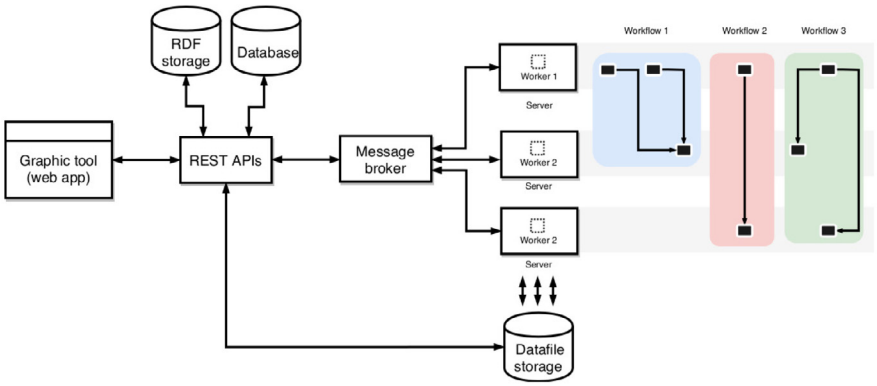

Fig. 2. Core TITAN platform's architecture is composed of a GUI, a REST API and an orchestrator for executing the workflows. The REST API requires an RDF storage and database solutions for data persistence. Besides, workflow executors also depends on a remote data file storage for storing component's outputs. Finally, a message broker is used for distributing messages between the components.

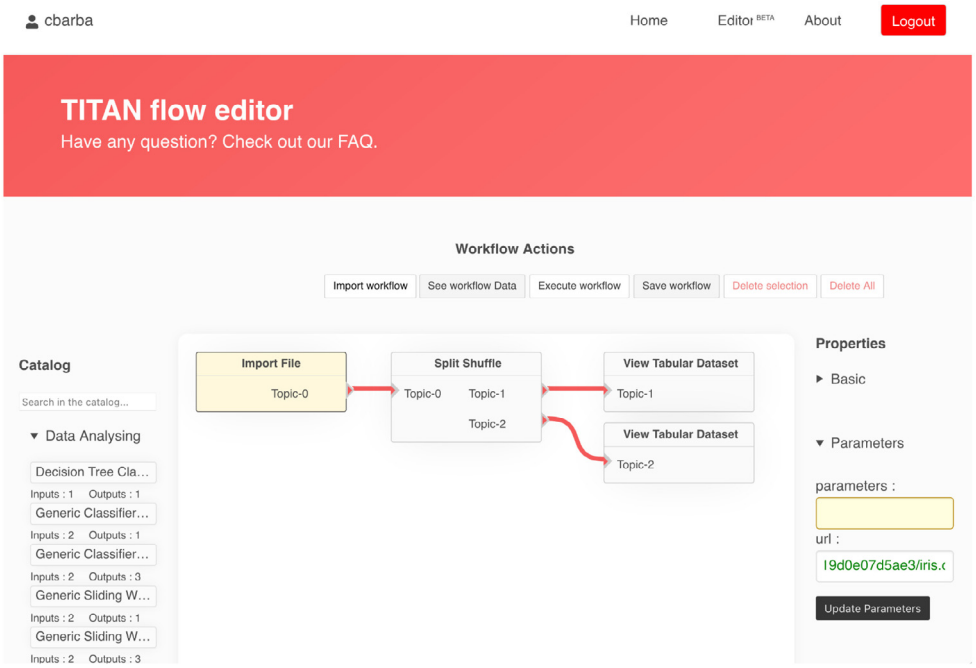

Fig. 3. Snapshot of the GUI tool. The components are on the left side of the picture. The right side shows the properties area. In the middle of the screen, we can find the canvas, where components can be dragged and dropped.

This web application is fed by ***REST APIs***. They are aimed at providing core functionalities to the platform (e.g., authentication, data persistence, semantic annotation of components, etc.) and to support the life cycle of a workflow, the visualisation of results (including the execution process of a workflow), and data management functionalities. The API methods send SPARQL queries to the RDF repository to retrieve required information. The workflows designed with the GUI (as well as all the user-related information) can be also stored in Virtuoso, through the API.

An ***orchestrator***, denoted by DRAMA,[6] is in charge of the execution of components using distributed workers. It exposes a private API that is queried by TITAN's API. This way, workflows triggered for execution are flattered into individual tasks that are sent to a highly-scalable and lightweight message broker (namely RabbitMQ [55]). DRAMA workers execute workflow tasks using multiple processes on a single machine or in a distributed fashion using a cluster of machines. A NoSQL *database*, MongoDB, is used to persist workflow execution state and other useful metadata.

To cope with a federated configuration, two aspects need to be considered: (1) components' inter-communication and (2) file processing. Components' inter-communication (i.e., passing data between components) is implemented with Apache Kafka [56], an scalable, fault-tolerant, and efficient distributed streaming platform. Apache Avro [57] helps us to deal with the heterogeneity of the components through its data serialisation system. The data description with Avro, through Avro schemes, are directly related to the component semantics. Binary files generated at any step of the workflow must be available to every other task which might not be executed on the same machine (*datafile storage*). Hence, HDFS [58] or MinIO[7] can be used as distributed file system (both available in TITAN).

6 In URL: https://github.com/KhaosResearch/drama.

7 https://min.io/

Listing 1: Python code example of creating a component in TITAN.

```
@dataclass
class Point(DataType):
    x: int = is_integer()
    y: int = is_integer()
    z: int = is_integer(default=0)


def execute(pcs: Process, z: int):
    """
    Generates random (x,y,z) points.
    """
    pcs.info([f"Generating point ({x},{y},0)"])
    x, y = [random.randint(0, 9) for _ in range(2)]
    point = Point(x=x, y=y, z=z)

    # send to downstream
    for _ in range(10):
        pcs.to_downstream(point)

    return {}
```

Listing 2: Python code example of reading data in TITAN.

```
def execute(pcs: Process):
    """
    Reads 3D-Point with cartesian coordinates.
    """
    inputs = pcs.get_from_upstream()

    for p in inputs["Point"]:
        pcs.info([f"Got {p}"])

    return {}
```

Listing 3: Python code example of reading data in a streaming way in TITAN.

```
def execute(pcs: Process):
    """
    Reads stream of 3D-Points with cartesian coordinates.
    """
    for key, p in pcs.poll_from_upstream():
        if key == pcs.inputs["Point"]:
            pcs.info([f"Got {p}"])
```

4. Component development and registration in TITAN

In this section, we provide basic knowledge on how to develop and register new components in TITAN.

