Future Generation Computer Systems 88 (2018) 373–384

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

An ontology for heterogeneous resources management interoperability and HPC in the cloud

Gabriel G. Castañé *, Huanhuan Xiong, Dapeng Dong, John P. Morrison

Department of Computer Science, University College Cork, Ireland

HIGHLIGHTS

- An ontology for interoperability in heterogeneous cloud infrastructures isproposed.
- Enable the adoption of heterogeneous physical resources in self managed clouds-Support for HPC-in-Cloud, hardware accelerators, resource abstraction methods.
- A proposed architecture to explot the semantic and sintactic benefits.
- Included into the CloudLightning project for large scale Cloud Computing environments.

ARTICLE INFO

Article history:
Received 31 December 2017
Received in revised form 19 April 2018
Accepted 30 May 2018
Available online 19 June 2018

Keywords:
Cloud interoperability
HPC in cloud
Resource management
Ontology
Self-management clouds

ABSTRACT

The ever-increasing number of customers that have been using cloud computing environments is driving heterogeneity in the cloud infrastructures. The incorporation of heterogeneous resources to traditional homogeneous infrastructures is supported by specific resource managers cohabiting with traditional resource managers. This blend of resource managers raises interoperability issues in the Cloud management domain as customer services are exposed to disjoint mechanisms and incompatibilities between APIs and interfaces. In addition, deploying and configuring HPC workloads in such environments makes porting HPC applications, from traditional cluster environments to the Cloud, complex and ineffectual.

Many efforts have been taken to create solutions and standards for ameliorating interoperability issues in inter-cloud and multi-cloud environments and parallels exist between these efforts and the current drive for the adoption of heterogeneity in the Cloud. The work described in this paper attempts to exploit these parallels; managing interoperability issues in Cloud from a unified perspective. In this paper the mOSAIC ontology, pillar of the IEEE 2302 — Standard for Intercloud Interoperability and Federation, is extended towards creating the CloudLightning Ontology (CL-Ontology), in which the incorporation of heterogeneous resources and HPC environments in the Cloud are considered. To support the CL-Ontology, a generic architecture is presented as a driver to manage heterogeneity in the Cloud and, as a use case example of the proposed architecture, the internal architecture of the CloudLightning system is redesigned and presented to show the feasibility of incorporating a semantic engine to alleviate interoperability issues to facilitate the incorporation of HPC in Cloud.

1. Introduction

Cloud computing environments offer customers a wide diversity of services through loosely coupled instances, and storage systems, guaranteeing certain levels of services. Features such as availability on demand, large capacity, elasticity, and service-level performance have been attracting end-users to migrate (to Cloudify) their applications from traditional cluster environments. It is estimated that by the year 2019 more than 85% of workloads will be processed by Cloud environments [1].

The Cloud market is expanding [2,3] and this growth is attracting specific users demanding specific services that are driving the traditional homogeneous Cloud infrastructure to become a heterogeneous ecosystem. In particular, supporting High Performance Computing (HPC) services in the Cloud requires an evolution of the traditional homogeneous cloud [4]. HPC as-a-service (HPCaaS) is leading the availability of services that, from a cloud management perspective, are challenging to support. Cloudifying HPC applications while maintaining performance, while desirable,

* Corresponding author.
E-mail addresses: gabriel.gonzalezcastane@ucc.ie (G.G. Castañé), h.xiong@ucc.ie (H. Xiong), d.dong@ucc.ie (D. Dong), j.morrison@ucc.ie (J.P. Morrison).

https://doi.org/10.1016/j.future.2018.05.086
0167-739X/

presents significant challenges. Virtualization management, virtual machine monitoring, communications, and processing overheads, detract from the bare metal performance of applications [5]. Currently, only a few such have been Cloudify-ed, since they have been optimized over decades for specific hardware architectures. Early forays into this process can be seen in the attempts to create models for determining the effectiveness of cloud environments in supporting such HPC workloads [6–8].

Cloud-based HPC solutions are offered by several commercial vendors. The include dedicated servers and containers used to access hardware accelerators in cloud computing infrastructures. For example, Amazon EC2 provides instances based on Xeon servers with GPU accelerators and FPGA instances as dedicated servers [9,10]. To support diverse resource abstraction methods, these providers offer multiple frameworks, as required. Foremost among these are OpenStack Nova [11] and OpenNebula [12] to manage virtualized environments; Kubernetes [13], Mesos [14], and Docker Swarm [15] to manage container environments; OpenStack Magnum [16] to manage containers within virtualized environments; OpenStack Ironic [17] to bare-metal servers; and finally, traditional cluster management frameworks for HPC/HTC such as SLURM [18] and ROCKS [19]. However, in spite of the maturity of these frameworks, none supports all resource abstraction mechanisms. Thus, in a heterogeneous environment multiple interacting frameworks are required to adequately support the diversity of resources in that environment. This complexity is exacerbated in the HPC domain where services running on heterogeneous resources may co-operate to deliver the HPC application. Combining resources in this manner is currently state-of-the-art and requires expert low-level configuration. Customers need to be able to configure, deploy, and link services associated with diverse instances and address the interoperability issues associated with using different resource abstractions under the control of multiple resource managers. The deployments of services interacting across a number of management domains are tailored to specific resources managers and are currently supported by specific-to-vendor interfaces potentially using different, and often incompatible, APIs.

Fig. 1 shows an idealized extensible heterogeneous cloud architecture containing multiple resources. This architecture would require sophisticated mechanisms to enable services hosted on different heterogeneous resources to interact within the workflow.

Physical resources (labeled from R_1 to R_5 in the figure) represent different sets of heterogeneous hardware available in the cloud infrastructure. For example, commodity hardware, servers with hardware accelerators, and servers with enhanced capabilities such as low latency networks. In the figure, four coexisting local resource managers are deployed in the system, partitioning the Cloud in support of multiple hardware types and resource abstraction methods: *LocalRM*1 uses a *VirtualizationLayer* for providing virtual machines on $R1$ and $R5$ resources, e.g., OpenStack Nova, Eucalyptus, or OpenNebula; $LocalRM_2$ and $LocalRM_4$ uses a *ContainerizationLayer* to provide containers on R_3, R_4, and part of the R_5 physical resources, e.g., Marathon, Kubernetes, or DockerSwarm; and finally $LocalRM_3$ accesses R_5 hardware resources in a dedicated server fashion, e.g. ROCKS, SLURM, or specific proprietary resource managers. On the left hand side, Fig. 1 shows how the incorporation of a new resource manager into the resources fabric will increase the number of resource management partitions within the management domain of the Cloud. For example, *Amazon EC2 F1* and *G2* instance types [20] offer virtual machines instances based on FPGA and GPU hardware that have to be manually configured and linked to reflect the composition of services within an application using this hardware.

A consequence of the work proposed in this paper is the elimination of this manual configuration step. The approach take is to define and use an ontology as a mechanism to achieve semantic interoperability [21] in heterogeneous systems.

The main goals of this work are: to create an ontology, known as the CloudLightning Ontology (CL-Ontology) that supports heterogeneous and high performance resources in the Cloud; to ameliorate interoperability issues between existing resource managers and resource abstractions; to maintain compatibility with previous standards for interoperability.

The CL-Ontology extends the IEEE-2302 (SIIF) — Standard for Intercloud Interoperability and Federation [22], and the IEEE-2301, Guide for Cloud Portability and Interoperability Profiles (CPIP) [23]. This extensions incorporate support for resource management, specific hardware accelerators, and different resource abstraction methods, such as virtual machines and containers into the Cloud; matching service requests to specific heterogeneous infrastructures, and enabling intelligent resource discovery. To the best of our knowledge this is the first attempt to explicitly address an Ontology supporting HPC in Cloud.

In addition to the CL-Ontology, in an effort to demonstrate its utility, this paper presents a conceptual Service Oriented Architecture (SOA) for autonomic resource management. In the proposed architecture, the CL-Ontology is used as part of a semantic engine to dynamically incorporate resources into the cloud resource fabric, and to support decisions making for targeting service requests to appropriate resources. Finally, the architectural design of the CloudLightning [24,25] (EU H2020-ICT programme under grant #643946) system is presented as use case into which the proposed semantic engine is incorporated, and where the management of heterogeneous resources at scale is undertaken.

2. Related work

Many efforts have been made towards addressing interoperability issues in the Cloud by creating common interfaces and APIs that enhance the compatibility between deployment models and public vendors. The Open Cloud Manifesto [26] is an initiative by industry for supporting open standards in cloud computing. The main targets are grouped into five categories: security, data application interoperability, data application portability, monitoring and portability between clouds. However, the Manifesto does not incorporate current cloud technologies nor support for hardware accelerators within any of the above mentioned categories. Similarly, the Universal Cloud Interface (UCI) [27,28] was proposed to solve inter-cloud interoperability avoiding lock-in issues with proprietary solutions by unifying the representation of all cloud resources in a common interface. Its evolution has been very limited and UCI does not incorporate concepts emerging from cloud technologies.

The Guide for Cloud Portability and Interoperability Profiles (CPIP) [23] assist cloud computing vendors and users in developing, building, and using standards/based cloud computing products and services. For each element, multiple options are proposed regarding interfaces, file formats and operational conventions. However, these are grouped in the "Standard profiles", as drivers for interoperability from a user perspective, does not support concepts on specialized hardware nor emerging resource abstraction methods in Cloud.

