Abstract—Cloud computing has been recognized as the most prominent way for hosting and delivering services over the Internet. A plethora of cloud service offerings are currently available and are being rapidly adopted by small and medium enterprises but also by larger organizations based on their many superiorities to traditional computing models. However, at the same time the computing requirements of the modern cloud application has been exponentially increased due to the available big data for processing. Nowadays, we discuss the emerging data-intensive applications that necessitate the wide adoption of multi-cloud deployment models, in order to use all the advantages of cloud computing without any restrictions with respect to who is providing infrastructural services. In this paper, we discuss a Metadata Schema for data-aware multi-cloud computing which aspires to form the appropriate background vocabulary that will aid the big data-aware application deployment for distributed and loosely-coupled multi-cloud applications.

Keywords—Metadata Schema, Multi-Cloud Computing, Big Data

I. INTRODUCTION

Over the last years, there is a struggle to auspiciously exploit the plethora of cloud service offerings from a vastly increasing number of cloud providers. Up to a certain extent this has been successfully addressed with respect to making selection decisions on software or platform-as-a-service offerings from different providers [1]. Nevertheless, the use of multi-cloud offerings especially at the level of infrastructure in order to cope with the needs of big data applications still remains a goal. Such applications have to efficiently deal with the volume, variety, velocity, and veracity of the data, using any resources available in a cost-effective and efficient way. In this respect, there is an increasing need for data-intensive computing in highly distributed and federated cloud environments. A generic challenge is to overcome scalability, resiliency, and security issues faced by big data and data-intensive applications on distributed platforms by using transparent and optimized multi-cloud resource provisioning. One of the critical issues towards tackling the generic challenge mentioned is to design and develop the appropriate methods and tools for adequately describing placement preferences, constraints, and optimisation goals.

In this paper, we discuss the initial design of a Metadata Schema that enables data-aware multi-cloud computing. Its objective is to aid the data management, access control, and data-aware application placement for distributed and loosely-coupled multi-cloud applications. This objective is addressed by introducing terminology and vocabulary aspects of metadata that will be mainly used for extending a relevant domain specific language (DSL), i.e. the Cloud Application Modelling and Execution Language (CAMEL) [2], for describing big-data applications, their requirements and the available offerings. We note that this Metadata Schema provides a thesaurus structure that describes entities and their interrelations and could be used to extend any other related DSL for cloud application placement and orchestration specification (e.g. TOSCA). Precisely, the schema hierarchically structures, into a vocabulary, all the concepts (represented as lexical terms) that are relevant for describing cloud application requirements, big data aspects and characteristics and the offered cloud infrastructure capabilities for optimizing multi-clouds placement. This paper is structured as follows: Section II aggregates and discusses the most relevant vocabularies or
ontologies that mention concepts related to data-aware multi-cloud computing and notes which aspects of them are re-used or extended regarding our Metadata Schema. Section III, sets the motivation for this work and highlights its potential value. In Section IV, we present the main aspects and details of the Metadata Schema, focusing on the Application Placement and Big Data-aware sub-models (depicted in Figure 1), while we conclude this discussion in Section V.

II. RELATED VOCABULARIES

We focus on efforts that introduce relevant concepts or hierarchies, valuable for constructing a generic schema that may serve as a background vocabulary for driving the placement and reconfiguration of big data applications over multi-cloud resources.