4.1. Component design and code implementation

A set of design rules are defined to enable the integration of new components in TITAN. The primary programming language is Python, but it is possible to integrate components developed using other languages. DRAMA framework implements all the abstractions required to develop and test new components. In this sense, components must be understood as isolated Python functions with at least one input argument passed by the framework itself (following an arbitrary number of parameters): a *Process class* whose attributes and methods can be accessed from within the function.

Listing 1 illustrates a component that generates random (x, y, z) (with fixed z) points using DRAMA. Data, which are sent by a Task, must inherit from `DataType`, an abstract class that includes methods to generate the associated serialisation scheme for the data automatically (line 2) . The datatype `Point` holds the (x, y, z) coordinates of a point (lines 3 to 5). Its Avro schema is to be effortlessly generated for serialisation purposes when sending this data to downstream (lines 17 and 18).

A second component could read those points to log their values as it is shown in the Listing 2 lines 5 to 8. Or as a stream of data points as is depicted in the Listing 3 lines 5 to 7. The Process class is bounded to a unique task identifier generated at run-time, that will be used to create a named temporal file in the system for debugging messages, setup the data file storage, etc. As mentioned before, it provides methods to get upstream data from the preceded task(s) in the workflow, send downstream data, download and upload files, log messages, etc. Potentially, this function can spawn other processes to, for example, run a Docker container or execute scripts in different programming languages (acting as a "wrapper").

4.2. Component annotation

As explained in Section 3.1, all core classes and properties related to components, parameters and data in TITAN are already defined in BIGOWL ontology. Thus, components[8] in workflows are represented as instances of BIGOWL in the form of RDF graphs.

To create the RDF graphs as BIGOWL class and property instances or its possible extensions, we used the ontology editor Protégé.[9] In order to add a new kind of component, a knowledge engineer firstly seeks its corresponding category (Data Collection, Data Processing, Data Analysis, Data Sink or its sub-classes) in BIGOWL. Properties to describe the component based on its type are relevant to TITAN. The GUI and the APIs use them to provide useful information on the existing components and automatically instantiate the workflow execution's software components. This process is summed up as follows:

1. Look up the BIGOWL category to classify the new component. In case it is needed (if the existing abstract categories do not fit the new component features), add a new sub-class of *Component* class in BIGOWL. This new sub-class could also include new data or object properties.
2. Search in BIGOWL for both data and object properties related to the class representing the category chosen in Step 1.
3. For each object property, create a new instance of its range[10] class (or one of its sub-classes), intending to meet with all their specifications. Then, the user defines the required instances.

[8] TITAN instances graphs: https://github.com/KhaosResearch/TITAN-graphs.

[9] Protégé is an open-source ontology editor and a knowledge management system: https://protege.stanford.edu/.

[10] An object property range axiom ObjectPropertyRange(OPE CE) states that the range of the object property expression OPE is the class expression CE. If OPE connects some individual with an individual x, then x is an instance of CE.

If a different developer is registering an alternative implementation of this component, he/she will only need to create a new instance of the class *Implementation* instance referring it, and following the rest of the steps from here.
4. Create a new instance of the class created in the Step 1.
5. The definition of these instances could provoke the need for creating new ones according to the BIGOWL ontology. The user must then repeat steps 2 and 3 to finish the RDF graph for the new component.
6. Link all the instances from Step 3, using the selected object properties, with the instance created in Step 4.

Fig. 4 shows the definition of a split shuffle component using Protégé 5.5.0. This component is an instance of *SplitShuffle* class from BIGOWL. It contains two annotations, six object properties and two data properties declared in this instance.

After the definitions and descriptions of the components, as well as their data and object properties, the new RDF graph with the new instances must be updated in the system (as Virtuoso or Stardog graph database schemes). After this, the new components are available in the web interface ready to be tested (see Fig. 3). In the case of a team getting the project and deploying their own instance, they have to control all these steps. In the case of the instance available at https://titan.khaos.uma.es, developers can submit their code to the GIT repository and provide the TITAN updates through the support email.

5. TITAN usage

TITAN workflows are based on DAG representations according to which tasks are represented as nodes, and both data and control flow dependencies between them are represented as edges. TITAN workflows can be divided into two versions: the user's description of the workflow or *abstract workflow*, and the workflow that is executed or *concrete workflow*. In TITAN, abstract workflows are composed of components and their interconnections. After setting all the components and their parameters up, they are instantiated in independent executable (Kafka) tasks (connected through Kafka topics). Consecutively, concrete and executable workflows are created.

Therefore, we follow the same idea described in BIGOWL [7], in which a workflow is composed of a set of linked components belonging to three categories: publisher, subscriber, and processor, which allows us to generate, consume, and process data, respectively.

In this section, we illustrate how to design a workflow in TITAN using the components published in its semantic registry.

5.1. Workflow description

TITAN GUI is an easy-to-use web interface in charge of providing the users with a tool for interacting with the different functionalities included in the platform. Through this tool, the users can create their descriptions of the workflows and also executed them. TITAN is an Open Source project, so developers can deploy an instance on their own premises.

User Management. Only registered users have access to the working space where they can design and execute workflows besides manage their data. The instance deployed in our infrastructure allows users to register with a limited disc and computation usage, as this is a demonstration environment that does not aim to provide a computational service. Users are provided with an HDFS folder for storing their data and analysis results. This provides a scalable infrastructure for Big Data cases.

Catalogue. This section of the GUI depicts all the components that the users have available for designing a workflow. The components are shown in four main categories: Data Analysing, Data Collection, Data Processing and Data Sink. This catalogue can be also explored using a search engine using keywords, to enable users to fast locate components without having to browse through the catalogue. Users can drag any component and drop it into the design canvas. The option *Delete Selection* provides the feature of removing elements from the web interface.

Properties. In this section, TITAN GUI shows the properties and parameters of a selected component in the canvas. Once a user puts a component in the canvas, the values by default of the properties and parameters are loaded. These default values can be defined in BIGOWL. If they are not defined the corresponding value is shown as empty in the user interface.

Workflow Level Actions. Users have access to the workflows they have designed and also to the different executions of each of them. These workflows can be exported as JSON files (with the option *See Workflow Data*), so they can be shared with other users that can import them in their own environment.

With *Import Ontology*, the user can upload a domain ontology to the platform through TITAN GUI. Then, for example, columns from files like CSV or XLSX can be aligned with classes from this domain ontology. Thus, TITAN allows annotating data with domain knowledge that is defined in an ontology.

The functionality *Validate Workflow* checks whether the workflow is semantically correct by means of SPARQL queries or SWRL rules, which are included in BIGOWL [7].