Another initiative, specifically targeting the IaaS service model, is the Open Cloud Computing Interface (OCCI) [29]. OCCI defines an interface to support the creation of hybrid cloud environments independently of cloud providers and frameworks. It specifies, in UML, real-world resources and their links, expressed in a similar manner to an OWL [30] ontology definition. However, OCCI targets a user-view perspective of the Cloud, in which resource managers manage traditional homogeneous resources, and hence, do not address interoperability issues caused when multiple services coexist and use diverse abstraction methods and specialized hardware.

By using ontologies, it is possible to generate intelligent decision support mechanisms for addressing interoperability issues in

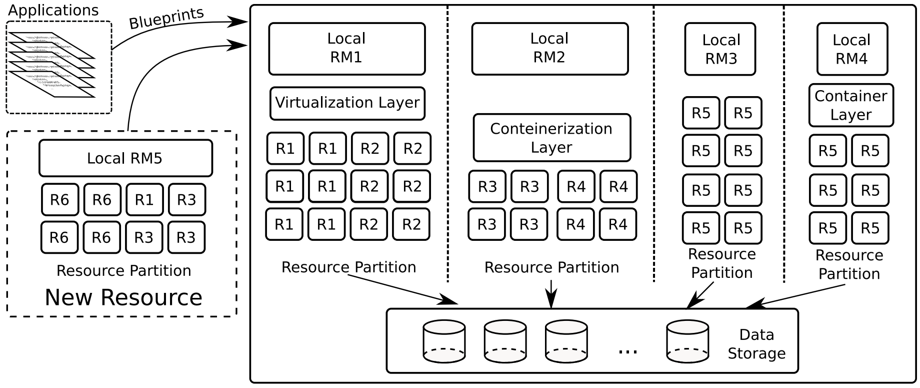


Fig. 1. An idealized cloud infrastructure containing heterogeneous resources.

services-based architectures. Accordingly, ontologies have already been used in the HPC arena. Zhao et al. [31] created an HPC ontology exposing the lack of a flexible and open infrastructure for the HPC community to effectively share, accumulate, and reuse knowledge. Although the authors emphasize the design of an HPC ontology, their work is in an early stage. Another ontology in HPC computing is proposed by Tenschert A. in [32], describing a matching method for decomposing HPC applications into distributed environments by using an HPC ontology. However, the main goal of the authors is to decompose applications between compute and data processes for HPC environments, rather than cloud environments.

Cloud ontologies have been widely used in recent years. Imam F. describes in [21] well known applications of Ontologies to Cloud Computing. These are grouped into security [33], resource management and service discovery [34,35], and interoperability categories. The most remarkable work in the interoperability category is the mOSAIC cloud ontology [36], showing a detailed and simple description of cloud computing resources. This ontology has been developed in the mOSAIC project within the European framework FP7 and is targeted to promote transparency in multiple clouds accesses. For its development both, the taxonomy proposed by the National Institute of Standards and Technology (NIST), and the IBM Cloud computing Reference architecture were extended, for the later incorporation to the Guide for Cloud Portability and Interoperability Profiles (CPIP) IEEE-2301 [23]. Thus, mOSAIC ontology is currently part of the standard SIIF-IEEE P2302 [22], addressing inter-cloud interoperability from the user interaction perspective, providing detailed entities for SLAs and Interfaces/APIs. Moreover, the ISO–IEC JCT 1, Standard in Cloud computing and Distributed Platforms [37] focuses on the standardization for interoperable distributed application platforms and web services, service oriented architectures, and cloud computing. Specifically, the ISO/IEC 18384 [38] uses ontologies to describe actors, services consumers, services providers, and legal aspects of the Cloud. However, recent advances in technology, hardware accelerators, resource managers, and resource abstraction methods required to avoid interoperability issues arisen from the incorporation of heterogeneous resources into Cloud, are not part of these standards.

Other initiatives in the EU frameworks FP7 and H2020 targeting interoperability and lock-in vendor issues in cloud computing environments have been explored. Foremost among these are: Cloud4-SOA [39], a platform that performs seamless adaptive multi-cloud management, semantically interconnecting heterogeneous PaaS offerings across different cloud providers with the same technology and supported by Apache Brooklyn blueprints and TOSCA extensions; PaaSport [40] where Cloud PaaS technologies are combined with lightweight semantics and in which application models and SLAs follow a Descriptions and Situations Pattern Technique, used to deliver a thin and non-intrusive Cloud-broker in the form of a Cloud Marketplace; Paassage [41] that constructs a deployment mechanism for applications across public and private clouds constrained by a set of rules described in the CAMEL [42] modeling language; ModaClouds [43] that follows a Model-Driven-Development for Clouds and Multi-Clouds, performing semi-automatic translations into code to enable deployments across public and hybrid cloud vendor platforms; and finally, the Mikelangelo project [44], that provides support for HPC in Cloud environments by implementing a bespoke virtual machine manager to allow the management in of HPC and Cloud resources. Although these projects succeed in exploring different alternatives for interconnecting heterogeneous vendor service offerings, enhancing extant interfaces, or providing decision support systems, either they neglect the incorporation of HPC into the Cloud and the subsequent need for having multiple resource management domains, or they do not support a wider heterogeneous environment, being constrained to only resource management, hardware accelerators, or resource abstractions in the same Cloud.

3. CL-Ontology

The CL-Ontology has been developed to address interoperability in a heterogeneous cloud resource fabric. The basis of the CL-Ontology is the mOSAIC ontology and, although the latter focuses on intercloud/multicloud environments from a SaaS and PaaS perspective, it provides a strong foundation enabling extensions to capture the emerging heterogeneous cloud.

Incorporating HPC environments in the cloud is made possible by extending the interoperability concept considered by the mOSAIC ontology with additional semantics to support: inter/intra-cloud environments, and heterogeneity in the usage of resources, resource managers, and co-processors.

The creation of the CL-Ontology is a methodical procedure for which each extension requires seeking and analyzing classes, concepts, attributes, and characteristic from the mOSAIC ontology to ensure consistency. Moreover, when an element has been added as a new class or subclass, it is necessary to analyze the relationships created between that extension and all of the relevant parts of the base ontolgy. That analysis unearth issues with relationships, constraints, and restrictions that were unforeseen and unexpected. More detailed information of this process can be found at [45].

The CL-Ontology has been developed in the OWL language by using Protégé 5.2.[1] as ontology editor, knowledge management, and visualization system, and used also to generate all Figures included in this section. Finally, for clarity and to distinguish between the original mOSAIC Ontology components (shown as *emphasized text*), the CL-Ontology extensions are presented highlighted in this section in **bold**.

1 https://protege.stanford.edu/.

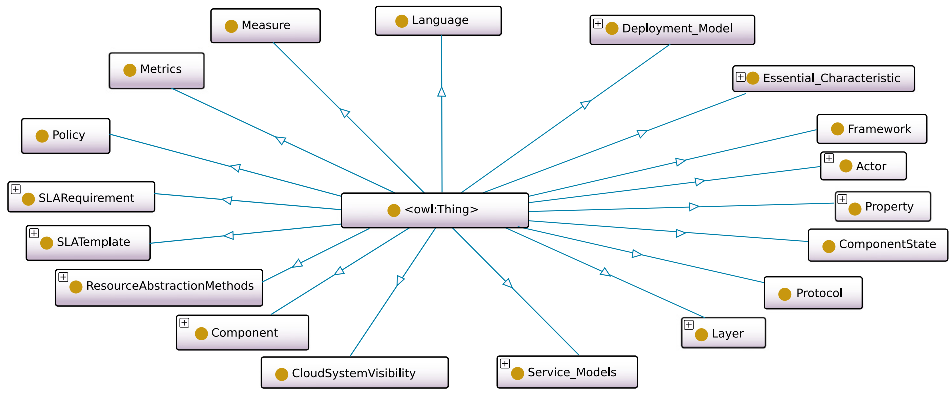


Fig. 2. Top level classes of CL-Ontology.

3.1. The TopLevel class ($\langle owl : Thing \rangle$)

Fig. 2 depicts the top level of the CL-Ontology. It consists of: a *Language* class identifying languages used for API implementations; a *DeploymentModel* class, which describes concepts of public, private or hybrid clouds; an *EssentialCharacteristics* class, which captures features of Cloud systems that users can exploit; a *Framework* class supporting programming frameworks; an *Actor* class specifying modes of interaction with the Cloud system; a *Property* class describing the characteristics of the elements of the Cloud; a *ComponentState* class identifying concepts for establishing the states of diverse Cloud components and resources; a *Protocol* class describing communication protocols used in Cloud environments; a *Layers* class containing individuals describing firmware, hardware, and software associated to the cloud infrastructure; a *ServiceModels* class supporting diversity of services offered in the Cloud; a *CloudSystemVisibility* class defining the deployment models of the cloud computing system; a *Component* describing resources, services, and elements part of the Cloud; a **ResourceAbstractionMethods** class, previously the *Technology* class in the mOSAIC ontology to describe virtualization, and that is extended to reflect current evolution of cloud ecosystems, supporting also containerization as technique to partition resources and used by customers to deploy services; a *SLATemplate* class describing the SLA specification templates; a *SLARequirement* class defining the SLA requirements for resources for negotiating with Cloud authorities; a *Policy* class grouping policies when SLAs are negotiated; and finally, *Metrics* and *Measures* classes used for evaluating properties of physical and abstracted resources.