We have detected a number of efforts that introduce definitions and taxonomies of relevant concepts. For example, Höfer and Karagiannis [3] analyze the available cloud computing services and identify some of their main characteristics, proposing a tree-structured taxonomy. Its purpose is to enable quick classifications and comparisons among different cloud computing services by starting from and elaborating on the IaaS/PaaS/SaaS classification. Youseff et al. [4] proposed a unified ontology of cloud computing, defining concepts distinguished in five layers, with three constituents to the cloud infrastructure layer. This work mainly discusses the inter-dependency and composability between the different layers in the cloud (i.e., SaaS, PaaS, IaaS, Data-Storage as a Service (DaaS), and Communication as a Service (CaaS)). A similar approach is also introduced in [5] that presents a search engine for cloud computing system and introduces the CO-1 and CO-2 ontologies to semantically define the relationship among cloud services. It is used for determining the similarity among cloud services using three types of reasoning (i.e. concept similarity reasoning, object property similarity reasoning, and datatype property similarity reasoning). Another relevant effort is the DICE project [6] that focuses on quality-driven development of big data applications. DICE offers a UML profile and tools that assist software designers reasoning about reliability, safety and efficiency of data-intensive applications. The DICE methodology covers quality assessment, architecture enhancement, continuous testing and agile delivery, relying on principles of the emerging DevOps paradigm [6]. Specifically, it has introduced a relevant metamodel for describing aspects of big data intensive applications. Several classes and properties are considered relevant (e.g. computation and storage nodes, computation and processing types etc.). Nevertheless, there is no direct support for expressing data location and big data related properties such as volume, transfer rates or even aspects of the operations that transfer data between cloud resources. Such concepts are covered in our proposed Metadata Schema.

In this work, we re-use and extend some of the definitions provided by the works mentioned above, for constructing a metadata schema that may cover all the necessary dimensions of big data placement on cloud resources.

Fig. 2. Application Placement Model’s UML Class Diagram (1/2)
III. MOTIVATION

In this section, we highlight the potential value of such a model for assisting and laying down the knowledge background based on which cloud application placement constraints and preferences can be expressed and exploited.

First, such a model will be used as a vocabulary that can extend the CAMEL language [2] (or any other similar DSL). CAMEL enables the specification of multiple aspects of multi-cloud applications, facilitating the optimised application placement and adaptation over multi-cloud infrastructures. This approach follows the model-driven engineering (MDE) paradigm that enables the modelling abstraction from the implementation details of heterogeneous cloud services. This also enables the development of appropriate mechanisms that allow both direct and programmatic manipulation of design and runtime models in order to facilitate the efficient matchmaking between cloud applications’ requirements and the available multi-cloud offerings. Among others, CAMEL introduces or builds on top of various sub-models in order to support the specification of cloud application requirements (e.g. Hardware, OS & Image and Provider Requirements, Location requirements, Security requirements, Scalability requirements/rules, Service Level Objectives (SLOs)). Based on these sub-models, an application developer should be able to describe its application requirements to drive the multi-cloud placement process. See for example in Tables 1 and 2 that capture a part of a cloud application specification in CAMEL that requires that a certain application should be placed only on VMs located in France, with a number of CPU cores between 8 and 32 and RAM between 4 and 8 GB). The main issue with any such DSL, like CAMEL or TOSCA, is that any concepts (e.g. CPU cores, RAM, location etc.) that are used in a requirement specification, are usually statically predefined. The process for extending it can be proven cumbersome (e.g. adding attributes like GPU regarding processing capabilities or anticipated Velocity with respect to data processing). This is a crucial issue especially for solutions that seek to find optimal cloud application placement in a big data-aware way, since currently it is not supported by any of the well-known related DSLs. For example, the optimal placement of a cloud application that performs batch processing over petabytes of data, is likely to be different from the optimal placement of an application that conducts real-time processing over data streams with a velocity of several gigabytes per second. Thus, big data related requirements need to be supported by an extended DSL (i.e. CAMEL) by formally incorporating a number of additional vocabulary terms that represent relevant big data related concepts (e.g. GPU offerings). Thus, the proposed Metadata Schema aspires to constitute the medium for modelling any concept necessary for expanding DSL’s expressivity with respect to cloud application requirements specifications and offerings descriptions, respectively.

Furthermore, this Metadata Schema is able to affect the variables that can be used as attributes by any constraint programming solver seeking to optimise a cloud application placement. This usually takes the form of a utility function minimisation or maximisation, and captures the optimisation goals of an application placement problem. The development of such a utility function is a domain specific problem which involves a ponderous trial and error process to be followed by an expert. Based on our approach, we provide a hierarchical structure of all the relevant classes that can be mapped into competing (shorter response times imply higher costs) parts of a utility function. Specifically, the utility function may involve four distinct parts (e.g. deployment/reconfiguration cost vs. RAM usage vs. batch processing throughput vs. response time), defined through proper mathematical expressions, which they should be combined by appending individual weights to each part. These weights define how much more important is one part of the utility function in comparison to another one (i.e. cost is 3 times more important than the response time). Based on the Metadata Schema, certain multi-criteria decision-making methods can be employed (e.g. Analytic Hierarchy Process) in order to properly calculate and append these weights in the final utility function.