And finally, *Execution Workflow* transforms the user's description of the workflow in an executable workflow. Each component from the JSON file is transformed into a RabbitMQ job.

5.2. Workflow execution

DRAMA framework is the underlying workflow orchestrator engine used by TITAN. There are two ways of executing workflow in DRAMA. Firstly, a workflow can be executed programmatically (i.e., using the `drama` package for Python 3.7) by passing a list of `TaskRequest` to a `WorkflowRequest` object. Each `TaskRequest` describes the task name, the module where the function can be found, optional parameters, and inputs, among others. Moreover, it is possible to see the execution status at any time using the task or workflow manager.

Secondly, it can also be deployed as a server. The API exposes an endpoint to execute a workflow given a list of tasks and their connections in JSON format. A second endpoint can be used to retrieve workflow execution status.

Whatever method is used, the workflow request is processed by DRAMA and unique workflow and tasks identifiers are generated. Then, tasks are send to the message broker, so that workers can get those messages and execute the main function by importing the corresponding task module.

During this execution process, TITAN automatically annotates the tasks (that are instances of the components), parameters, data, and workflows. Through all those annotations (using RDF triples), TITAN keeps or creates data traceability not only for data themselves, but also for parameters, tasks, workflows, and their results in each step and the final workflow result.

6. Use cases

In order to show a better understanding of the platform and how its modules collaborate between them for facilitating all the steps that are involved in the process of creating an executable workflow, three different use cases are described in this section.

- The first one is focused on classic data mining analysis on academic problem instances.

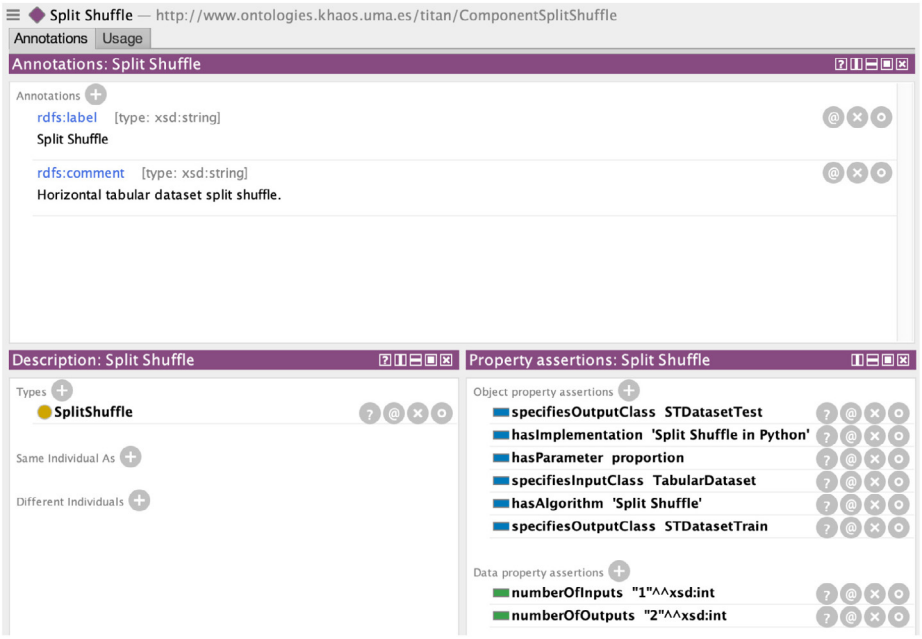

Fig. 4. Definition of the split shuffle component using Protégé 5.5.0.

- The second case study is centred on Big Data analysis on human activity recognition using a deep learning algorithm.
- The third case study makes an analysis of a region of interest for classifying the changing detection in satellite images using a supervised machine learning algorithm.

In this way, we aim at covering, as much as possible, different aspects in Big Data applications: algorithmic analyses (machine learning and deep learning), velocity and volume issues (streaming processing), real-world and academic data problems, and Big Data ecosystems (Apache Spark and Apache Kafka).

Each of these use cases is composed by steps that can be defined with the help of BIGOWL as components encapsulated within tasks. In order to materialise the components defined with this ontology in executable scripts (written in Java, Python, R, etc.), we follow the next methodology:

1. The TITAN components are mapped to one of the four types of components defined by BIGOWL, namely *DataAnalysing*, *DataCollection*, *DataProcessing*, and *DataSink*, e.g., a component that loads data from an external source is defined as a *DataCollection* component.
2. We define the input *Data* as the super class of all the classes representing the different types of data (e.g. *DataSet*, *DataTable*, etc.). These data are associated with Avro schemes, which describe the fields allowed in the record and its data type.
3. In this step the component parameters are defined. All of them are instances of the BIGOWL *Parameter* class.
4. The goal of the fourth step is to define as RDF the workflow tasks, which are components with specific data for tackling a specific problem (e.g. form where we can load data, configuration parameters for a component, etc.).
5. Finally, we use the SPARQL queries in Virtuoso for assessing the correctness of the composed workflow, checking whether the task instances are mutually compatible among them throughout the complete workflow. Furthermore, if instead of Virtuoso, Stardog is used, TITAN allows reasoning over the compatibility and correctness in the workflow (this process is fully described in [7]).

6.1. Case study 1: Iris flower classification

In this section, we describe an example of a machine learning workflow implemented using the proposed platform. The idea is to predict a response variable from exploratory observations. In particular, the aim is to classify the different species of the Iris flower from a dataset consisting of 50 observations from each of the three species (Setosa, Versicolour, and Virginica).[11]

This workflow is generated with TITAN, as illustrated Fig. 5, which comprises the following components: (1) the dataset is loaded. Then, (2) it splits into train and test data. After that, (3) train data are used to train a machine learning model, and (4) both test data and the model are used to test the classifier's prediction. Finally, (5) some useful metrics derived from the model are highlighted.

Taking this into account, we can semantically annotate each of the previous steps as separate tasks, linked together in a specific order:

- The first task encompasses a *Import tabular dataset* component, which loads the Iris dataset. This component specifies one output class of type *GenericDataSet*.
- The second task uses a *Split shuffle* component to split the dataset into train and test, whose input and output classes are *GenericDataSet* and has one parameter to indicate the ratio of the splitting process.
- A third task uses a *KNN Classifier Train* component with one input class of type *GenericDataSet* and one output class of type *TrainModelKNN*, and several parameters for tuning the model.
- The fourth task requires as input both, a machine learning model (*KNN model*) and test data (type *GenericDataSet*) to validate the classifier and returns the validation results (type *ValidationResults*), by means of *Validate model* component.
- The *ValidationResults* from the previous task is consumed by a *VisualizeResults* component, to visualise the metrics.

