PaaSport Semantic Model: An Ontology for a Platform-as-a-Service Semantically Interoperable Marketplace

Nick Bassiliades[a,b,*], Moisis Symeonidis[a], Panagiotis Gouvas[c], Efstratios Kontopoulos[a,d], Georgios Meditskos[a,d], and Ioannis Vlahavas[a,b]
[a]*School of Science & Technology, International Hellenic University, 14th km Thessaloniki - N. Moudania, GR 57001 Thermi, Thessaloniki, Greece*
[b]*Dept. of Informatics, Aristotle University of Thessaloniki, GR 54124 Thessaloniki, Greece*
[c]*Ubitech Ltd., Thessalias 8 & Etolias 10, GR 15231 Chalandri, Athens, Greece*
[d]*Information Technologies Institute, Centre of Research & Technology Hellas, 6th km Charilaou-Thermi Rd, P.O. Box 60361, GR 57001 Thermi, Thessaloniki, Greece*

Abstract. PaaS is a Cloud computing service that provides a computing platform to develop, run, and manage applications without the complexity of infrastructure maintenance. SMEs are reluctant to enter the growing PaaS market due to the possibility of being locked in to a certain platform, mostly provided by the market's giants. The PaaSport Marketplace aims to avoid the provider lock-in problem by allowing Platform provider SMEs to roll out semantically interoperable PaaS offerings and Software SMEs to deploy or migrate their applications on the best-matching offering, through a thin, non-intrusive Cloud broker. In this paper we present the PaaSport semantic model, namely an OWL ontology, extension of the DUL ontology design pattern. The ontology is used for semantically annotating a) PaaS offering capabilities and b) requirements of applications to be deployed. The ontology has been designed to optimally support a semantic matchmaking and ranking algorithm that recommends the best-matching PaaS offering to the application developer. The DUL design pattern offers seamless extensibility, since both PaaS Characteristics and parameters are defined as classes; therefore, extending the ontology with new characteristics and parameters requires the addition of new specialized subclasses of the already existing classes, which is less complicated than adding ontology properties.

Keywords: Cloud computing, Platform-as-a-Service, Semantic Interoperability, Ontology, Cloud Marketplace

1 Introduction

Platform-as-a-Service (*PaaS*) is a Cloud computing service model, where a provider company provides a computing platform allowing customers to develop, run, and manage web applications created using programming languages, libraries, services, and tools, typically supported by the PaaS provider [40]. The consumer of the PaaS does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment. Compared to the *Infrastructure-as-a-Service* (*IaaS*) model, where the consumer has control over operating systems, storage, and might have limited control of select networking components, in the PaaS model the provider takes care of the complexity of building, configuring and maintaining the infrastructure layer, whereas the developer worries only about developing, testing and deploying an application on the provided platform. Finally, compared to the *Software-as-a-Service* (*SaaS*) model, where the provider's application is running on a cloud infrastructure and the consumer can just run them over the web, possibly with limited

* Corresponding author. E-mail: nbassili@csd.auth.gr.

user-specific application configuration settings capability, in the PaaS model the provider has full control over the application's lifecycle.

There are many market reports, such as [8], [13], [61], that indicate that PaaS has a very positive economic outlook for the Cloud market, with an expected growth rate higher than the whole Cloud market, as well as the IaaS and SaaS markets. PaaS, due to its service model, allows for low capital investment. It enables the deployment of applications without the need for provisioning hosting capabilities. It, thus, helps save on the cost incurred for buying and managing the underlying hardware and software. The PaaS model minimizes the incremental cost required for scaling the system with growth in the service usage, while allowing for resource sharing, reuse, life-cycle management, and automated deployment. For these benefits, PaaS is preferred over other solutions for application and service development.