The **Provider** class usage, which is a subclass of *Actor*, has been updated from the *offersVirtualMachine* to **offersResourceAbstraction**, having as subproperties: **offerDedicatedResource**, *offerVirtualMachine*, and **offerDockerContainer**.

3.2. The Property class

The *Property* class has *FunctionalProperty* and *NonFunctionalProperty* subclasses. The *NonFunctionalProperties* class describes essential characteristics or attributes of cloud components, in which following subclasses remain unchanged from the mOSAIC ontology: *Autonomy*, *Availability*, *Performance*, *QoS*, *Reliability*, *Scalability* and *Security*. The *CommunicationNonFunctionalProperty*, *ComputingNonFunctional*, and *PropertyDataNonFunctionalProperty* classes, subclasses of *NonFunctionalProperties*, describe physical features of processors, communications and storage of resources. However, the lack of detail for describing hardware accelerators has driven the CL-Ontology to incorporate a **CoprocessorNonFunctional** class for supporting co-processors and specialized hardware, and hence recognizing heterogeneity in the cloud resource fabric. Fig. 3 depicts an exploded view of the **CoprocessorNonFunctional** class.

3.2.1. The non functional properties class

GPUs, Many Integrated Cores (MICs) architectures, and FPGAs co-processors, have been chosen as co-processors included into the CL-Ontology as they represent the most part of the market for HPC in Cloud, however, the CL-Ontology is not limited to these and can be extended with different existing and future co-processors using the subclasses of the **CoprocessorNonFunctional** class, in which a broad set of specific hardware accelerators can be uniquely characterized in one of these categories, independently of the manufacturer and model. It captures heterogeneous hardware resources being added to the cloud resource fabric in a generalized fashion (Section 4.2), enabling a resources discovery process for finding a suitable host for each heterogeneous resource requested by cloud customers (Section 4.1).

To generalize these hardware types, specialized hardware datasheets have been investigated and key (model, architecture and manufacture independent) properties associated to each accelerator have been extracted. Thus, GPUs are characterized by the subclasses of the **GPUNonFunctionalProperties** class: the **GPUComputingProperty** subclass identifies properties of GPU processors such as the clock frequency, Flops at single and double precision, and numbers of streams; and the **GPUMemoryProperty** subclass is used to describe the memory system of GPUs, in which memory bandwidth and memory size are included as subclasses.

The subclasses of the **MICNonFunctionalProperty** class are used to characterize MIC accelerators. The **MICSerieProperty** subclass includes **XeonPhi3110**, **XeonPhi5120**, **XeonPhi7110**, **XeonPhi7120** individuals; the **MICComputing** subclass describes compute capabilities of MIC hardware, such as architecture, number of cores, and frequency of the cores; and the **MICMemory** subclass allows to describe memory cache size, memory type, memory size, and memory controller features into its subclasses.

The subclasses of the **FPGANonFunctionalProperties** class are used to categorize FPGAs: the **FPGAArchitecture** subclass with **Aria**, **Cyclone**, **MAX**, **Stratix**, and **Virtex** individuals; the **FPGAGeneration** subclass distinguishes between possible incompatibilities arising between FPGAs generations; the **FPGARoutingArchitecture** subclass with **HFPGA**, **HRSA**, and **APEX** routing schemes describe different configurations of FPGAs; the **FPGAClock** subclass identifies properties related to the maximum frequency and number of clocks described by its nested subclasses; the **FPGAComputingBlock** subclass contains the **FPGAComputingBlockDSP** subclass to describe DSPs architecture and the number of DSPs integrated in the FPGA, and the **FPGAComputingLogicBlocks** subclass to identify connection type, number of lookup tables, and number of flip-flops available in the computing block properties class; I/O specifications of the accelerator are described in the **FPGAIOBlock** subclass, and this is composed of the **FPGANumberIOBlocks**, and **FPGARoutingChannels** classes, describing the

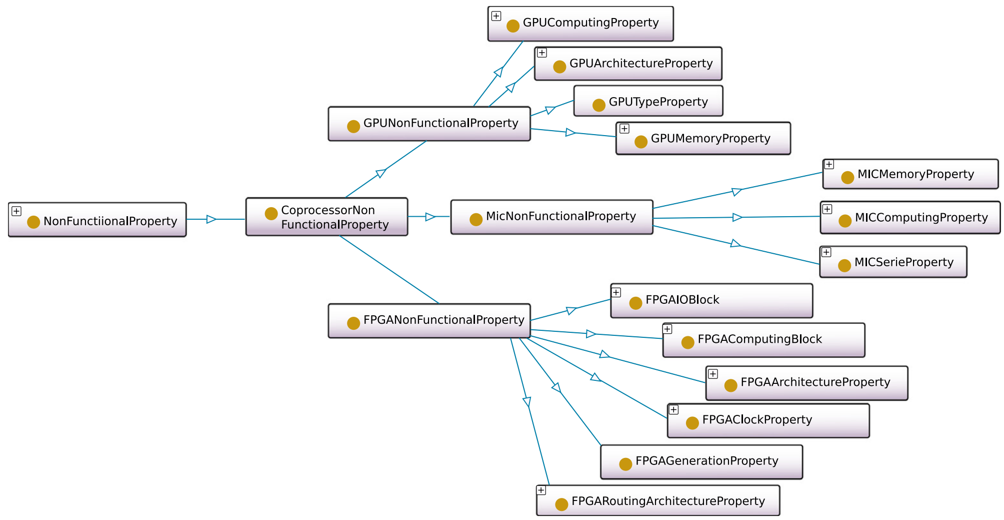


Fig. 3. Non-functional properties.

number of horizontal and vertical routing channels; finally, the **FPGAIOBlockConnectionBox** subclass identifies communication specifications of FPGAs, such as connection type, with bidirectional and unidirectional elements as individuals, number of connections per box, and number of connection boxes.

3.2.2. The Functional Properties class

The *FunctionalProperties* class has been extended from the mOSAIC ontology to include multiple resource abstraction methods. Thus, this class consists of: *Accounting*, *BackupAndRecovery*, *Consistency*, *Encryption*, *Identification*, *Replication*, *Monitoring*, *Management*, and **ResourceAbstractionDescription** subclasses.

Fig. 4 depicts the subclasses of *FunctionalProperties*. The *Monitoring* subclass was extended with physical and resource abstractions monitoring processes. Previously, in the mOSAIC ontology, this subclass focused only on resources offered to the customers. Two subclasses have been added: the **AbstractedResourceMonitoring** subclass, identifying metrics associated with heterogeneous resource abstraction methods offered by vendors, and the **PhysicalResourceMonitoring** subclass, to describe metrics of physical resources. The metrics associated to the **AbstractedResourceMonitoring** subclass are composed of: *BandwidthUsage* in which *DownloadUsage* and *UploadUsage* are subclasses; *StorageUsage*; *MemoryUsage* in with **GeneralMemoryUsage** and **SpecificHardwareMemoryUsage** are subclasses to provide metrics on the memory of heterogeneous systems; and, in the same manner, the **ProcessingUsage** class provides metrics on the *CPUProcessingUsage* subclass and the **SpecificHardwareProcessingUsage** subclass. In addition to the metrics described in the subclasses for th **AbstractedResourceMonitoring** class, the **PowerConsumptionUsage** class was included as subclass into the **PhysicalResourceMonitoring** class to describe the monitoring process of the energy consumed by physical resources.

The *VMDescription* class has been replaced with a **ResourceAbstractionDescription** class, to capture properties associated with containers and dedicated servers, currently offered as resource abstraction methods by cloud vendors.

The last *FunctionalProperties* subclass modified to incorporate heterogeneity in cloud environments is the *Management* class. The mOSAIC ontology associated subclasses are: *ImageManagement*, *NetworkManagement*, *StorageManagment*. In addition to these, the **InfrastructureMonitoringManagement** subclass has been incorporated into the CL-Ontology to specify management frameworks for collecting metrics from heterogeneous hardware, including as individuals: **OpenStackCeilometer**, *AmazonCloudWatch*, *AppDynamics*, and **SNAP**. The **ServicesManagement** subclass has added to the *Management* class for supporting service oriented architectures, in which services configurations and properties are specified by users but managed by vendors. A group of individuals has been added to the *Management* class, currently used in Public and Private Clouds, such as **OpenStack**, *AWSElasticBeans*, **Mesos**, **Slurm**, *AWSCloudformation*, and **Rocks** individuals.

3.3. The Layer class

The *Layer* class, Fig. 5, is defined based on OCCI, NIST, and IBM Cloud Computing Reference Architecture to identify firmware, hardware, and software of the Cloud. New subclasses have been incorporated into the *Application*, *FirmwareAndSoftware*, *OperationalLayer*, *ServiceLayer*, and *SoftwareKernel* classes. The *Application* class describing services defined by users by using the *UserComponents* subclass, does not support current service oriented architectures where users define their applications using a Service Description Language (SDL) in which components and topology are defined as part of the specification of services. To support users applications and requests based on SOAs, the **SequencingComponent** subclass has been added to the *Application* class, describing communications between user components. This supports **OpenStack Solum** [46] and **Apache Brooklyn** [47] as individuals, enabling service oriented architectures in PaaS services; and the **OpenStack Heat** [48] individual representing service oriented architectures orchestration in IaaS services.