11 This dataset is free and is publicly available at the UCI Machine Learning Repository: https://archive.ics.uci.edu/.

Fig. 5. Snapshot of the workflow of the classification of the Iris dataset in the graphical tool. The first step is when the dataset is loaded. Then, in the second step, the data are split into train and test data. After that, as the third step, train data are used to train a machine learning model and, in the fourth step, both test data and the model are used to test the classifier's prediction. Finally, some useful metrics derived from the model are highlighted.

Listing 4: First step of SPARQL queries for case study of Supervised Classification for Iris dataset.

```
SELECT DISTINCT
  (COUNT (?parameterTask) = COUNT(?parameterComp)
  as ?result)
WHERE {
    titan:WorkflowIris rdf:type dmop:Workflow .
    titan:WorkflowIris bigowl:hasTask ?task .
    ?task rdf:type bigowl:Task .
    ?task bigowl:hasComponent ?comp .
    OPTIONAL {
        ?task bigowl:hasParameter ?parameterTask .
        ?comp bigowl:hasParameter ?parameterComp .
        ?parameterTask bigowl:instanceParameterOf
        ?parameterComp .
    }
}
GROUP BY ?parameterTask ?parameterComp
```

Listing 5: Second step of SPARQL queries for case study of Supervised Classification for Iris dataset.

```
% SELECT DISTINCT ?task1 ?task2
% WHERE {
% titan:WorkflowIris rdf:type dmop:Workflow .
% titan:WorkflowIris bigowl:hasTask ?task1 .
% titan:WorkflowIris bigowl:hasTask ?task2 .
% ?task1 rdf:type bigowl:Task .
% ?task2 rdf:type bigowl:Task .
% ?task1 bigowl:connectedTo ?task2 .
% ?task1 bigowl:hasComponent ?comp1 .
% ?task2 bigowl:hasComponent ?comp2 .
% OPTIONAL { ?comp1 rdfs:label ?name1 . }
% OPTIONAL { ?comp2 rdfs:label ?name2 . }
% ?comp1 bigowl:specifiesOutputClass ?outputC .
% ?comp2 bigowl:specifiesInputClass ?inputC .
% ?task1 bigowl:specifiesOutputClass ?outputTask .
% ?task2 bigowl:specifiesInputClass ?inputTask .
% ?inputC rdf:type ?class .
% ?outputC rdf:type ?classOC .
% ?outputTask rdf:type ?classOT .
% ?inputTask rdf:type ?classIT .
% ?classOC rdfs:subClassOf* ?class .
% ?classOT rdfs:subClassOf* ?class .
% ?classIT rdfs:subClassOf* ?class .
% FILTER (?class!=owl:NamedIndividual)
% }
% GROUP BY ?task1 ?task2
```

Fig. 6 depicts how the Iris dataset is defined through a RDF graph using the two ontologies above (i.e., BIGOWL, and domain ontology). Iris dataset properties are designated using BIGOWL properties. Its columns are connected with the domain ontology through BIGOWL properties *represents* and *hasSemanticMeaning*.

For this workflow, Listing 4 and Listing 5 correspond to the required SPARQL queries applied by the semantic model to check the correctness and task compatibility of the workflow. These queries are automatically generated from the TITAN-GUI when the validation button is clicked. The first step (1) assesses whether the Iris workflow has declared all the tasks' parameters. The second SPARQL query (2) assesses the correctness of the workflow as a whole. These SPARQL queries are based on the ideas described in [7].

6.2. Case study 2: Human activity recognition

Human activity recognition (HAR) has received increasing interest in many applications such as health care, where Big Data have experienced significant growth in generating data and producing analytical software solutions. In this use case, a volume of 30 TBs of data is collected using accelerometers implanted in wearable devices. The classification includes different types of movements: short actions such as household activities (e.g., meal preparation, room cleaning), long actions (e.g., walking or running), rest (e.g., sitting down, sleeping), and finally, non-wearing the wearable.

HAR is tackled here by using a blend of both two deep learning algorithms: Convolutional Networks (ConvNet) and Bidirectional LSTM [59], or *RecNetLSTM* for short. Nevertheless, driven by the strict requirement of managing such a large amount of data, Apache Spark is used in this case study as a parallel platform for model execution.

This workflow can be arranged in ten steps (as shown in Fig. 7). The three firsts consist of training the model, and remaining ones are devoted to the activity classification. First of all, (1) the datasets of the 11 activities are loaded and all the CSV are jointed in only one that includes a new column with the activity that represents. Then (2), the new CSV is split into two CSVs, train and test. After that, (3) the model is trained with the RecNetLSTM algorithm. In step (4), the data from the wearable is loaded in CSV format. After that (5), the model from the step 3 and data from step 4 are classified. Then (6) the classification is shown. In parallel (7), the model from step 3 and the test data from step 2 are loaded and the model is assessed. Finally, the testings results are depicted in the graphical tool.

Fig. 8 depicts the different activities carried out by a sample patient during the day. We can see how during the night resting is the main activity (although there are some movements around 4:00 and 5:00), later around 8:00 the patient starts to be more active and does short and long moves and finally, after 21:00 resting is the main activity again.

Each step is annotated as individual tasks, linked together in a specific order:

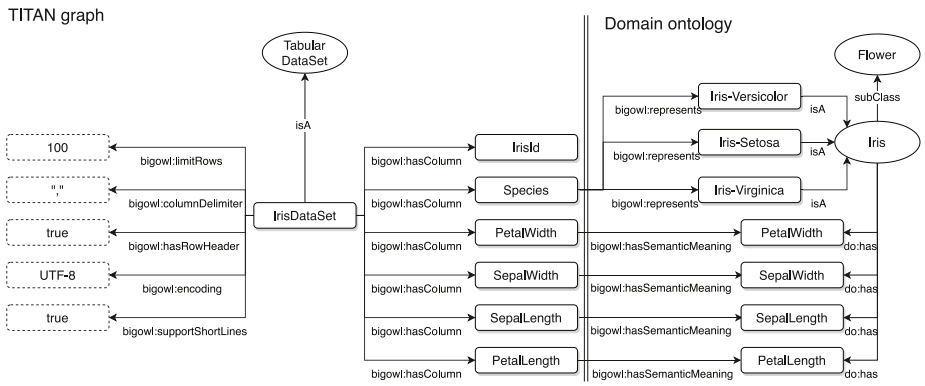

Fig. 6. Iris dataset definition through RDF graphs using BIGOWL and Iris Domain ontologies. *bigowl* and *do* prefixes indicates properties from BIGOWL and Domain ontology, respectively (dotted-boxes represents data properties in the ontology).