Although giant vendors occupy this emerging space as de-facto standards, including Microsoft, Amazon, Google, and Salesforce.com, many small-medium companies try to enter the PaaS market, not without important inhibiting factors. The battle for dominance in the market between the big vendors makes them reluctant to agree on widely accepted standards promoting their own, mutually incompatible Cloud standards and formats [23]. This dominance increases the lock-in of customers in a single Cloud platform, preventing the portability of data or software created by them. However, even if in theory application and/or data portability is supported, the high complexity and in most cases the additional switching costs discourage users from doing so [2]. The high effort required for exporting application and data from a Cloud platform discourages start-ups and SMEs from entering and bestirring in the flourishing Cloud market [54]. Of course, vendor lock-in is ubiquitous in the computer market, but since the Cloud technology and market are still shaping, there is still a chance to avoid it.

Cloud specialists argue that in the years to come the leading enterprise software vendors, as well as the large Cloud providers will introduce new PaaS offerings, while both large and small Cloud PaaS providers will grow through partnerships [28]. The formation of partnerships and federations between heterogeneous Cloud PaaS Providers involves interoperability. The Cloud community and the European Commission [19] have realized the significance of interoperability and have initiated several approaches to tackle this challenging issue. The first efforts to explore interoperability in PaaS are also well on track, e.g. CAMP (Cloud Application Management for Platforms) [10]. Several studies indicate that Cloud community should:

- promote common standards and interoperability of public Cloud systems, to maximize economies of scale across the globe and create the preconditions for portability between Cloud vendors;
- create the pre-conditions so that the principle of data access and portability between Cloud vendors is widely accepted and the risk of lock-in of users in proprietary systems is prevented.

In order to lower the barriers that prevent the small-medium Cloud PaaS vendors and software companies from entering the PaaS market, the latter should be able to choose between different Cloud PaaS offerings, offered by the former, and should also be able to deploy their applications easily and transparently between Cloud providers whenever needed. For example, application developers can select either the most reliable platform, or the most well reputed one, or the most cost-efficient one, or simply the one that best meets the technical requirements of their services and applications [6]. Furthermore, application users could decide to switch between platforms, when an SLA (Service Level Agreement) is breached or when the cost is too high, without setting data and applications at risk, e.g. loss of data or inability to run the application on a different platform.

An open market of interoperable Cloud platforms will facilitate small-medium Cloud providers to enter the market and strengthen their position [22]. Furthermore, when interoperability problems have been resolved small-medium Cloud vendors could cooperate through new business models (e.g. in the form of coalitions) in order to cope with demand fluctuations. For example, an unexpected increase in processing power capacity could force Cloud providers to cooperate in order to overcome the problem of limited resources. Otherwise, SMEs would seem unreliable to provide the negotiated QoS (Quality of Service), leading consumers to rely on big players for hosting their services and data [28]. Notice, however, that this is only a visionary aspect of our work and it is outside the scope of this paper.

1.1 Scope of the paper

With the above problem formulation in mind, the PaaSport project [49] aims to resolve application and data portability issues that exist in the Cloud PaaS market through a flexible and efficient deployment and migration approach. These include, but are not limited to image conversion to be suitable for target

hypervisor, compression to aid, speed of transfer, image encryption, secure protocols, QoS guarantees, trust issues and cost sharing models. To this end, PaaSport combines Cloud PaaS technologies with lightweight semantics, in order to:

- specify and deliver a thin, non-intrusive Cloud broker (in the form of a Cloud PaaS Marketplace),
- implement the enabling tools and technologies, and
- deploy fully operational prototypes and large-scale demonstrators.

As already discussed, PaaSport's scope is to enable:

- Cloud vendors (in particular SMEs) to roll out semantically interoperable PaaS offerings, improving their outreach to potential customers, particularly the software industry.
- Software SMEs to seamlessly deploy business applications on the best-matching Cloud PaaS and/or migrate these applications on demand.

PaaSport contributes to aligning and interconnecting heterogeneous PaaS offerings, overcoming the vendor lock-in problem and lowering switching costs. Note that even if "giant vendors" are not our primary target group, since it would be unlikely for them to cooperate with a cloud broker that could potentially cause them to loose clients and, thus, register their offerings in our marketplace, we actually do support "giant vendors", such as Amazon, at the technical level, in order to ease deployment / migration of our marketplace client applications between clouds of "giant" vendors and smaller vendors. In this case, our marketplace clients have to make a separate agreement with e.g. Amazon, provide us with their keys for the platform, and then our system would just use the clouds public API to provide our brokering services.