TABLE I. REQUIREMENT MODEL EXAMPLE IN CAMEL

```plaintext
requirement model ScalarmRequirement {
  quantitative hardware CoreIntensive {
    core: 8..32
    ram: 4096..8192
  }
  os Ubuntu {os: 'Ubuntu' 64os}
  location requirement FranceReq {
    locations [ScalarmLocation.FR]
  }
}
```

TABLE II. LOCATION MODEL EXAMPLE IN CAMEL

```plaintext
location model ScalarmLocation {
  region EU {
    name: 'Europe'
  }
  country FR {
    name: 'France'
    parent regions [ScalarmLocation.EU]
  }
}
```

Summarizing the introduction of this Metadata Schema brings the following advantages:

- Formally and graphically declare, in a vocabulary, all the necessary terms that describe concepts to be used for comparing deployment alternatives
- Add data aspects in a DSL (i.e. CAMEL) without hard-coding any concepts
• Provide a unified way for stating the vocabulary terms’ importance captured in a utility function
• Support model extensibility by easily incorporating other vocabularies
• Support model reusability

IV. METADATA SCHEMA

A. Overview

We introduce our Metadata Schema that comprises the following model facets: Application Placement Model, Big Data Model, Context-Aware Security model. The several different facets of this Metadata Schema are analysed below, while a bird’s eye view of the schema can be found in Figure 1. For each of the facets their main top-level concepts are also depicted while explained in the sections below. For the representation of a comprehensible overview of our Metadata Schema, we used a detailed mind map for an easier walkthrough of the Schema’s main aspects which can be found here: http://melodic.cloud/assets/images/MELODIC_Model_vFinal.png

The first model facet of the Schema is the Application Placement Model, which provides a hierarchical structure over a number of concepts and properties that can be used either for describing cloud application placement requirements, constraints and preferences, or for describing the available cloud offerings, mainly at the IaaS and PaaS levels. This includes concepts that reveal processing (e.g. CPU), storage (e.g. Capacity), network (e.g. Bandwidth) as well as hypervisor characteristics or capabilities at IaaS level. At a PaaS level, we include concepts that characterise the available or required platform type, environment (e.g. OS) as well as the security controls (e.g. Data Sanitization) that it currently supports. In order to derive this set of classes and properties of this facet, the main vocabularies and/or ontologies that we reused and extended were the Saloon ontology [7] and CAMEL [2].

The second model facet of the Schema is the Big Data Model for multi-cloud management. This model provides a hierarchical structure over a number of concepts and properties that can be used for describing characteristics of data to be processed, that should be considered during application placement or cloud reconfiguration decisions. For deriving this model facet we mainly built on and extended several vocabularies like: DICE [6], the work in [8] and the CSA Big Data Taxonomy [9]. Nevertheless, to the best of our knowledge, this is the first systematic effort that tries to capture all the different data-related aspects that are important for data-intensive applications. Thus, this model facet reveals big data aspects (e.g. Volume, Velocity, Quality etc.), data management details (e.g. Acquisition, Data Storage, Processing etc.), data location and timestamp along with the relevant data domains (e.g. Finance, Social Networking etc.), that characterize the big data to be processed in multi-cloud environments.

The last model facet of the Schema is the Context-Aware Security model. This model aggregates a number of concepts and properties for describing and enforcing context-aware...
access control policies. This part corresponds to the Context-aware model presented in [10] and [11] extending it with a number of concepts that consider the infrastructural requirements and available offerings in multi-cloud application scenarios. We note that due to space limitations for each of these sub-models we discuss the details of only their main classes and properties while we depict using UML class diagrams some of their most noteworthy parts. Moreover, the third part of the Metadata Schema is not discussed since its details are available here: [10], [11].

### B. Application Placement Model

The Application Placement model refers to the following top-level concepts (see also Figure 2): IaaS, Provider, PaaS.