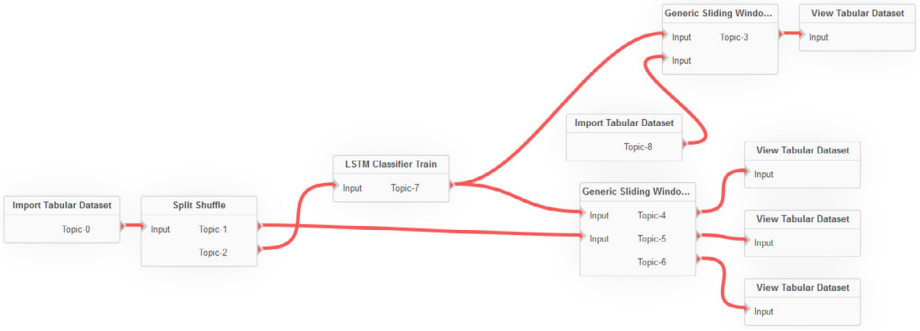

Fig. 7. Snapshot of the classification workflow for the Human Activity Recognition in the TITAN-GUI.

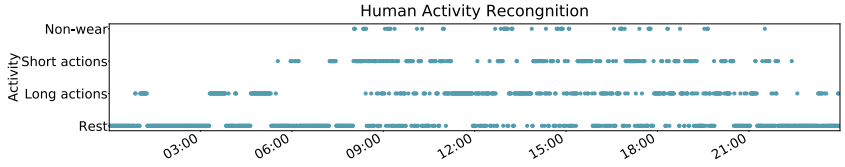

Fig. 8. Snapshot of the Human Activity Recognition in the graphical tool. It is shown how during the night resting is the main activity, later around 8:00, the patient starts to be more active and does short and long movements and finally, after 21:00 resting is the main activity.

- The first task encompasses a *Read zip file* component, which loads a compressed file containing several CSV files for each activity. This component specifies one output class of type *ZipFile*. It will unzip it to merge the CSVs into one. This component generates a single annotated CSV file as output class (*GenericDataSet*).
- The second task *SplitShuffle* divides the *GenericDataSet* into two *GenericDataSet*.
- A third task uses a *RecNetLSTM Training* component with one input class of type *GenericDataSet* and one output class of type *TrainModelRecNetLSTM*, and some parameters for tuning the model.
- The fourth task reads a new *GenericDataSet*, which is used for classifying its activities.
- The fifth task requires as input both a machine learning model *TrainModelRecNetLSTM* and CSV data from the fourth step (type *GenericDataSet*) to predict the data and return the prediction results (*GenericDataSet*).
- The sixth task encompasses a *Visualise Tabular Datase* component, which depicts a CSV file with the prediction.
- The seventh task requires as input both a machine learning model (*TrainModelRecNetLSTMmodel*) and CSV data from the third step (type *GenericDataSet*) to assess the data and return the testing results (*GenericDataSet*).
- The eighth, ninth and tenth tasks, *Visualise* provides to TITAN a task for showing the testing results.

6.3. Case study 3: Automatic monitoring of Earth Observation Satellite images

This workflow aims to automatically classify regions of interest in the context of Earth Observation using open-data multispectral images taken by satellite constellation Sentinel-2.[12]

This workflow can be described in eight steps as follows (see Fig. 9): (1) the satellite images (in data products) are downloaded from the Copernicus website.[13] Then, (2) the images are preclassified with QGIS code to indicate the localisation of each pixel. Next, (3) the dataset is loaded for its classification. Then, (4) a pre-process is made to remove outlier data. Following, (5) data is split into training and testing data. After that (6), training data is used to train an SVM model, and (7) both testing data and the model are used to test the prediction of the classifier. Finally, (8) some useful metrics derived from the model are highlighted.

12 https://sentinel.esa.int/web/sentinel/missions/sentinel-2

13 https://www.copernicus.eu/en/accessing-data-where-and-how/conventional-data-access-hubs

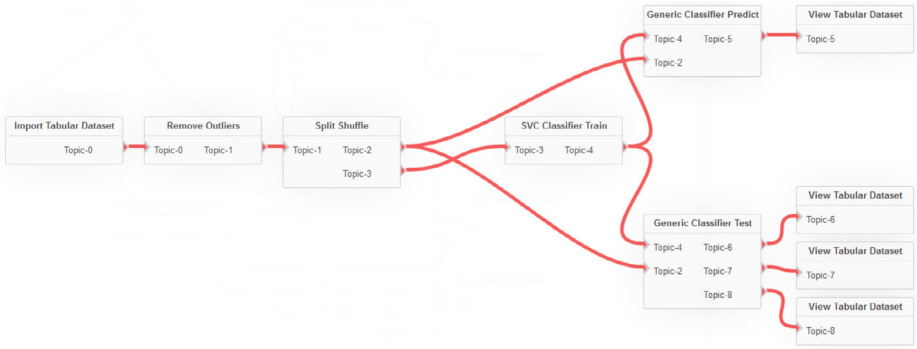

Fig. 9. Snapshot of the workflow of the classification of the Automatic monitoring and change detection on Earth surface in the graphical tool.

To test this workflow for classifying an entire region, we conduct an experiment with real data from Íznajar reservoir in Córdoba (Spain). The goal of this example is to distinguish between vegetation (green area), buildings, water and dry land (land without vegetation). Fig. 10 depicts four images, where the real colour image is shown on the top left, brands predictions on the top right, index prediction on the bottom left, and finally QGIS classification on the bottom right.

Taking the aforementioned steps into account, each of the previous steps are semantically annotated as separate tasks, linked together in a specific order:

- The first task encompasses a *Read tabular dataset* component, which loads the image dataset. This component specifies as input and output class of type *GenericDataSet*.
- The second task *Remove outlier* deletes the rows in the dataset whose class calculated in step 2 has been erroneous. This tasks follows the *Mahalanobis distance* [60] for detecting the outliers.
- The third task uses a *Split dataset* component to split the dataset into a train and test. TITAN reuses the components, so this is the same that it has been used in the Use Case 1 and whose input and output class is a *GenericDataSet*, furthermore, has one parameter to indicate the ratio of the splitting process.
- The fourth task uses a *SVM Training* component with one input class of type *GenericDataSet* and one output class of type *TrainModelSVM*, as well as several parameters for tuning the model.
- The fifth task requires two inputs, a machine learning model (*SVM model* model) and test data (type *GenericDataSet*) in order to use the classifier and return the predict results (type *Tabular Dataset*). Intending to manage this hard-demanding in execution time classification, TITAN provides us with the Python libraries: Multiprocessing and Pandas, which let us parallelisation the classification.
- The last task, has two input, the model *TrainModelSVM* and the *GenericDataSet* from step three for assessing the model. The results are consumed by three *Visualise Results* components, to visualise the metrics.