In order the above to be realized, the PaaSport project has, among others, a) developed an open, generic, thin Cloud *broker architecture* for an interoperable PaaS offerings marketplace, b) defined a unified *semantic model* for PaaS offerings and business applications, c) developed a unified *PaaS API* (Application Programming Interface) that allows the deployment and migration of business applications transparently to the developer and independent of the specificities of a PaaS offering, d) implemented a *marketplace infrastructure* for semantically–interconnected PaaS offerings, e) defined a unified *service-level agreement model* addressing the complex characteristics and dynamic environment of the Cloud PaaS marketplace, f) defined a Cloud PaaS offerings discovery, shortlisting and *recommendation algorithm* for providing the user with the best-matching PaaS offering, and g) delivered a set of PaaS *marketplace utilities* and user-centric *front-ends*. In this paper, we briefly present the *broker architecture* (a) and we report on the unified *semantic model* (b) in detail. Furthermore, we also present how the semantic model has been integrated into the *persistence layer* of the PaaSport *marketplace infrastructure* (d).

Specifically, in order to support the PaaSport marketplace, a unified semantic model has been defined, in the form of an OWL ontology, for representing the necessary characteristics and attributes for the definition of the capabilities of PaaS offerings, the requirements of the business applications to be deployed through the proposed Cloud Marketplace, and the SLAs to be established between offering providers and application owners. The PaaSport ontology has been defined as an extension of the DOLCE+DnS Ultralite (DUL) ontology design pattern [26]. This offers extensibility, since both PaaS Characteristics and parameters are defined as classes, so extending the ontology with new characteristics and parameters requires just to add new classes as subclasses of the already existing ones, which is less complicated than adding new ontology properties. This extensibility advantage reflects also on the persistence layer of the PaaSport marketplace, which is built using a relational database. The relational database of PaaSport can be easily extended, as the semantic model evolves, without the need to change existing tables; only new ones need to be added.

Finally, the fact that PaaS Characteristics and parameters are represented as first-class objects allows the semantic matchmaking and ranking algorithm, developed for providing the application developer with the best-matching Cloud PaaS offering, to be extensible because it is agnostic to specific PaaS Characteristics and parameters. Notice that the recommendation algorithm is only briefly presented in this paper (Section 5) and is detailed in [5].

In the rest of this paper, we review related work in Section 2 on existing ontologies for Cloud computing. We then briefly present the PaaSport marketplace in Section 3, including its architecture, the requirements, use cases, and functionality relevant to the PaaSport semantic models. In Section 4 we present in detail the semantic models. Section 5 describes how the PaaSport Semantic Model interacts with the Recommendation and the Persistence layer of the PaaSport marketplace, briefly focusing on the recommendation algorithm. Section 6 evaluates the PaaSport Semantic Model by presenting the verification of the ontology and the evaluation of SPARQL query performance comparing to alternative semantic models. Finally,

Section 7 concludes the paper with a discussion on future work.

2 Related Work

Up until now, there is relevant research on semantically annotated Cloud services, which focused on partial aspects of Cloud computing relevant to PaaSport. However, no major breakthroughs have yet been reported in the field of discovering Cloud services and performing the matchmaking with the developers' functional requirements. This matchmaking process would further need to establish a defined minimum SLA among the offering provider and the SMEs. In the following, we briefly examine and present the most relevant work on semantic models relevant to the work performed within PaaSport.

Androcec et al. (2012) [1] surveyed Cloud Computing ontologies, according to type and applications and focused on four (4) categories of Cloud computing ontologies according to specific scopes:

- *Cloud resources and services description*: Ontologies in this category describe Cloud resources and services, classify the current services and pricing models or define new types of Cloud services. Representative examples belonging to this category can be found in [66], [65], [20] and [17].
- *Cloud security*, namely, models that describe and/or improve security in Cloud environments, such as [58], [39].
- *Cloud interoperability* that deals with how to use ontologies for achieving interoperability among different Cloud providers and their services. Representative examples include [42] and [31].
- *Cloud services discovery and selection*: This category includes the use of ontologies for discovering and selecting the best Cloud service alternative. The most characteristic of the numerous related approaches are [38], [9], [16], [30], and [64], while one of the most innovative ones is presented in [55]. The system architecture of the latter involves (a) a semantic annotation module that encapsulates domain ontologies, (b) a semantic indexing module utilized for discovery purposes, and, (c) a semantic search engine that is exposed to end-users. Using keyword-based search queries, matching and retrieval of identified Cloud services is performed.