The IaaS top-level class encapsulates all the attributes related to cloud infrastructural resources that are required and offered for deploying multi-cloud applications. It reuses and extends the requirement model of CAMEL [2]. Its properties include the: refersToVM, refersToRack, Supports RequestsPer Second, hasVMCost, hasBare MetalCost, has Availability, hasCloudLocation, hasCloudProvider. For example, the hasVMCost property associates the IaaS class with a value expressed in floating-point format (float) denoting the usage cost of a certain virtualised resource. The main subclasses of IaaS are the following: Processing, Storage, Network, Cloud, and Hypervisor. The Processing class involves any infrastructural feature bound to the processing capability of virtualised resources. The Storage class describes the ephemeral or persistent storing capabilities that are required or offered by a certain IaaS resource. The Network class refers to the network related aspects that bound the operation of an offered or a requested IaaS resource. The Cloud class groups the characteristics of virtualised resources for easier reference and use. The Hypervisor class is used to express the characteristics of the used hypervisor software, firmware or hardware for creating and commissioning virtual machines.

The Provider class captures the characteristics of IaaS or PaaS providers and extends the hierarchy and concepts used in the Saloon ontology [7]. Its properties include the: offersIaaS, offersPaaS, hasProviderReputation, hasGreenFootprint. For example, the hasGreenFootprint property refers to a Boolean value that denotes whether or not the provider’s offerings have the minimum possible impact on the environment.

The PaaS class encapsulates all the attributes related to platform level resources that are required and offered for deploying multi-cloud applications (see also Figure 3). Its properties include the following: isOfferedByProvider, usesCloud, hasCloudLocation, hasCostFunction, hasAvailability, hasPricingType. The usesCloud object property associates the PaaS class with the Cloud class (of the Application Placement sub-model) for denoting with one reference the characteristics of the underlying IaaS level resources used for offering PaaS services. The main subclasses of PaaS (presented in Figure 3) are the following: Platform, Environment, Application Server, PaaS Configuration and Security Controls. The Platform subclass is used to register and select any one of the different available PaaS offerings depending on the scope of their offered services (e.g. OpenShift, CloudFoundry etc.). It includes the API subclass for referring to integrated platform offerings (e.g. Google Cloud AI, Amazon Machine Learning). The Environment subclass encapsulates all the aspects that identify the platform level cloud environment [3]. It involves the Framework, Image and Container subclasses. The Framework subclass is used for describing the offered or requested web framework capabilities for rapid development (e.g. PLAY, DJANGO). The Image subclass refers to the pre-installed cloud images available for initializing an IaaS resource and the Container subclass is used for describing any executable packages of software that include code, runtime, system tools, system libraries and settings for stand-alone execution. The Application Server subclass accumulates all the necessary application server information that might be requested (e.g. Apache Tomcat 9.0.x, Jetty 9.3.3 etc.). The PaaS Configuration subclass is used in order to register all the configuration details needed for using platform level cloud services.