The *FirmwareAndHardware* class describes the firmware and hardware of cloud infrastructures. The *Hardware* subclass, the classes representing physical resources *Cache*, *CPU* and *Memory*; have been nested into the **CommodityHardware** class, where traditional cloud hardware is included. In addition, **Network** has been incorporated as a subclass of **CommodityHardware** to describe available network connections between physical hardware with **Ethernet**, **FastEthernet**, **GigabitEthernet** and **10GbEthernet** subclasses to incorporate IEEE-802.3 — 10 Mbps, IEEE-802.3u — 100 Mbps, IEEE-802.3z — 1000 Mbps, and IEEE-802.3ae — 10 Gbps protocols. Moreover, to decouple specialized hardware from commodity hardware, the **SpecificHardware** subclass with **Coprocessor** and **SpecializedNetwork** children have been added to the *Hardware* class. The **Coprocessor** class contains **FPGA**, **GPU** and **DFE** subclasses to define specific hardware accelerators that can be attached to the Cloud, while the **SpecializedNetwork** class is

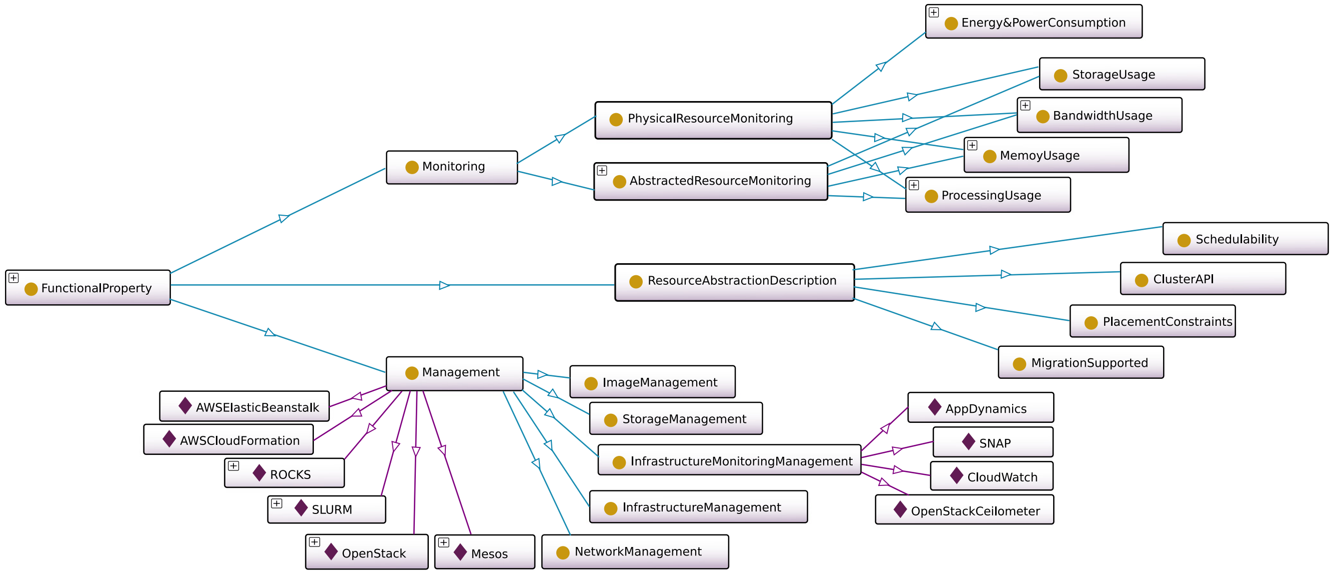


Fig. 4. Functional properties of CL-Ontology.

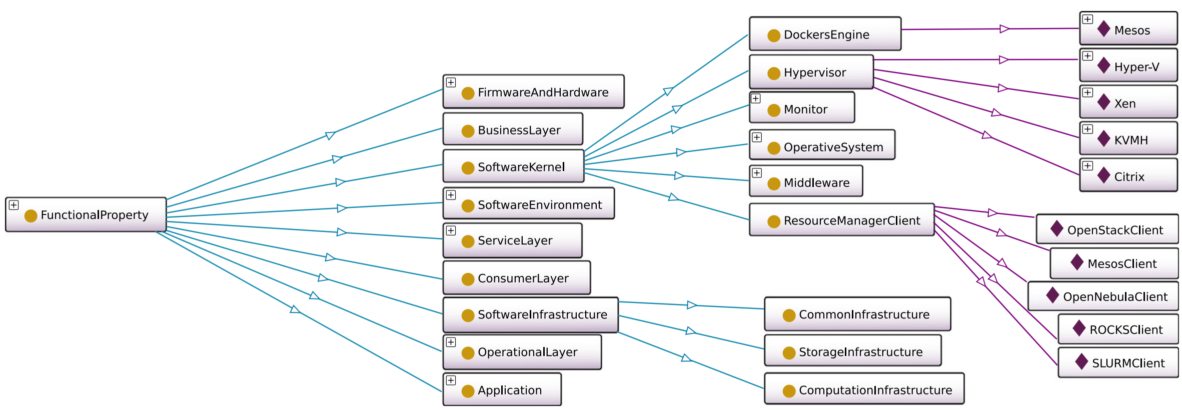


Fig. 5. Layer class of CL-Ontology.

included with **100GbEthernet** as a subclass to describe the IEEE-802.3bj protocol and Infiniband connections. However, as specific services might require specialized libraries only supported by concrete hardware, an object property **requiresLibrary** has been added to describe this relationship between the **Coprocessor** and *Library* classes .

The *OperationalLayer* class was incorporated into the mOSAIC ontology based on the IBM Cloud Computing Reference Architecture to describe the operational infrastructure layer of cloud computing systems. However, it does not contain nested classes as the mOSAIC ontology focuses on interoperability in the Cloud from the user's perspective. To describe the different resource managers that can be currently deployed in Cloud systems, the **BaremetalResourceManager**, **ContainerResourceManager**, and **VMResourceManager** classes have been incorporated into this class, having as individuals: **OpenStack**, **Mesos**, **ROCKS**, and **SLURM**; and with the object properties: **offersVirtualMachines** in the **VMResourceManager** domain, **offersContainers** in **ContainerResourceManagers** domain, and **offersDedicatedServers** in **BaremetalReso urceManager** domain.

In the mOSAIC ontology, the *VirtualMachine* class was nested within the *ServiceLayer* class. The CL-Ontology moves this class into the **ResourceAbstraction** class, which is placed as a subclass of the *ServiceLayer* class together with the **DedicatedServer** and **DockerContainer** subclasses. The **ResourceAbstraction** class describes current resource abstraction methods in the cloud being used for deploying user applications – containers, virtual machines, and dedicated servers – these are a main cause of Cloud management fragmentation. All classes and subclasses using containerization techniques has been named with the prefix *Docker*, to differentiate them from mOSAIC ontology concept of "CloudletContainers", used as entities to negotiate user SLAs with the "Cloud Negotiator" and "Cloud Mediator".

The *SoftwareKernel* class describes the software internals of servers that virtualize resources offered to users. The *Hypervisor*, *Middleware*, *Monitor*, and *OperatingSystem* subclasses have been extended with the **DockersEngine** subclass to describe a software kernel supporting containers, instead of traditional virtual machines supported by the *Hypervisor* class. Moreover, the **ResourceManagerClient** subclass has been added to the *SoftwareKernel* class to describe the software clients installed in compute and storage servers and used by resource managers to govern resources. The *Monitor* class has been extended to represent a broader concept of monitoring resources with the **ResourceAbstractionMonitor** subclass being a modification of the *VirtualMachineMonitor* class of the mOSAIC ontology and representing the monitoring of resources abstractions. The **PhysicalResourcesMonitor** subclass has been added to support the monitoring of physical hardware resources.

Fig. 5 depicts new individuals incorporated into the CL-Ontol ogy: **Hyper-V** and **Citrix** into the *Hypervisor* class, **Mesos** into the

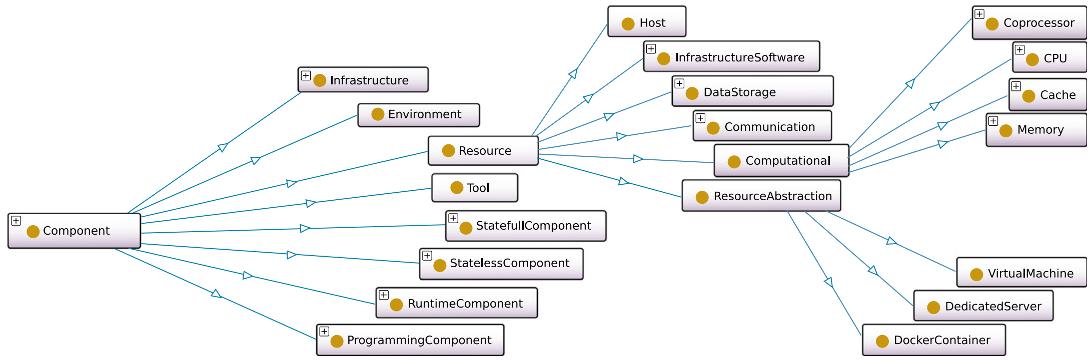


Fig. 6. Components class and subclasses in CL-Ontology.

DockersEngine class; and **OpenStackClient**, **MesosClient**, **OpenNebulaClient**, **RocksClient**, and **SlurmClient** into the **ResourceManagerClient** class.