Another related work paradigm, *OpenCrowd's Cloud Taxonomy* [44], focuses on the latter of the four categories above. More specifically, the Cloud Taxonomy is an online, freely navigable taxonomy that categorizes Cloud Services according to both their service model (IaaS, PaaS or SaaS) and application context. It enables users to discover and access Cloud services so that they can further navigate to respective home pages. Moving beyond just a static model, the Cloud Taxonomy is interactive, where users can contribute comments and recommend additional products to include, aiming at encouraging the dialog between Cloud computing services vendors and developers.

In [18], a unified taxonomy for IaaS and an IaaS architectural framework are presented The taxonomy is structured around seven layers: core service, support, value-added services, control, management, security and resource abstraction. The authors also introduce an IaaS architectural framework that relies on the unified taxonomy, describing the layers in detail and defining dependencies between the layers and components. The resource characteristics of the PaaSport model bare some resemblances with the resource abstraction layer of [18]. However, since PaaSport's semantic model focus is on PaaS, there are a lot of differences.

Kourtesis et al. [36] focus on semantic-based QoS management and monitoring for cloud-based systems and proposes a new framework that combines semantic technologies and distributed datastream processing techniques. Among the discussion of challenges and future directions, they mention that a proper semantic-based architecture for cloud computing should contains adequate definitions of functional and non-functional properties, as well as different cloud perspectives (technical vs. business). The PaaSport Semantic Model properly addresses the above on the PaaS Parameter and Characteristics level (see section 4.3.4).

Dastjerdi et al. [16] described a framework that facilitates the discovery of Cloud services (IaaS). End-users can register their requirements through a web portal including software, hardware & interfaces description. Using Open Virtualization Format (OVF), the corresponding disk images are generated. The description of users' requirements is expressed in a semantically annotated format (WSML, Web Service Modeling Language), in order to discover the most appropriate IaaS provided through a matchmaking process. Finally, an SLA is established and a 3rd-party SLA manager component is responsible to monitor and verify respective compliance. The matching module consists of the ontologies and a matchmaking algorithm. Two main ontologies are utilized through the process, namely the requirements ontology and the virtual unit ontology. The former ontology captures

the requestor's virtual unit requirements, which are defined as functional properties (e.g., number of CPUs, memory size) and non-functional properties (e.g., budget, location) that represent QoS requirements. Virtual unit ontology provides an abstract model for describing virtual units and their capabilities to let IaaS providers advertise their services.

Tsai et al. [62] emphasize on migrating cloud applications between cloud platforms. A Service Oriented Cloud Computing Architecture (SOCCA) is proposed where cloud computing resources are componentized, standardized and combined in order to build a "cross-platform virtual computer". An ontology mapping layer is configured over these services as a means of masking the differences between cloud providers. Cloud brokers interact with the ontology mapping layer for deploying applications in one cloud or another depending on a series of parameters, such as the budget, SLAs and QoS requirements that are negotiated with each provider. SOCCA applications can be developed using the standard interfaces provided by the architecture or the platform unique APIs of a cloud provider. Compared to PaaSport, SOCCA uses the ontology as an abstraction level for different could APIs rather than service discovery and selection.