The Security Controls is a subclass of the PaaS class that refers to all the possible security enforcement mechanisms that may be offered or required as a service for protecting the operation of hosted cloud applications. All its subclasses refer to specific security controls that have been classified based on the latest version of the Cloud Controls Matrix [12] introduced by the Cloud Security Alliance (e.g. CSA-IAM-02, CSA-IAM-09, CSA-IVS-01 etc.). For example, the CSA-IAM-02 subclass refers to all the relevant security controls offered as a PaaS service that belong to the CSA control domain entitled as: Identity & Access Management - Credential Lifecycle / Provision Management. Specifically, it involves subclasses and properties that can be used to denote the offering or the required functionalities for registering and updating the identity of entities permitted to request access to cloud resources and sensitive data (e.g. Authentication, Access Logging, and Credential Lifecycle Management). The CSA-IAM-09 subclass refers to the CSA control domain entitled as: Identity & Access Management - User Access Authorization [12]. It involves the Authorization subclass which is used to denote the offering or required functionalities for controlling the way access is permitted to cloud resources or persisted sensitive data (e.g. RBAC, Attribute-based access control). The CSA-IVS-01 subclass refers to all the relevant security controls offered as a PaaS service that belong to the CSA control domain entitled as: Infrastructure & Virtualization Security - Audit Logging / Intrusion Detection [12]. Its IDS subclass is used to provide information about the characteristics of intrusion detection systems (IDS) offered or required for monitoring the virtual resource for malicious activities or any policy violations. The CSA-IVS-06 subclass refers to all the relevant security controls offered as a PaaS service that belong to the CSA control domain entitled as: Infrastructure & Virtualization Security - Network Security [12]. Its IPS
subclass is used to provide information about the characteristics of intrusion prevention systems (IPS) for examining network traffic flows and patterns in order to detect and prevent vulnerability exploits. The CSA-IVS-13 subclass refers to all the relevant security controls offered as a PaaS service that belong to the CSA control domain entitled as: Infrastructure & Virtualization Security - Network Architecture [12]. Its DDoS Mitigation subclass is used to provide details on the distributed denial-of-service (DDoS) prevention capabilities offered or required for alleviating cyber-attacks that aim to constitute a cloud resource temporarily or indefinitely unavailable by flooding it with superfluous requests. The CSA-GRM-10 subclass refers to all the relevant security controls offered as a PaaS service that belong to the CSA control domain entitled as: Governance and Risk Management - Risk Assessments [12]. It involves the Security Risk Assessment subclass that lists the requested or offered tools for determining the security risks related to the virtualised resources use (e.g. Vulnerability Assessment). The CSA-EKM-02 refers to all the relevant security controls offered as a PaaS service that belong to the CSA control domain entitled as: Encryption & Key Management - Key Generation [12]. Its Key Management subclass is used to mention required or offered mechanisms necessary for creating, revoking and relaying cryptographic keys (to be used for encrypting/decrypting sensitive data) and also ensuring that these keys will not revealed to any unauthorized or malicious users [13]. The CSA-EKM-03 subclass refers to all the relevant security controls offered as a PaaS service that belong to the CSA control domain entitled as: Encryption & Key Management - Sensitive Data Protection [12]. Its Encryption subclass denotes the capability of offering encryption and decryption as a service from a certain virtualised resource. The CSA-DSI-07 subclass refers to all the relevant security controls offered as a PaaS service that belong to the CSA control domain entitled as: Data Security & Information Lifecycle Management - Secure Disposal [12]. Its Data Sanitization subclass denotes the capability of offering deliberate, permanent, and irreversible removal of data stored on a virtualised resource. Last, the CSA-BCR-02 subclass refers to all the relevant security controls offered as a PaaS service that belong to the CSA control domain entitled as: Business Continuity Management & Operational Resilience - Business Continuity Testing [12]. Its Security Testing subclass is used to mention any required or offered testing techniques and tools that verify the appropriate support of a certain virtualized resource for Confidentiality, Integrity, Authentication, Authorization, Availability and Non-repudiation.

C. Big Data Model

The Big Data Model refers to the following top-level concepts (see also Figures 4 and 5): Big Data Aspects, Data Location, Data Timestamp, Data Management and Data Domains. For each of these top-level core classes, we provide their main respective subclasses, their main properties along with their descriptions. The Big Data Aspects top-level class encapsulates all the attributes that can be used in order to describe the main characteristics of big data to be processed by applications hosted on multi-clouds (see also Figure 4). Based on such attributes, preferences on quantitative and qualitative