The *Technology* class, has been extended with the **ResourceAbstractionMethods** class. The *VirtualizationTechnology* class has been nested into this class. In addition, the **DockerTechnology** class has been incorporated to support containerization technologies, and the **BareMetalTechnology** class is used to represent direct access to dedicated resources without the use of virtualization/containerization techniques. The object properties have been completed with the addition of the **hasAbstractionMethod** property, and nested subproperties: **hasDockerTechnology** and **hasBaremetalTechnology**, with the *hasVirtualizationTechnology* class, being reused from the mOSAIC ontology.

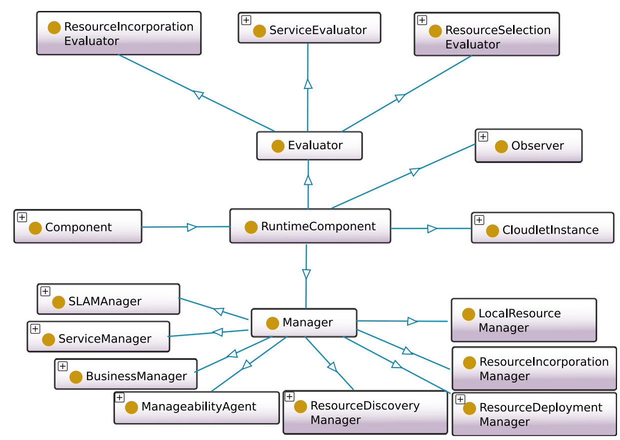
3.4. The Component, Runtime and Stateless classes

The *Component* class is the main class of the CL-Ontology and it contains all the cloud elements. Foremost among these are resources, services, and elements of the infrastructure. Fig. 6 depicts the subclasses of the *Component* class. These are the *Infrastructure* subclass, the *Environment* subclass, the *Resources* subclass, the *Tool* subclass, the *StatefullComponent* subclass, the *StatelessComponent* subclass, the *RuntimeComponent* subclass, and the *Programming-Component* subclass.

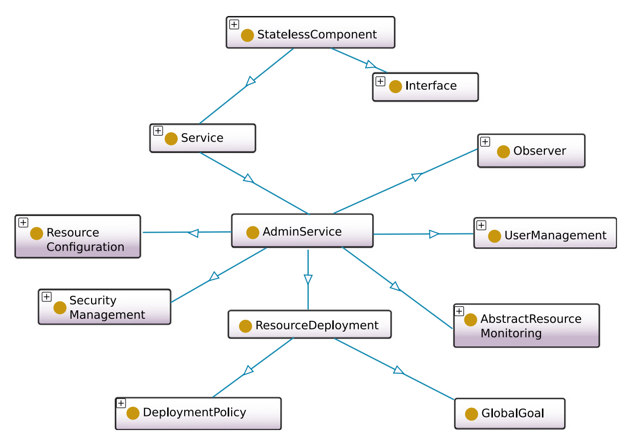
Most of classes and subclasses contained in the *Component* are subclasses of other classes described above. For brevity, only the modified classes and those appended to the mOSAIC ontology are described.

The *RuntimeComponent* class, Fig. 7(a), contains software elements performing management, selection, and evaluation tasks in cloud environments. The **ResourceIncorporationEvaluator** subclass is nested within the *Evaluator* subclass of the *RuntimeComponent* class and describesresources incorporated into the cloud resource fabric. The **ResourceSelector** class describes the selection of possible implementations and resource abstraction methods associated with users requests. The *Manager* subclass contains the **ResourceIncorporationManager** subclass, describing the actions performed in incorporating a newly added resource into the cloud fabric, the **ResourceDiscoveryManager** class describes the selection of resources that is used for deploying a service, and the **ResourceDeploymentManager** class describes the management of composition of services in a unified manner and it describes communicating single deployments of each service to the **LocalResourceManager** subclass, in which individuals are **OpenStack**, **Kubernetes**, and **Marathon**.

The *Stateless* class is depicted in Fig. 7(b). It describes elements of the services that do not have persistent state. It is contained within the *Interfaces* subclass and the *Services* subclass. The *AdminServices* subclass has been extended from the mOSAIC ontology to incorporate resource abstraction methods and resources



(a) RuntimeComponent.



(b) Stateless component.

Fig. 7. Stateless and Runtime component.

deployment process support. Therefore, the **AbstractedResourceMonitoring** class, described above as it is also a subclass of *FunctionalProperties*, replaces the *ResourceMonitoring* class. Moreover, the **ResourceDeployment** class describes the actions for deploying resource abstractions in physical resources. It represents the interests of cloud providers who express their objectives in a resource deployment policy manner — e.g., improving the energy consumed by servers, or maximizing servers utilization, a **DeploymentPolicy** class and a **GlobalGoal** class have been incorporated into the ontology, representing efficiency objectives of providers, both are subclasses of the **ResourceDeployment** class.

Finally, the *Statefull* class in which the modifications over the mOSAIC ontology have been described in this section within the *Layer*, *FirmwareAndHardware*, and *OperationalLayer* classes, as *Statefull* subclasses are also subclasses supporting these entities.

4. Proposed architecture

A Service Oriented Architecture (SOA) allow end users to describe their cloud applications as service delivery work-flows descriptions or *blueprints*. These blueprints are compositions of functional components, associated resources, configurations, and sequencing constraints of those components. SOAs are suggestive of an organization in which components assume control over specific roles in an effort to isolate the cloud customer from the internal functioning of the Cloud and to recognize the benefits of allowing the cloud provider to have full control over their resources, a separation of concerns detailing how customers can focus on application life cycle management while allowing cloud providers to focus on the underlying resource life cycle management would create opportunities for significant optimizations in both domains. Example frameworks employing the SOA philosophy include OpenStack Solum [46] and Apache Brooklyn [47] in PaaS environments, and OpenStack Heat [48] in IaaS environments, however, none of these offer the separation of concerns as expressed above.

The architecture proposed in this paper is an SOA, introducing separation of concerns [49] as described above. This separation augments the traditional service oriented architectures by allowing users to concentrate on describing functional components, configurations, and service level agreement constraints and by allowing the cloud provider to concentrate on providing the appropriate resources to satisfy those requirements. In a cloud environment consisting of an heterogeneous resource fabric, there may be many different types of resource that fulfill the user's requirements. In that situation, the separation of concerns principle would allow the cloud service provider to make a final choice, and in doing so to optimize efficiency objectives such as improving resource utilization and reducing energy consumption.

Fig. 8 depicts a heterogeneous cloud infrastructure where the cloud management domains are separated into logical partitions each of which is in principle managed by dedicated resource manager. In practice these domains may be created dynamically by a *Resource Coordinator* as novel hardware types are incorporated into the resource fabric. The *Resource Coordinator* consists on several functional blocks that can be deployed as agents or services, several information storage components, and a *Semantic Engine*. This engine is the interface between the Cloud and its users, and the Cloud and candidate resources applying for incorporation into the resource fabric.

4.1. Resource coordinator service delivery work flow

The *Resource Selection* component of the *Semantic Engine* receives blueprints submitted by users (Label (1) in Fig. 8). Next to receive a blueprint, the *Resource Selection* component analyses the blueprint using the CL-Ontology. In this process, the *Service Catalog* is queried to obtain information on available implementations associated with the services requested by the blueprint. The *Service Catalog* stores available implementations offered by the cloud provider, supported by diverse resource abstractions on the heterogeneous hardware. For example, a matrix multiplication service may be recorded in the catalog as having implementations based on diverse physical and abstracted resources such as CPU, GPU, FPGA, containers and virtual machines.

The *Resource Selection* component uses the list of possible implementations and consults the CL-Ontology to construct a seman tic-based *resource blueprint*, exploiting the semantic and syntactic structure of the CL-Ontology. This is forwarded to the *Resource Discovery* component (Label (2) in Fig. 8).

The *URDs Storage* contains a description of all resources that are part of the infrastructure, such as endpoints for deploying and monitoring resources, available abstraction methods, and managers information.

The *Resource Discovery* selects a hardware type and abstraction method associated with an existing resource in the cloud infrastructure. Accessing the *URDs Storage*, the *Resource Discovery* component retrieve information on concrete resources and technologies to support each service described in the *resource blueprint*. This information, and metrics provided by monitoring frameworks deployed in the infrastructure, act as inputs to heuristics that decide the efficiency objectives defined by providers.

As a result of this process, a specific implementation is chosen and captured in a *resourced blueprint* specification, embodying resources in a particular logical partition of the Cloud. If no appropriate resource is found, the request is rejected and this outcome is communicated to the end user. Alternatively, if an appropriate resource is found, the *Resource Deployment* component (Label (3) in Fig. 8) interfaces with the appropriate logical partition via its logical resource manager to deploy the service as per user requirements.