SMICloud [29] is a framework trying to meet similar requirements to PaaSport, not taking into account though semantic similarities between different cloud providers and concepts. It is based on the Service Measurement Index (SMI) identified by the Cloud Service Measurement Index Consortium (CSMIC) [11] and proposes SMICloud, a framework that can compare different Cloud providers based on user requirements. A vital step is the Analytical Hierarchical Process (AHP), based on which the ranking mechanism is required to solve the problem of assigning weights to features considering the interdependence between them, thus providing a quantitative basis for ranking Cloud services. The SMI framework provides a holistic view for selecting a Cloud service provider based on accountability, agility, assurance of service, cost, performance, security, privacy and usability. The SMICloud framework provides features such as service selection based on QoS requirements and ranking of services based on previous user experiences and performance of services. Overall, Cloud computing services are evaluated based on qualitative and quantitative Key Performance Indicators (KPIs), i.e. service response time, interoperability, etc. Finally, the ranking system computes the relative ranking values of various Cloud services. The ranking system takes into account two parameters before deciding where to lease Cloud resources from: (a) the service quality ranking based on AHP, and, (b) the final ranking based on the cost and quality ranking.

Various European research projects deal with issues related to using semantics for PaaS portability and interoperability. The *Cloud4SOA* [12] project focuses on resolving interoperability and portability issues existing in current Cloud infrastructures and on introducing a user-centric approach for applications which are built upon and deployed using Cloud resources. Cloud4SOA semantically interconnects heterogeneous PaaS offerings across different Cloud providers that share the same technology. The design of the Cloud4SOA consists of a set of interlinked collaborating software components and models to provide developers and platform providers with a number of core capabilities: matchmaking, management, monitoring and migration of applications. Cloud4SOA focuses on resolving the semantic incompatibilities raised both within the same as well as across different Cloud PaaS systems and enable Cloud-based application development, deployment and migration across heterogeneous PaaS offerings. Cloud4SOA combines three complementary computing paradigms: Cloud computing, Service Oriented Architectures (SOA) and lightweight semantics. The Cloud4SOA Semantic Model consists of five tiers, covering the entire spectrum of fundamental Cloud entities and their relations: infrastructure (IaaS), platform (PaaS), application (SaaS), user and enterprise.

In this work, we have used the Cloud4SOA ontology [34] as the basic knowledge source for capturing the necessary concepts, entities and relationships for Cloud computing, focusing mainly on the PaaS layer and secondarily on the IaaS and SaaS layers (see section 4.1). As already discussed in the introductory section, the main focus of PaaSport is to build a PaaS offerings marketplace where app developers would be able to select the platform offering that best matches the application requirements; thus the focus of the PaaSport semantic models also lies in this direction, i.e. how to best serve semantic matchmaking and ranking of offerings. The Cloud4SOA ontology has made fixed assumptions about the characteristics and parameters needed for matching an application to an offering, and furthermore, had not represented at all how the values of parameters are compared between a request and an offering. For example, when a requirement states that the offering's network latency should be 5ms, it means 5 or less, whereas memory capacity 2GB means at least 2. The Cloud4SOA ontology represents and tests all these requirements as exact matches, whereas in PaaSport ontology we are able to

represent various range matches, including the ability to describe various units.

In order to do so, we have combined concepts of the Cloud4SOA ontology with the upper ontology DUL design pattern [26] in order to provide a seamlessly extensible semantic model able to describe any current and future parameter for PaaS without the need to reconstruct the core modeling structures of the ontology and the matchmaking algorithms (see next subsection).

The *mOSAIC* project [41] aimed at creating, promoting and exploiting an open-source Cloud application programming interface and a platform targeted for developing multi-Cloud oriented applications. An additional key goal was to ensure transparent and simple access to heterogeneous Cloud computing resources and to avoid proprietary solutions. Furthermore, it aimed to improve interoperability among existing Cloud solutions, platforms and services, both from the application-developer and the application-user perspectives. Within mOSAIC, semantic techniques are used for describing application requirements. The *Semantic Engine* component of mOSAIC infers the infrastructural requirements from the application description and produces a vendor agnostic SLA template [50]. The *Semantic Engine* helps users in selecting APIs components and functionalities needed for building new Cloud applications as well as in identifying the proper Cloud resources to be consumed. It introduces a new level of abstraction over the Cloud APIs, by providing semantic based representation of functionalities and resources, related by properties and constraints. Using the *Semantic Engine*, the developer of Cloud applications can semantically describe and annotate the developed components, specify application domain related concepts and application patterns. The mOSAIC's Cloud ontology [21] (developed in OWL) describes services and their wrapped interfaces and consists of 15 different base classes. It is built upon existing standards and proposals analysis through annotation of documents and it is used in the mOSAIC's semantic processing. It has been populated with instances of Cloud provider APIs and services specific terms. The underlying platform provides utilities in order to facilitate interoperability among different Cloud services, portability of the developed services on different platforms, intelligent discovery of services, service composition and management of SLAs.