---

Fig. 4 Big Data Model’s UML Class Diagram (Big-Data Aspects part)
dimensions of virtualized resources can be expressed. Its properties include the following: hasDataOwner and hasDataLocation. For example, the hasDataLocation property associates the Big Data Aspects class with the Data Location class of the Big Data Model in order to denote where certain data artefacts may be found. The main subclasses of Big Data Aspects are the following: Data Density, Data Variety, Data Value and Data Quality. The Data Density subclass reveals details on big data observed or expected and includes the subclasses Volume and Velocity. The Volume subclass reveals details on the expected amount of data artefacts to be processed by a multi-cloud application. The Velocity subclass reveals details on the anticipated speed of data to be processed along with the types of feeds that may be encountered (e.g. Real-time, On demand, Time-series, Continuous). The Data Variety class refers to the different types of data that should be processed by a multi-cloud application, stating an increased diversity of data that should be stored, processed or combined. Its subclasses Format and Type refer to the structural variety that big data may involve which is expressed using certain schemes and models (e.g. binary large object (BLOB), JSON, XML etc.) or to the media variety that big data may involve with respect to the medium in which data get delivered (e.g. audio, image, video, text), respectively. The Data Value class refers to big data aspects that reveal the business importance of data which is bound to the potential of improving a business entity’s decision making capabilities, corresponding mainly to its uniqueness, usability and comprehensiveness. The Data Quality class encapsulates another group of important big data concepts that reveal aspects about how accessible, secure, compact, volatile or uncertain the data is. This involves several other subclasses like Volatility and Compression. Volatility can be used for referring to the level of convenience offered when attempting to access certain data artefacts. For example, accessing encrypted data might deteriorate the accessibility with the benefit of securing sensitive data. The Compression subclass refers to if and how data has been encoded in order to use fewer bits than its original representation (i.e. as it was captured from the relevant data sources). This can be succeeded by identifying and eliminating statistical redundancy (lossless compression) or by removing unnecessary or less important information (lossy compression).

Fig. 5 Big Data Model’s UML Class Diagram (Data Management part)
The **Data Location** top-level class encapsulates all the concepts that can be used for describing the origin of data or the current or required physical/network location where the data can be stored or processed by a multi-cloud enabled application. Its properties include the: `isStorageLocation`, `sameAs`, `notSameAs`, `hasSparsity`, `hasPreferredLocation`, `hasAllowedLocation`, `hasUnacceptableLocation`, `hasPhysicalLocation`, `hasNetworkLocation`, `hasCloudLocation`. For example, `hasSparsity` property associates the Data Location class to a string that denotes how distributed (e.g. Low, Medium, High) are the data sources or data locations exploited for producing a dataset. The main subclass of Data Location is the Origin class which involves all the relevant concepts for defining the source location of the data artefacts to be processed distinguishing between batch and stream origin.

The **Data Timestamp** top-level class includes all the necessary concepts for describing the temporal characteristics of data artefacts to be processed by a multi-cloud application. Its properties include the: `hasTimePoint`, `hasTimeInterval` for referring to the precise point in time or period (bounded by two time points) respectively, at which data was or should be processed. The main subclasses of Data Timestamp are the following: `Creation Date`, `Deletion Date`, `Acquisition Date`, `Processing Date`, and `Transfer Date` for defining when a certain data element was created, deleted, acquired, processed or altered respectively.