7. Discussion

One of the leading contributions we claim with the design and implementation of TITAN is the ability to provide the tools for describing, designing, developing, annotating, re-using, and running workflows involving Big Data analytics. All the elements of TITAN are described following the classes and properties defined in BIGOWL ontology, and this allows us to annotate all the metadata flowing from multiple data sources, processing components, and analytic algorithms.

The results obtained in the three case studies indicate that TITAN allows creating separate components that can be used or re-used in countless different workflows. TITAN provides us with DRAMA API, that contains the necessary software architecture for developing any workflow component such as analysing components (KNN, SVM, etc.), sink components (store in MongoDB or HDFS, etc.) or even of loading data (read CSV files o Mongo resources, etc.).

TITAN-API and BIGOWL enable to progressively deliver component recommendations for the construction of Big Data analytics workflows. The resulting workflows are indeed enhanced with knowledge that explicitly describes and registers the data lineage (data provenance in database systems), from sources to results, as well as checking errors from workflow designing to its execution time. It is worth mentioning that in BIGOWL, the semantic model and data linage is mapped with RDF triples. They refer to records of the inputs, entities, systems, algorithms, and processes that influence data of interest, hence providing a historical record of the data obtained (as results) and its origins (as sources).

8. Concluding remarks and future work

This work presents TITAN, a platform for the generation and deployment of data science workflows for Big Data analytics. TITAN is featured by a design and operation mode driven by semantics at different levels: data source, problem domain, and workflow component. The semantic annotation of all the metadata involved in a Big Data workflow offers a series of main advantages, such as semantic validation of workflow composition, the possibility of linking knowledge domain data with processing components, as well as monitoring and keeping data quality and consistency at any step of the data life cycle.

As future lines of research, the definition of a methodology to capture and represent expert knowledge, when there are not existing ontologies, opens the possibility to generate new families of algorithms of Big Data analytics, in which the knowledge's domain changes from being introduced *ad hoc* (or extracted by some mechanism), to be introduced in a (semi-)automatic way. The challenge will be the development of mechanisms that facilitate this task to domain experts and, besides, facilitate the reuse of different algorithms that use the same techniques through a homogeneous mechanism to introduce the knowledge of the domain. Furthermore, it can also lead to improvements in the obtained results of the analysis process by improving their quality or the efficiency of the algorithms [61,62]

Future work entails as well as the generation of numerous sample workflows oriented to Big Data streaming processing and analysis. All these new samples will increment the number of ready-to-use components in the internal catalogue of TITAN, which will be indeed made available for scientific and industry communities.

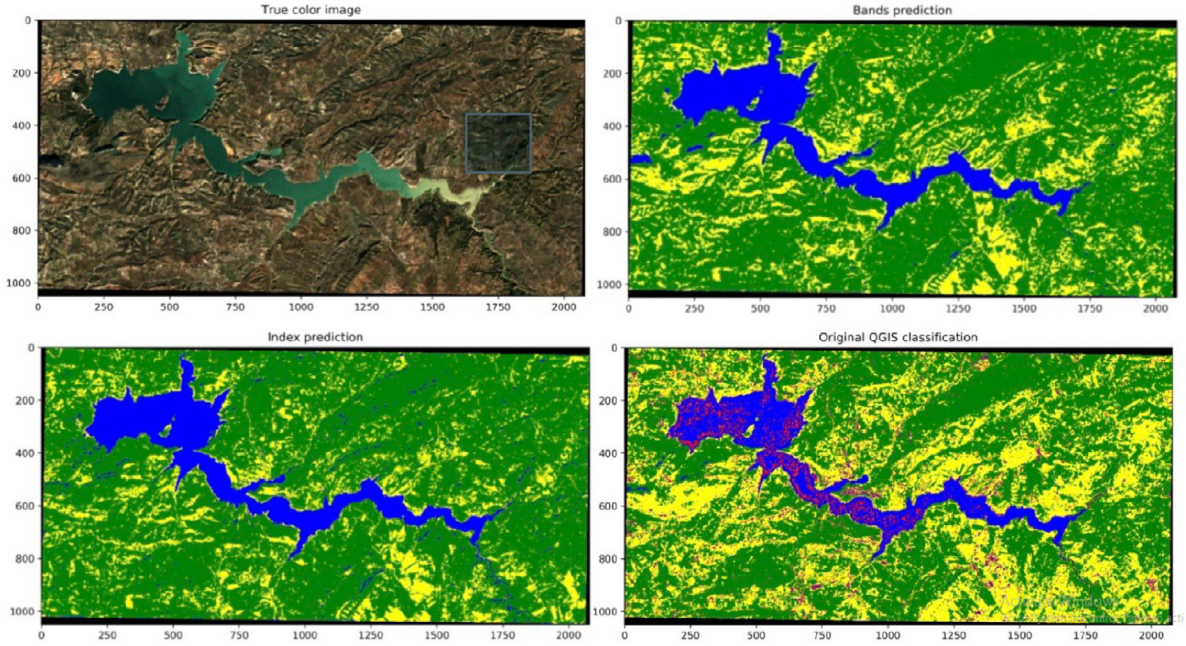

Fig. 10. Snapshot of the classification result of the Íznajar reservoir in TITAN. The real colour image is shown on the top left, brands predictions on the top right, index prediction on the bottom left, and finally QGIS classification on the bottom right.

Another line of future work consists in developing mechanisms for easing the process of adding new components, so developers can include them and directly test them in the deployment available at https://titan.khaos.uma.es. This new development will be included in the Open Source project so it can be used also in other deployments of TITAN.

CRediT authorship contribution statement

Antonio Benítez-Hidalgo: Software, Validation, Writing – review & editing. **Cristóbal Barba-González:** Methodology, Validation, Formal analysis, Writing – review & editing. **José García-Nieto:** Methodology, Validation, Formal analysis, Writing – review & editing. **Pedro Gutiérrez-Moncayo:** Software, Validation. **Manuel Paneque:** Software, Validation. **Antonio J. Nebro:** Validation, Writing – review & editing. **María del Mar Roldán-García:** Validation, Writing – review & editing. **José F. Aldana-Montes:** Funding acquisition. **Ismael Navas-Delgado:** Conceptualization, Validation, Writing – review & editing, Supervision, Project administration.