REMICS [53] and *ARTIST* [4] are two projects focusing on the migration of legacy systems (e.g. banking applications written in Cobol) to the Cloud. REMICS focusses on the technical migration of such applications. It applies model-driven techniques to recover the legacy system into UML models, and then transforms these UML models into SOA models that can be deployed in a Cloud setting, and continuously evolved later on. ARTIST reuses some of the REMICS results and additionally focusses on the business aspect of the migration, i.e. how to also modernize the business models of SME and companies migrating to the Cloud. The main use of semantics in these two projects is to identify semantic differences between behavioral semantic model specifications in order to manage software evolution [37]. REMICS focuses on model-driven interoperability to facilitate the replacement of a migrated service with another service in case of service failure and recovery. However, in REMICS, the decision related to the replacement of a service by another one is fully left to the designer while PaaSport uses the Recommendation layer to recommend replacement. However, notice that there is another big difference between REMICS and PaaSport; REMICS deals with behavioral semantics of all services / components of an application, whereas PaaSport deals with the semantics of the computing platform on which the application will run and cares only on capacity features of the offered services, such as resources (quality, quantity), performance, and functionality (e.g. database systems, programming language version compatibilities, etc).

3 The PaaSport Marketplace

The PaaSport project focuses on resolving cloud platform interoperability and cloud application portability issues that exist in the Cloud PaaS market through a flexible and efficient deployment and migration approach. To this end, PaaSport combines Cloud PaaS technologies with lightweight semantics in order to specify and deliver a thin, non-intrusive Cloud broker (in the form of a Cloud PaaS Marketplace), to implement the enabling tools and technologies, and to deploy fully operational prototypes.

PaaSport aims to enable Cloud vendors to roll out semantically interoperable PaaS offerings leveraging their competitive advantage and the quality of service delivered to their customers, making their offerings more appealing and improving their outreach to potential customers. PaaSport also aims to facilitate Software SMEs to deploy business applications on the best-matching Cloud PaaS and to seamlessly migrate these applications on demand. Therefore, PaaSport aims to aligning and interconnecting heterogeneous

PaaS offerings, overcoming the vendor lock-in problem and lowering switching costs.

In the following subsections, we first discuss the stakeholders and requirements and, then, some use cases that are most relevant to the scope of this paper, namely the use of semantics in the PaaSport Marketplace. Next, we present the architecture of the PaaSport Marketplace infrastructure and, finally, we detail on the functionality of the semantic models in PaaSport.

3.1 Semantic Model Requirements

This subsection describes the stakeholders and requirements that drove the development of the PaaSport semantic models. These stakeholders had been identified in PaaSport project deliverable D1.1 [45]. The stakeholders involved in semantic modelling are as follows:

- **DevOps Engineer**: A solution architect seeking an optimal PaaS platform to develop, deploy at, or migrate to, a complex Cloud application. From the viewpoint of the semantic models, the most important optimization criteria for the search and decision-making process are the technical requirements for (or the capabilities of) the platform, regarding both services and resources offered, as well as the SLAs, i.e. the Quality-of-Service (QoS) characteristics of the platform and its services.
- **PaaS Provider**: An enterprise whose business model includes the delivery and operation of one or more PaaS solutions. A PaaS provider defines the technical aspects, the pricing models, reference values for quality of service parameters, and terms and conditions that apply to their offerings.
- **PaaSport Broker Administrator**: An individual assigned to the operation, maintenance, and management of the PaaSport Cloud broker and marketplace system.
- **Service (or PaaS) consumer**: An individual or an enterprise who uses the application (deployed by the DevOps Engineer) on the platform offering. Service consumers are usually concerned only with SLAs, i.e. how the application is delivered through the platform.