4.2. Incorporating heterogeneous resources into the resource fabric

The process of incorporating heterogeneous resources into the cloud management domain is transparently executed using a Plug & Play mechanism [50]. The Resource Incorporation plugin (*RI-plugin*) deployed on each resource registers/deregisters the physical hardware with the *Resource Coordinator*. The information collected for registering a resource includes a description of the physical features, a mechanism for accessing the resource, a framework description associated with the management of the resource, and a telemetry endpoint with which the resource is registered and from which utilization information can be retrieved. As part of the registration process, the *Semantic Engine* receives the resource description and from it a semantic-based Unified Resource Description (URD) is created, referencing the semantic and syntactic structure of the CL-Ontology. The URD enables the management of multiple heterogeneous resources in a uniformed manner. URDs can thus describe bare metal, virtual machines, containers, networked hardware being treated as a group to preserve connectivity information, and software resource managers, hiding locally managed subsystems. Servers with attached accelerators such as GPUs, MICs and DFEs typically cannot be virtualized due to the specific nature of the accelerators. To incorporate these resources into the Cloud, these server–accelerator pairs can also be represented as a URDs. In some cases, it may be possible to virtualize the server and to associate a partition of its accelerator with that virtualized component. In that case, the virtual component and the accelerator partition are seen as a single URD. The granularity of an URD is thus dependent on what aspect of the resource is being exposed to the cloud.

Finally, as result, the URD is stored into the *URDs Storage* and, in this manner, can be accessed from *Resource Discovery* component to schedule deployments and, therefore, from *Resource Deployment* component to deploy services.

5. CloudLightning — realization of proposed architecture

The CloudLightning (CL) project is based on the principles of self-organization and self-management (SOSM). It addresses the complexity introduced by heterogeneous resources in large scale cloud computing infrastructures. Fig. 9 depicts the high level

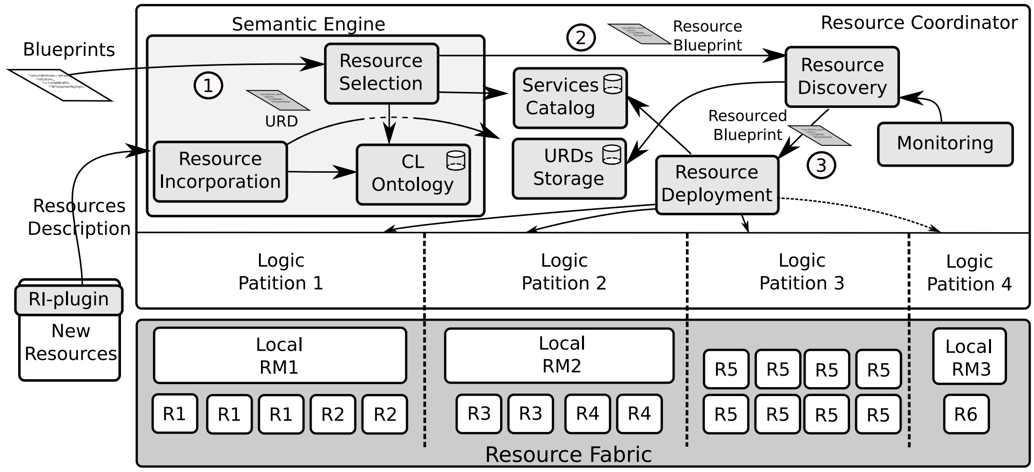


Fig. 8. CL-Ontology proposed architecture overview.

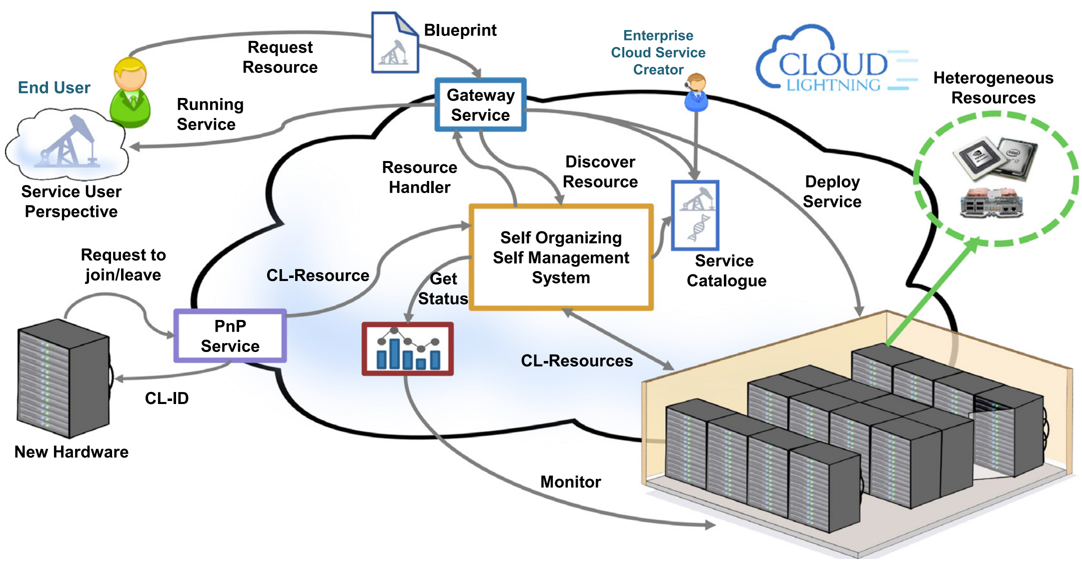


Fig. 9. High level overview of CloudLightning system.

overview of the components that build the CloudLightning architecture and the interactions between users and the system.

The CL project focuses on: (1) creating a heterogeneous cloud by exploring how specialized hardware, including high-performance computing machines, can be seamlessly integrated into the cloud resource fabric; (2) creating an SOA augmented with a separation of concerns approach to application and resource management; (3) moving the resource abstraction selection to the management of the cloud, allowing cloud providers to incorporate efficiency objectives on the resource fabric; and (4) addressing interoperability problems arising from using multiple heterogeneous resource managers at scale.

The CL system consists of a number of self-contained, loosely-coupled components.

- The **Gateway Server** component processes user requests in the form of blueprints. Each request is decomposed in multiple services and given to the Self Organising Self Management system (SOSM). Appropriate resources are located in infrastructure, the Gateway server then deploys services of the blueprint on to those resources using the deployment manifest associated with each service.
- The **SOSM system** component to interact with multiple resource management domains to identify appropriate resources that satisfy service level agreements and the efficiency objectives of the CSP.
- The **CL-Plug and Play** component to dynamically incorporate, remove, and configure new hardware resources and resource managers, existing or envisaged — including subsystems, such as HPC systems.
- A **Telemetry** component to interface with multiple telemetry endpoints and provides a uniform interface between those end points and SOSM system.

Fig. 10 shows detailed view of the CL architecture as realization of the proposed architecture depicted in Fig. 8. On the left hand side, two interaction points with the CL system are shown: a new resource incorporation process into the cloud resource fabric, and the arrival of a blueprint.

When a Resource Incorporation (*RI*) plugin attempts to register a resource with the Cloud, the Plug and Play Server (*PnP server*). The role of the resource incorporation in the CloudLightning system is performed by the Plug and Play Server (*PnP server*), and it behaves at it is described in Section 4.

Therefore, each request received for registering/deregistering resources with the SOSM system is analyzed in the context of the *CL-Ontology* component, which checks the correctness of each request. Afterwards, a semantic-based *CL-Resource* (corresponding to *URD* in Fig. 8) is created in which the most significant information of the resource is represented, providing access to the physical resource via a unique identifier. The newly created *CL-Resource* is stored in a CL-Resource Storage (*URDs Storage* in Fig. 8), which is realized CL system by using a mongoDB database for high availability. This database holds information on the resources which can be used by the SOSM when processing resource requests to satisfy blueprint descriptions.

Users requests, in the formal blueprints (Label 1 at Fig. 10), are processed by the *Gateway Server* component. There, requests are decomposed and, depending on the services required by the blueprint and on the available implementations for each service within the *services catalog*, appropriate resources are identified and recorded in a in the blueprint. These are forwarded to the SOSM system, specifically to the *Cell Manager* (Label 2 at Fig. 10).

The internal structure of the *SOSM* system [51] acts as the *Resource Discovery* component as described in Fig. 8. Its hierarchical structure orchestrates multiple user requests targeting properly at the underlying logic partitions. The *Cell Manager* component, on top of the SOSM hierarchy, implements the semantic engine functionality provided by the *Resource Selection* component in Fig. 8. By using the *CL-Ontology*, semantic-based services requests are issued which result in a *resourced blueprints*. In such a blueprint, the identity of specific physical resources and resource abstractions are recorded and into which the appropriate service implementation will subsequently be deployed.

Requests are sent from the *Cell Manager* to one of the *pRouters* identifying the logical partition of the Cloud in which establish heterogeneous resources reside. *pSwitches*, situated under the *pRouters*, further partition the resource space into smaller and more efficiently manage domains. The lowest logical level in the *SOSM* hierarchy is occupied by virtual Rack Managers (*vRMs*). These resource managers directly control and, where appropriate, virtualize a subset of the physical resource fabric whose hardware type is reflected in on the type of is parent *pRouter*. While navigating through the *SOSM* hierarchy, resource requests are directed along a path which automately results in them being satisfied by the "most suitable" underlying physical resource. This suitability is determined by the upward propagation of the status of the physical resources and associated metrics are combined, in each level of the hierarchy, into a measure known as Suitability Index (SI) [51]. In effect, requests follow the path determined by the highest value of the SI at each level in the hierarchy. Moreover, the efficiency objectives of the cloud provider can be captured as a vector of weights that is propagated downwards through the hierarchy, becomes part of the SI calculation and thus is used bias the resource selection, reflecting the relative importance dynamically placed by the provider on each objective.