Declaration of competing interest

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

Funding

All of the sources of funding for the work described in this publication are acknowledged in the manuscript.

Intellectual property

We confirm that we have given due consideration to the protection of intellectual property associated with this work and that there are no impediments to publication, including the timing of publication, with respect to intellectual property. In so doing, we confirm that we have followed the regulations of our institutions concerning intellectual property.

References

[1] A. Zomaya, S. Sakr, Handbook of Big Data Technologies, first ed., Springer International Publishing, 2017.
[2] R. Elshawi, S. Sakr, D. Talia, P. Trunfio, Big data systems meet machine learning challenges: Towards big data science as a service, Big Data Res. 14 (2018) 1–11, http://dx.doi.org/10.1016/j.bdr.2018.04.004.
[3] A. Neilson, Indratmo, B. Daniel, S. Tjandra, Systematic review of the literature on big data in the transportation domain: Concepts and applications, Big Data Res. (2019).
[4] W. Raghupathi, V. Raghupathi, Big data analytics in healthcare: promise and potential, Health Inf. Sci. Syst. 2 (1) (2014) 3.
[5] I.J. Taylor, E. Deelman, D.B. Gannon, M. Shields, et al., Workflows for e-Science: Scientific Workflows for Grids, vol. 1, Springer, 2007.
[6] S. Wolfert, L. Ge, C. Verdouw, M.-J. Bogaardt, Big data in smart farming – A review, Agricult. Syst. 153 (2017) 69–80, http://dx.doi.org/10.1016/j.agsy.2017.01.023.
[7] C. Barba-González, J. García-Nieto, M. Roldán-García, I. Navas-Delgado, A. Nebro, J. Aldana-Montes, BIGOWL: Knowledge centered big data analytics, Expert Syst. Appl. 115 (2019) 543–556.
[8] Y. Gil, V. Ratnakar, J. Kim, P.A. Gonzalez-Calero, P. Groth, J. Moody, E. Deelman, Wings: Intelligent workflow-based design of computational experiments, IEEE Intell. Syst. 26 (1) (2011).
[9] A. Konys, Ontology-based approaches to big data analytics, in: International Multi-Conference on Advanced Computer Systems, Springer, 2016, pp. 355–365.
[10] E.W. Kuiler, From big data to knowledge: an ontological approach to big data analytics, Rev. Policy Res. 31 (4) (2014) 311–318.
[11] T.R. Gruber, A translation approach to portable ontology specifications, Knowl. Acquis. 5 (2) (1993) 199–220, http://dx.doi.org/10.1006/knac.1993.1008.
[12] R. McClatchey, A. Branson, J. Shamdasani, Z. Kovacs, et al., Designing traceability into big data systems, 2015, arXiv preprint arXiv:1502.01545.
[13] T.A.S. Siriweera, I. Paik, B.T. Kumara, QoS and customizable transaction-aware selection for big data analytics on automatic service composition, in: 2017 IEEE International Conference on Services Computing, SCC, IEEE, 2017, pp. 116–123.
[14] T. Akila, I. Paik, S. Siriweera, QoS-aware rule-based traffic-efficient multiobjective service selection in big data space, IEEE Access 6 (2018) 48797–48814.
[15] M. Kleppmann, J. Kreps, Kafka, samza and the unix philosophy of distributed data, IEEE Data Eng. Bull. 38 (4) (2015) 4–14.
[16] M. Zaharia, M. Chowdhury, M. Franklin, S. Shenker, I. Stoica, Spark: Cluster computing with working sets, in: Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing, HotCloud'10, USENIX Association, 2010, p. 10.