PaaS providers supply the Cloud-based application developers with the available PaaS offerings. The DevOps Engineers build applications that will be deployed and executed on PaaS offerings (platform services). The DevOps Engineers search for PaaS offerings that satisfy their applications' requirements. After a successful negotiation with a PaaS provider, the DevOps Engineer deploys the applications on the PaaS offering, whereas the service (or PaaS) consumer uses the application on the PaaS offering. An application can vary from a simple service, such as a relational database management system or a lightweight Web application, to a heavy software system, e.g. an ERP or a CRM.

Table 1. PaaSport functional requirements relevant to the Semantic Models.

Stakeholder	Functional Requirements Description	Semantic Models
DevOps Engineer	Enable transparent migration of data/applications (portability).	Application Offering
DevOps Engineer	Be able to manage instances across multiple Cloud providers.	All
DevOps Engineer	Enable use of metadata in the declaration of PaaS providers and during matchmaking.	All
DevOps Engineer	Support PaaS offerings with elasticity features.	All
DevOps Engineer	Be able to search for PaaS services that are held.	Offering Application
DevOps Engineer	Be able to support a marketplace (application selling business model, SLA adaptation and support, service billing policy).	All
DevOps Engineer	Be able to support self-service provisioning and management.	Application
DevOps Engineer	Be able to get recommendations in selecting a Cloud provider based on a hybrid recommender system approach.	All
DevOps Engineer	Be able to manage the geographic region in which an application is deployed.	All
PaaS Provider	Be able to publish service offerings in a service catalogue (service characteristics, policies, application platform availability and performance)	Offering SLA
PaaS Provider	Manage the SLA contracts	SLA

Table 2. PaaSport non-functional requirements relevant to the Semantic Models.

Non-functional Requirements Description
Supporting abstraction (hide many details of system and application infrastructure from developers and their applications).
Uniform service description (SLA offering), using standard formats.
SLAs with clear policies and guidelines for maintenance and version management of the platform and policies for version compatibility for APIs between the platform and the application.

Table 3. Requirements that involve the PaaSport Semantic Models.

Ontological Requirements Description	Requirement Type	Fulfilment
The ontology should be semantically interoperable or interoperable-ready with similar ontologies	Interoperability	By using an upper ontology (DOLCE+DnS Ultralite) we facilitate semantic interoperability at the conceptual level.
The ontology should be interoperable with industry standard models of Cloud platforms and applications.	Interoperability	In PaaSport project deliverable D1.3 [47] we describe how the ontology is aligned with the CAMP metamodel [10]. Furthermore, the TOSCA notion of software dependencies between application requirements and service capabilities has been used for the PaaSport offering and application semantic models.
Characteristics and properties of the ontology should be general enough to cover any platform offering and Cloud application.	Usability (Extensibility)	We have studied several Cloud platform offerings for identifying common features (Section 4.1). We have used the Cloud4SOA ontology [34] as inspiration. The PaaSport semantic model is the union of all offerings models.
The semantic models should be easily extensible and modular. It should be easy to add new characteristics and properties, without requiring the adaptation of existing ones.	Usability (Extensibility)	The DUL upper-ontology that was used to build the PaaSport ontologies ensures extensibility and modularity since it represents offering parameters detached from the characteristics they characterize (Section 4.2).
The ontology should support efficient and scalable reasoning and processing with regard to the recommendation algorithm of offerings.	Efficiency - Scalability	In [5] we describe that the recommendation algorithm has linear complexity to the number of instances and parameters.
The ontology should interoperate easily with other system components, especially with the Persistence Storage of the PaaSPort platform.	Usability (Operability)	The Persistence layer of PaaSport uses a relational database which is mapped to the RDF data model using the D2RQ platform (Section 5). In this way data is kept at a single place (no need for DB synchronization).
The ontology should support the representation of concepts relevant to the PaaSport domain, e.g. semantic annotation of offerings and applications for the purposes of recommendation.	Usability (Understandability)	We use a core ontological model for characteristics and parameters that affect all the semantic models (offerings, application, SLA models). In this way matchmaking and selection can be easily performed both syntactically and semantically