The **Data Management** top-level class encapsulates all the relevant concepts that can be used in order to describe major technological choices with respect to how big data is acquired, stored, processed, transferred or replicated for redundancy reasons (see also Figure 5). Its properties include the: `hasDataTimestamp` and `hasAgent`. For example, the first is an object property that associates the Data Management class with the Data Timestamp class of the Big Data model in order to express the time when certain data artefacts where acquired. The main subclasses of Data Management are the following: `Acquisition`, `Data Storage`, `Processing`, `Transfer` and `Redundancy`. The Acquisition subclass is used in order to describe the required or offered types of big data acquisition in the frame of a multi-cloud application devised to process it. In addition, through its **Pull-based** and **Push-based** subclasses captures concepts related to the pull-based paradigm for acquiring data, where there is a request for triggering the transmission of data and its synchronous interception [14] or concepts related to the push-based paradigm for acquiring data asynchronously, where the request for a given transaction is initiated by the publisher [14]. The Data Storage subclass encapsulates all the concepts that can be used for characterising the way that input or output data should be stored. The hierarchy involved updates the storage infrastructure taxonomy that [9] presented and involves the following subclasses: `Relational`, `Non-Relational`, and **NewSQL**. For example, the Non-Relational subclass refers to databases (also called NoSQL) used for persisting data that are not modelled using tabular relations and present certain advantages over the relational databases, especially for big data since they offer design simplicity and more efficient horizontal scaling. This involves **Key-Value Store** [15], **Dynamo-Inspired** [16], **In-Memory** (e.g. Memcached, Aerospike), **Document Oriented Store** (e.g. MongoDB, NosDB) and **Big Table-inspired Store** [17] all captured as subclasses of Non-Relational. The NewSQL subclass refers to a type of parallel database management systems that provides the same scalable performance of non-relational systems while still maintaining the same level of transactional support (i.e. support the properties of Atomicity, Consistency, Isolation, and Durability – ACID) as the traditional relational databases [9]. This involves the New-In-Memory (e.g. VoltDB, H-Store), **GraphDB** [18] and **Multimodel DB** [19] captured as subclasses of NewSQL. The Processing subclass encapsulates all the concepts that can be used for describing and classifying the various types of big data processing that can be conducted by a multi-cloud application. The hierarchy introduced updates both the DICE model for big data intensive application [6] and the computer infrastructure taxonomy presented by CSA Big Data Taxonomy [9]. In involves the following properties: `hasRawMetric`, `hasComplexMetric`, `isLongLived`, `hasDataProcessingCost`, `hasProcessingCost`, `hasAppProcessingLocation`, `hasConstraints`, `hasPriority`, `forProductionUsage`, `isRealTime`, `isNearRealTime`. For example, the `isNearRealTime` property associates the Processing class with a boolean value that denotes whether or not the processing takes place almost at the same time as input data is produced. This property implies a time delay introduced due to network lag, between the data source and the processing location. The Processing class involves the following subclasses: `Stream Processing`, `Batch Processing`, `Hybrid Processing`, `DistributedML`, `Computational Complexity`, and **Methodology**. For example, the Hybrid Processing is a subclass that involves processing techniques that can be classified in the space between stream and batch processing. These are also called micro-batching techniques and treat streams as a sequence of small batches or chunks of data that are processed in near real-time (e.g. Apache Spark). The Transfer subclass of Data Management refers to any concept that can be used for describing aspects related to communicating data artefacts between their data sources and the processing or storing locations. Last, the Redundancy subclass encapsulates any approach used for persisting the same data artefacts in several separate places, either in a single database, or in remote databases for detecting and reconstructing lost or damaged data [20].

The **Data Domains** top-level class encapsulates all the relevant concepts that characterize data based on the industries that produce it or need to extract information from it [9]. Specifically, we reuse and extend the big data taxonomy introduced by the Cloud Security Alliance [9] and includes the following subclasses: `Sensor Data`, `Network Data`. 
Security, Social Networking, Finance, Retail, Large Scale Science, Visual Media and Audio Media. For example, the Large Scale Science subclass addresses all the different data artefacts that may be used or produced in large scale science experiments and activities which are usually exploited through batch processing.

V. CONCLUSIONS

In this work, we introduced the details of a Metadata Schema for data-aware multi-cloud computing. It corresponds to a vocabulary based on which dedicated software components will be able to interpret requirements, constraints and offerings’ characteristics in order to properly manage big data, optimise the placement of their processing jobs and control all accessing requests in multi-cloud environments. The Schema comprises the Application Placement, Big Data and Context-Aware Security models that include classes and properties for defining where a certain big data application should be placed, what are the unique characteristics of the data artefacts that it needs to process, and what are the contextual aspects that may be used for bounding the access to the sensitive data.

The next steps of this work include the development of two design-time tools for managing and exploiting the presented Metadata Schema. The first tool will constitute the Metadata Schema editor, which will be used to create, update and maintain all the classes, properties and instances of the presented model. We note that this tool, during any modifications of the metadata schema, will consider backwards compatibility with applications based on earlier versions of the model by revealing any issues to be resolved. The second tool will be used during application modelling, in order to help application developers to calculate and fine-tune the weights (priorities) of the polynomial utility functions (that use this model as background knowledge), as mentioned in section III. This information will be valuable for formulating the utility function to be used by constraint programming solver mechanisms, in order to generate real deployment plans meeting the constraints and requirements of application developers and DevOps in Multicloud usage scenarios.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No. 731664. The authors would like to thank the partners of the MELODIC project (http://www.melodic.cloud/) for their valuable advices and comments.

REFERENCES