[17] N.F. Noy, D.L. McGuinness, et al., Ontology Development 101: A Guide to Creating Your First Ontology, Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, Stanford, CA, 2001.
[18] B. McBride, The resource description framework (RDF) and its vocabulary description language RDFS, in: Handbook on Ontologies, Springer, 2004, pp. 51–65.
[19] S. Staab, R. Studer, Handbook on Ontologies, Springer Science & Business Media, 2013.
[20] S. Harris, A. Seaborne, E. Prud'hommeaux, SPARQL 1.1 query language, 2013, W3C Recommendation 21 (10).
[21] E. Prud, A. Seaborne, et al., SPARQL query language for RDF, 2006, W3C Recommendation.
[22] R. Mitchell, L. Pottier, S. Jacobs, R.F. da Silva, M. Rynge, K. Vahi, E. Deelman, Exploration of workflow management systems emerging features from users perspectives, in: 2019 IEEE International Conference on Big Data, Big Data, IEEE, 2019, pp. 4537–4544.
[23] C.E. Scheidegger, H.T. Vo, D. Koop, J. Freire, C.T. Silva, Querying and re-using workflows with vstrails, in: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD '08, ACM, New York, NY, USA, 2008, pp. 1251–1254.
[24] J. Freire, C.T. Silva, S.P. Callahan, E. Santos, C.E. Scheidegger, H.T. Vo, Managing rapidly-evolving scientific workflows, in: International Provenance and Annotation Workshop, Springer, 2006, pp. 10–18.
[25] T.M. McPhillips, T. Song, T. Kolisnik, S. Aulenbach, K. Belhajjame, K. Bocinsky, Y. Cao, F. Chirigati, S.C. Dey, J. Freire, D.N. Huntzinger, C. Jones, D. Koop, P. Missier, M. Schildhauer, C.R. Schwalm, Y. Wei, J. Cheney, M. Bieda, B. Ludäscher, YesWorkflow: A user-oriented, language-independent tool for recovering workflow information from scripts, 2015, CoRR abs/1502.02403. arXiv:1502.02403. URL http://arxiv.org/abs/1502.02403.
[26] M.R. Berthold, N. Cebron, F. Dill, T.R. Gabriel, T. Kötter, T. Meinl, P. Ohl, K. Thiel, B. Wiswedel, KNIME-the Konstanz information miner: version 2.0 and beyond, AcM SIGKDD Explorations Newsl. 11 (1) (2009) 26–31.
[27] A.A. Documentation, Apache airflow documentation-airflow documentation, 2019.
[28] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. Nieva de la Hidalga, M.P. Balcazar Vargas, S. Sufi, C. Goble, The Taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud, Nucleic Acids Res. 41 (W1) (2013) 557–561.
[29] D. Turi, P. Missier, C. Goble, D. De Roure, T. Oinn, Taverna workflows: Syntax and semantics, in: Third IEEE International Conference on E-Science and Grid Computing, E-Science 2007, IEEE, 2007, pp. 441–448.
[30] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P.J. Maechling, R. Mayani, W. Chen, R.F. Da Silva, M. Livny, et al., Pegasus, a workflow management system for science automation, Future Gener. Comput. Syst. 46 (2015) 17–35.
[31] B.T.G.S. Kumara, I. Paik, J. Zhang, T.H.A.S. Siriweera, K.R.C. Koswatte, Ontology-based workflow generation for intelligent big data analytics, in: 2015 IEEE International Conference on Web Services, 2015, pp. 495–502.
[32] Y. Gil, K. Cobourn, E. Deelman, C. Duffy, R. Ferreira da Silva, A. Kemanian, C. Knoblock, V. Kumar, S. Peckham, L.A. Carvalho, et al., MINT: model integration through knowledge-powered data and process composition, 2018.
[33] S.D. Peckham, The CSDMS standard names: Cross-domain naming conventions for describing process models, data sets and their associated variables, 2014.
[34] M. Albrecht, P. Donnelly, P. Bui, D. Thain, Makeflow: A portable abstraction for data intensive computing on clusters, clouds, and grids, in: Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies, 2012, pp. 1–13.
[35] R. Mecklenburg, Managing Projects with GNU Make: The Power of GNU Make for Building Anything, O'Reilly Media, Inc., 2004.
[36] T. Tannenbaum, D. Wright, K. Miller, M. Livny, Condor: a distributed job scheduler, in: Beowulf Cluster Computing with Windows, 2001, pp. 307–350.
[37] M.D. Wilkinson, M. Links, BioMOBY: an open source biological web services proposal, Brief. Bioinform. 3 (4) (2002) 331–341.
[38] M. Bhandarkar, MapReduce programming with apache Hadoop, in: 2010 IEEE International Symposium on Parallel & Distributed Processing, IPDPS, IEEE, 2010, p. 1.
[39] V.K. Vavilapalli, A.C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, et al., Apache hadoop yarn: Yet another resource negotiator, in: Proceedings of the 4th Annual Symposium on Cloud Computing, 2013, pp. 1–16.
[40] M. Zaharia, R.S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M.J. Franklin, et al., Apache spark: a unified engine for big data processing, Commun. ACM 59 (11) (2016) 56–65.
[41] M. Atkinson, S. Gesing, J. Montagnat, I. Taylor, Scientific workflows: Past, present and future, 2017.
[42] J.A. Novella, P. Emami Khoonsari, S. Herman, D. Whitenack, M. Capuccini, J. Burman, K. Kultima, O. Spjuth, Container-based bioinformatics with Pachyderm, Bioinformatics 35 (5) (2019) 839–846.
[43] M. Palankar, A. Iamnitchi, M. Ripeanu, S. Garfinkel, Amazon S3 for science grids: a viable solution? in: Proceedings of the 2008 International Workshop on Data-Aware Distributed Computing, 2008, pp. 55–64.
[44] B. Wilder, Cloud Architecture Patterns: Using Microsoft Azure, O'Reilly Media, Inc., 2012.
[45] A.K. Mishra, J.L. Hellerstein, W. Cirne, C.R. Das, Towards characterizing cloud backend workloads: insights from Google compute clusters, ACM SIGMETRICS Perform. Eval. Rev. 37 (4) (2010) 34–41.
[46] P. Di Tommaso, M. Chatzou, E.W. Floden, P.P. Barja, E. Palumbo, C. Notredame, Nextflow enables reproducible computational workflows, Nature Biotechnol. 35 (4) (2017) 316–319.
[47] D. Merkel, Docker: lightweight linux containers for consistent development and deployment, Linux J. 2014 (239) (2014) 2.
[48] G.M. Kurtzer, Singularity 2.1. 2-Linux application and environment containers for science, 2016.
[49] V. Korkhov, D. Vasyunin, A. Wibisono, A.S. Belloum, M.A. Inda, M. Roos, T.M. Breit, L.O. Hertzberger, VLAM-G: Interactive data driven workflow engine for Grid-enabled resources, Sci. Program. 15 (3) (2007) 173–188.
[50] Y.L. Simmhan, B. Plale, D. Gannon, Karma2: Provenance management for data-driven workflows, Int. J. Web Serv. Res. 5 (2) (2008) 1–22.
[51] B. Cao, B. Plale, G. Subramanian, P. Missier, C. Goble, Y. Simmhan, Semantically annotated provenance in the life science grid, in: Proceedings of the First International Conference on Semantic Web in Provenance Management-Volume 526, CEUR-WS. org, 2009, pp. 17–22.
[52] N. Del Rio, P.P. Da Silva, A.Q. Gates, L. Salayandia, Semantic annotation of maps through knowledge provenance, in: International Conference on GeoSpatial Sematics, Springer, 2007, pp. 20–35.
[53] C. Halaschek-Wiener, J. Golbeck, A. Schain, M. Grove, B. Parsia, J. Hendler, Annotation and provenance tracking in semantic web photo libraries, in: International Provenance and Annotation Workshop, Springer, 2006, pp. 82–89.
[54] K. Cerans, G. Barzdins, R. Liepins, J. Ovcinnikova, S. Rikacovs, A. Sprogis, Graphical schema editing for stardog OWL/RDF databases using OWLGrEd/S, in: OWLED, Vol. 849.
[55] D. Dossot, RabbitMQ Essentials, Packt Publishing Ltd, 2014.
[56] J. Kreps, N. Narkhede, J. Rao, Kafka: A distributed messaging system for log processing, in: Proceedings of 6th International Workshop on Networking Meets Databases (NetDB), Athens, Greece, 2011.
[57] D. Vohra, Apache avro, in: Practical Hadoop Ecosystem: A Definitive Guide to Hadoop-Related Frameworks and Tools, A Press, Berkeley, CA, 2016, pp. 303–323.
[58] D. Borthakur, et al., HDFS architecture guide, Hadoop Apache Project 53 (1–13) (2008) 2.
[59] S. Hurtado Requena, C. Barba-González, M. Rybinski, F.J. Baron-Lopez, J. Wärnberg, I. Navas-Delgado, J.F. Aldana-Montes, et al., Análisis de los datos del acelerómetro para detección de actividades, 2018.
[60] R. De Maesschalck, D. Jouan-Rimbaud, D.L. Massart, The mahalanobis distance, Chemometr. Intell. Lab. Syst. 50 (1) (2000) 1–18.
[61] W. Song, S.C. Park, Genetic algorithm for text clustering based on latent semantic indexing, Comput. Math. Appl. 57 (11–12) (2009) 1901–1907.
[62] C.B. González, J. García-Nieto, I.N. Delgado, J.F.A. Montes, A fine grain sentiment analysis with semantics in tweets, IJIMAI 3 (6) (2016) 22–28.