The process of deploying multiple resource abstraction methods — virtual machines and containers and of having them cooperate, would require different resource managers to collaborate to deliver a composition of service hosted across these different resource types. From the users perspective, all services supported by heterogeneous resources should interact seamlessly as if they were in the same resource pool. Therefore, an integration strategy is used in CloudLightning to create a unified virtual network infrastructure management across all platforms horizontally [52]. OpenStack Neutron is used to connect physical resources to the same networking infrastructure but each is managed by an appropriate, and possibly distinct, resource manager.

When a service component *resource blueprint* is received at the *vRM* level, a resource abstraction method is initialized in order to support it. As result, associated to each service described in the original *blueprint*, an homologous service is created in a *resourced template* filled with resource abstraction type, endpoint, accessing method, and implementation chosen.

Once the *resourced template* is adequately filled with the information required to access the resource abstractions, it is returned to the *Gateway Server* (label 4 at Fig. 10), where information contained is used for deploying implementations associated to services [53] (label 5 at Fig. 10). By implementing this functionality, the *Gateway Server* realizes the *Resource Deployment* component described in the proposed architecture.

Finally, the control of the application is given back from the *Gateway Server* to the end user.

6. Conclusions and future work

This work was motivated by the desire to simplify the deployment of collaborating services across heterogeneous resources in a manner that automated resource interoperability. Prior to this work, expert users were required to manually configure this collaborative environments. By creating a formal ontology capturing heterogeneity across resource abstraction methods, physical resources, and resource managers, it became possible to express HPC-like environments within the Cloud. Thus, the Cloud and HPC machines, traditionally considered to be incompatible, could be profitably combined within the same design space.

Designing an ontology from scratch for such complex systems is an enormous undertaken. To make the task tractable and to increase end user acceptance, the CL-Ontology leverage the mOSAIC ontology, which is part of the SIIF-IEEE 2302 Standard, and extend it appropriately to capture the emerging heterogeneous cloud. This work is currently proposed as basis of the IEEE-2303, Standard for Adaptive Management of Cloud Computing Environments [54], with the hope of being supported and updated periodically.

The Ontology described in this work attempt to guide cloud architects in production of the next generation of cloud systems by:

- creating a central knowledge repository in which a common understanding of the information can be shared and improved.
- alleviating current interoperability and lock-in vendor issues between resource managers, resource abstractions, physical resources, and public and private cloud software stacks in inter/intra cloud environments.
- showing that to construct a system in which multiple heterogeneous resources can coexist in the same cloud environment is possible.
- proposing an ontology as part of a semantic engine to address the complexity of building HPC environments in Cloud.

In addition to the creation of the CL-Ontolgy, this paper proposes a conceptual architecture for supporting the processes of dynamic incorporation of hardware accelerators into the cloud resource fabric, and ontology-based resource management. An realization of this conceptual architecture is presented into the CloudLightning system to illustrate how the CL-Ontology can be applied for autonomous resource management. Although this work is at an early stage of development, it has been used as an initial use case to illustrate the effectiveness of the semantic engine as an Ontology-based resource management for decision making processes.

Supporting dynamic expanding/contracting of resources into the cloud resource fabric, and increasing diversity in resource management/resource abstraction methods, focus future efforts on taking care of this aspects since the interaction points with the proposed architecture do not handle security and fault tolerance

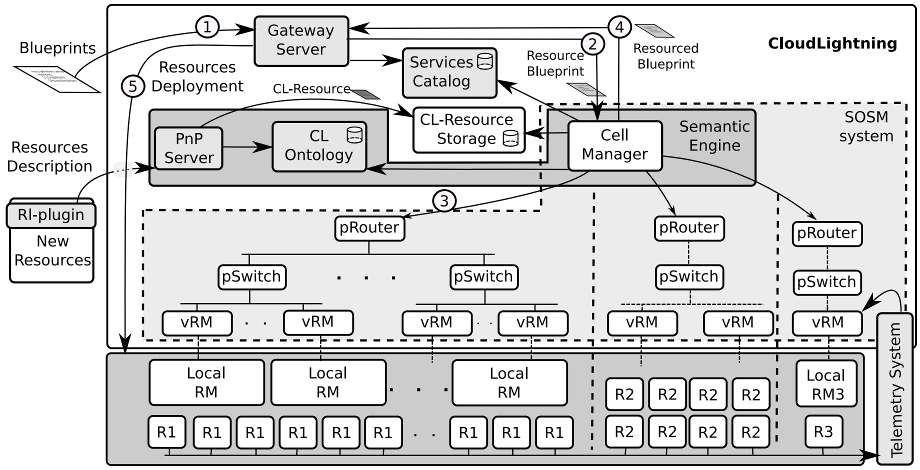


Fig. 10. Exploded view of CloudLightning architecture.

aspects. Therefore, additional work is needed to ensure an incremental design approach to incorporate to the CL-Ontology (1) fault tolerance in HPC in cloud environments; (2) security components and elements to avoid inadvertent requests or bogus resource incorporation actions that will result in corruption of the cloud information.

Finally, once the ontology-based resource management architecture will be integrated, future efforts will be placed on the evaluation of the performance, resources utilization, and energy consumption within the CloudLightning system, comparing the obtained results with other traditional resource management techniques.

Acknowledgment

This work is funded by the European Unions Horizon 2020 Research and Innovation Programme through the CloudLightning project under Grant Agreement Number 643946.

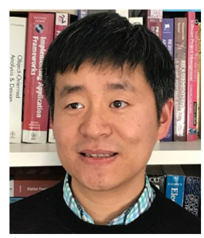
References

[1] C.V.N. Index, Forecast and methodology, 2014–2019 white paper, 2015. Retrieved 23rd September.
[2] S.R. Group, 2016 Review Shows $148 billion Cloud Market Growing at 25% Annually, Tech. Rep., Synergy Research Group, Reno, NV, United States, 2017.
[3] Garner, Gartner Says Worldwide IT Spending Forecast to Grow 2.7 Percent in 2017, Tech. Rep., Garner, STAMFORD, Conn, United States, 2017.
[4] J. Fowers, G. Brown, P. Cooke, G. Stitt, A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications, in: Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays, FPGA '12, ACM, New York, NY, USA, 2012, pp. 47–56. http://dx.doi.org/10.1145/2145694.2145704. http://doi.acm.org/10.1145/2145694.2145704.
[5] J.P. Walters, A.J. Younge, D.I. Kang, K.T. Yao, M. Kang, S.P. Crago, G.C. Fox, GPU passthrough performance: A comparison of KVM, Xen, VMWare ESXi, and LXC for CUDA and OpenCL applications, in: 2014 IEEE 7th International Conference on Cloud Computing, CLOUD, IEEE, 2014, pp. 636–643.
[6] S.P.T. Krishnan, S.P.T. Krishnan, B. Veeravalli, V.H. Krishna, W.C. Sheng, Performance characterisation and evaluation of WRF model on cloud and HPC architectures, in: 2014 IEEE Intl. Con.f on High Performance Computing and Communications, 2014 IEEE 6th Intl. Symp. on Cyberspace Safety and Security, 2014 IEEE 11th Intl. Conf. on Embedded Software and Syst, HPCC, CSS, ICESS, 2014, pp. 1280–1287. http://dx.doi.org/10.1109/HPCC.2014.218.
[7] E. Serrano, G. Bermejo, J.G. Blas, J. Carretero, Evaluation of the feasibility of making large-scale X-Ray tomography reconstructions on clouds, in: 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2014, pp. 748–754. http://dx.doi.org/10.1109/CCGrid.2014.106.
[8] P. Zaspel, M. Griebel, Massively parallel fluid simulations on Amazon's HPC cloud, in: 2011 First International Symposium on Network Cloud Computing and Applications, 2011, pp. 73–78. http://dx.doi.org/10.1109/NCCA.2011.19.
[9] AWS announces seven new compute services and capabilities to support an even wider range of workloads. http://www.businesswire.com/news/home/20161130006132/en/AWS-Announces-Compute-Services-Capabilities-Support-Wider. (Accessed 10 December 2017).
[10] EC2 instances (F1) with programmable hardware. https://aws.amazon.com/blogs/aws/developer-preview-ec2-instances-f1-with-programmable-hardware/. (Accessed 17 December 2017).
[11] OpenStack Nova. http://docs.openstack.org/developer/nova/. (Accessed 17 December 2017).
[12] Open Nebula. https://opennebula.org/ (Accessed 17 December 2017).
[13] Kubernetes. http://kubernetes.io/ (Accessed 16 December 2017).
[14] Apache Mesos. http://mesos.apache.org/documentation/latest/. (Accessed 11 December 2017).
[15] Docker Swarm. https://github.com/docker/swarm. (Accessed 16 December 2017).
[16] OpenStack Magnum. https://github.com/openstack/magnum. (Accessed 12 December 2017).
[17] OpenStack Ironic. http://docs.openstack.org/developer/ironic/deploy/user-guide.html. (Accessed 10 December 2017).
[18] OpenStack Heat. https://slurm.schedmd.com/. (Accessed 10 December 2017).
[19] OpenStack Heat. http://www.rocksclusters.org/wordpress/. (Accessed 13 December 2017).
[20] Amazon EC2 Instance types. https://aws.amazon.com/ec2/instance-types/. (Accessed 29 May 2017).
[21] F.T. Imam, Application of ontologies in cloud computing: The state-of-the-art, 2016. arXiv preprint arXiv:1610.02333.
[22] IEEE P2302 — Standard for Intercloud Interoperability and Federation (SIIF). https://standards.ieee.org/develop/project/2302.html. (Accessed 10 December 2017).
[23] IEEE P2301 — Guide for Cloud Portability and Interoperability Profiles (CPIP). https://standards.ieee.org/develop/project/2301.html. (Accessed 13 December 2017).
[24] H2020 EU CloudLightning project, self-organising, self-managing heterogeneous cloud. http://cordis.europa.eu/project/rcn/194118_en.html. (Accessed 13 December 2017).
[25] T. Lynn, H. Xiong, D. Dong, B. Momani, G. Gravvanis, C. Filelis-Papadopoulos, A. Elster, M.M.Z.M. Khan, D. Tzovaras, K. Giannoutakis, D. Petcu, M. Neagul, I. Dragon, P. Kuppudayar, S. Natarajan, M. McGrath, G. Gaydadjiev, T. Becker, A. Gourinovitch, D. Kenny, J. Morrison, Cloudlightning: A framework for a self-organising and self-managing heterogeneous cloud, in: Proceedings of the 6th International Conference on Cloud Computing and Services Science, 2016, pp. 333–338.
[26] O.C. Manifesto, Open cloud manifesto, vol. 20, 2009. Availabe online: www.opencloudmanifesto.org/Open.
[27] Unified cloud interface project. https://code.google.com/archive/p/unifiedcloud/. (Accessed 12 December 2017).
[28] A. Parameswaran, A. Chaddha, Cloud interoperability and standardization, SETlabs Brief. 7 (7) (2009) 19–26.
[29] O.-W.A. Edmonds, A. Papaspyrou, T. Metsch, Open cloud computing interface-core, Update, 2016, p. 6.
[30] W.O.W. Group, et al. {OWL} 2 web ontology language document overview, 2009.
[31] Y. Zhao, C. Liao, X. Shen, Poster: An infrastructure for HPC knowledge sharing and reuse, in: Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ACM, 2017, pp. 461–462.

[32] A. Tenschert, Ontology matching in a distributed environment, 2016.
[33] T. Takahashi, Y. Kadobayashi, H. Fujiwara, Ontological approach toward cybersecurity in cloud computing, in: Proceedings of the 3rd International Conference on Security of Information and Networks, ACM, 2010, pp. 100–109.
[34] M.Á. Rodríguez-García, R. Valencia-García, F. García-Sánchez, J.J. Samper-Zapater, Ontology-based annotation and retrieval of services in the cloud, Knowl.-Based Syst. 56 (2014) 15–25.
[35] A. Tahamtan, S.A. Beheshti, A. Anjomshoaa, A.M. Tjoa, A cloud repository and discovery framework based on a unified business and cloud service ontology, in: 2012 IEEE Eighth World Congress on Services, SERVICES, IEEE, 2012, pp. 203–210.
[36] F. Moscato, R. Aversa, B.D. Martino, T.F. Forti, V. Munteanu, An analysis of mOSAIC ontology for Cloud resources annotation, in: 2011 Federated Conference on Computer Science and Information Systems, FedCSIS, 2011, pp. 973–980.
[37] ISO/IEC JTC1 — Standard for Cloud computing and distributed platforms. https://www.iso.org/isoiec_jtc1sc38.html. (Accessed 14 December 2017).
[38] ISO/IEC 18384-3:2016 — Information technology – Reference Architecture for Service Oriented Architecture (SOA RA). https://www.iso.org/obp/ui/#iso:std:iso-iec:18384:-3:ed-1:v1:en. (Accessed 14 December 2017).
[39] A.J. Ferrer, D.G. Pérez, R.S. González, Multi-cloud platform-as-a-service model, functionalities and approaches, Procedia Comput. Sci. 97 (2016) 63–72.
[40] N. Bassiliades, M. Symeonidis, G. Meditskos, E. Kontopoulos, P. Gouvas, I. Vlahavas, A semantic recommendation algorithm for the PaaSport platform-as-a-service marketplace, Expert Syst. Appl. 67 (2017) 203–227.
[41] K.U. Sri, M.B. Prakash, J. Deepthi, A frame work to dropping cost in passage of CDN into hybrid cloud, Int. J. Innov. Technol. Res. 5 (2) (2017) 5829–5831.
[42] A. Rossini, Cloud Application Modelling and Execution Language (CAMEL) and the PaaSageWorkflow, Tech. Rep. ESOCC-EU Projects Track, Springer, Taormina, Italy, 2015.
[43] E. Di Nitto, G. Casale, D. Petcu, On MODAClouds toolkit support for DevOps, in: Advances in Service-Oriented and Cloud Computing: Workshops of ESOCC 2015, in: Revised selected papers, vol. 567, Springer, Taormina, Italy, September 15–17, 2015, 2016, p. 430.
[44] B. Koller, N. Struckmann, J. Buchholz, M. Gienger, Towards an environment to deliver high performance computing to small and medium enterprises, in: Sustained Simulation Performance 2015, Springer, 2015, pp. 41–50.
[45] Open semantic framework — adding ontology concept. http://wiki.opensemanticframework.org/index.php/Adding_an_Ontology_Concept_using_Prot%C3%A9g%C3%A9. (Accessed 21 March 2018).
[46] Solum. https://github.com/openstack/solum. (Accessed 12 December 2017).
[47] Apache Brooklin. https://brooklyn.apache.org. (Accessed 11 December 2017).
[48] OpenStack Heat. https://github.com/openstack/heat. (Accessed 15 December 2017).
[49] D. Dong, H. Xiong, J.P. Morrison, Separation of concerns in heterogeneous cloud environments workshop on tools for an energy efficient cloud, in: Workshop on Tools for an Energy Efficient Cloud, TEEC 2017, IEEE, 2017.
[50] G.G. Castañe, D. Dong, H. Xiong, J.P. Morrison, A plug-and-play mechanism for incorporating heterogeneous resources into cloud environments, J. Syst. Softw. (in press).
[51] C. Filelis-Papadopoulos, H. Xiong, A. Spătaru, G.G. Castañé, D. Dong, G.A. Gravvnis, J.P. Morrison, A generic framework supporting self-organisation and self-management in hierarchical systems, in: Parallel and Distributed Computing (ISPDC), 2017 16th International Symposium on, IEEE, 2017, pp. 149–156.
[52] D. Dong, P. Stack, H. Xiong, J.P. Morrison, Managing and unifying heterogeneous resources in cloud environments, in: IEEE 7th International Conference on Cloud Computing and Services Science, CLOSER 2017, IEEE, 2017.
[53] T. Selea, I. Drăgan, T.-F. Fortiş, The cloudlightning approach to cloud-user interaction, in: Proceedings of the 1st International Workshop on Next generation of Cloud Architectures, ACM, 2017, p. 4.
[54] IEEE P2303 — standard for adaptive management of cloud computing environments. https://standards.ieee.org/develop/project/2303.html. (Accessed 08 December 2017).

Gabriel González Castañé is a Postdoctoral Researcher in University College of Cork. He holds a Ph.D. in Computer science from University Carlos III of Madrid on the topic of energy aware Cloud Computing simulations and a Masters in Distributed Systems. He has participated in several Spanish National Projects and he led a simulation workpackage in CACTOS FP7 project. Currently he is the technical coordinator of CloudLightning – self organization and self management clouds – H2020 project. His interests are discrete event simulation, cloud computing, autonomic computing and distributed systems.

Dr. Huanhuan Xiong is a Senior PostDoctoral Researcher in University College Cork. She received a B.Sc. in Economics from Wuhan University of Technology (Wuhan, China) in 2004, and a M.Sc. in Software Engineering and a Ph.D. in Geographic Information System (GIS) from Wuhan University (Wuhan, China) in 2006 and 2012. She worked in IC4 (Irish Centre for Cloud Computing & Commerce) for three years, and she has expertise in cloud migration, cloud architecture, cloud interoperability and scalability. Her research Interests include cloud architecture, game theory, self-organized and self-optimized systems.



Dapeng Dong is a senior Postdoctoral Researcher at the Boole Centre for Research in Informatics of University College Cork, Ireland. He received his Ph.D. in computer science and M.Sc. in Software and Systems for Mobile Networks from University College Cork, Ireland. His research interests include self-organizing and self-managing cloud architecture, cloud resource optimization, and big data analytics.

Professor John P. Morrison (Male) B.Sc., M.Sc, Ph.D., Dip. TLHE, Senior MACM, Senior MIEEE.

John Morrison is the founder and director of the Centre for Unified Computing. He is a co-founder and co-director of the Boole Centre for Research in Informatics and a co-founder and co-director of Grid-Ireland. Prof. Morrison has held a Science Foundation of Ireland Investigator award. Currently, he is a Principle Investigator in the Irish Centre for Could Computing and Commerce and is the coordinator of the European H2020 funded CloudLightning project, which seeks to create a self-organizing, self-managing, heterogeneous cloud architecture. Prof Morrison has published widely in the field of Parallel Distributed and Grid Computing and has been the guest editor on many journals including the Journal of Super Computing and the Journal of Scientific Computing. He is on the Editorial Board of Multi-Agent and Grid Systems. He is a senior member of the ACM and a senior member of the IEEE. Prof Morrison is a member of the I2Lab Advisory Board in the University of Central Florida. He has served on dozens of international conference programme committees and is a co-founder of the International Symposium on Parallel and Distributed Computing.