Table 1 shows the most important functional requirements of the PaaSport Marketplace that affect also the Semantic Models. Specifically, we show the specific sub-model each requirement involves, namely Offering, Application or SLA. Furthermore, **Table 2** presents the most important non-functional requirements of the PaaSport Marketplace relevant to the Semantic Models that have to do mostly with the issue of interoperability. Finally, **Table 3** presents requirements related to the Semantic Models / Ontologies themselves, as well as the recommendation algorithm. These requirements have been identified by analyzing the functionality of the Semantic Models in the PaaSport project.

3.2 PaaSport Uses Cases relevant to the use of the Semantic Models

The PaaSport use cases [46] most relevant to the use of semantics are the following:

- Manage semantic profile of application
- Search PaaS offering
- Manage PaaS offering

For the sake of space, here we will only present a fusion of the first two use cases. Let a DevOps engineer initiate a search for appropriate PaaS offerings that meet the functional and non-functional requirements that his/her company's application imposes on the platform, in order to select one of the offerings to deploy the application. The course of actions by the system and the user (DevOps engineer) are the following:

1. The DevOps Engineer initiates the search for PaaS offerings.
2. The PaaSport marketplace displays two options to the user, a semantic search based on an application semantic profile and a search based on a syntactic matching of manually entered search criteria.

3. The DevOps Engineer chooses the semantic search option.
4. The system presents the user a list of all application sematic profiles that have been created by the user.
   4.1. The user cannot find an appropriate semantic application profile, so he/she decides to create a new one.
      4.1.1. The system creates an empty semantic profile and presents a form to the user that allows him to edit the semantic description of his application.
      4.1.2. The user fills the form with the semantic description of his application requirements (functional and non-functional) and finally stores it.
      4.1.3. The system successfully validates the entries of the user and stores the application semantic profile. After storing, it presents an option to the user that allows him to initiate a search for a suitable PaaS offering based on the created profile.
      4.1.4. The system continues with step 4.
5. The user selects the application semantic profile to be used as search criteria.
6. The system identifies the PaaS offerings that match the search criteria (functional application requirements).
7. For each PaaS offering that matches the search criteria, the system instantiates and presents SLA offers.
8. The system presents a list of all matching PaaS offerings together with the corresponding SLA offer to the user. The list is ordered according to the user's preferences (non-functional application requirements).
9. The user selects the PaaS offering from the list that optimally meets his/her requirements.
   9.1. The DevOps Engineer selects a PaaS offering from the list and initiates the process of viewing the detailed description of the PaaS provider to inform himself about the provider's information such as pricing and offering ratings.
   9.2. The system loads and presents the detailed description of the selected PaaS provider. It embeds pricing and rating information about the provider and the PaaS offering.
   9.3. The DevOps Engineer is unconvinced of the PaaS offering due to i.e. bad ratings or provider statistics.
   9.4. The user repeats step 9 until he finds a suitable PaaS offering.
10. The user initiates the application deployment process on the selected platform at step 9.

3.3 Architecture of the PaaSport Marketplace Infrastructure

The PaaSport Architecture (**Figure 1**) constitutes a thin, non-intrusive broker and marketplace that mediates between competing or even collaborating PaaS offerings [46]. It relies on open standards and introduces a scalable, reusable, modular, extendable and transferable approach for facilitating the deployment and execution of resource intensive business services on top of semantically enhanced Cloud PaaS offerings.

It comprises of the following five artefacts:

- The Adaptive Front-ends that support seamless interaction between the users and the PaaSport functionalities, through a set of configurable utilities that are adapted to the user's context;
- The PaaSport Semantic Models that serve as the conceptual and modelling pillars of the marketplace infrastructure, for the annotation of the registered PaaS offerings and the deployed applications profiles;
- The PaaS Offering Recommendation Layer that implements the core functionalities offered by the PaaSport Marketplace Infrastructure, such as PaaS offering discovery, recommendation and rating;
- The Monitoring and SLA Enforcement Layer that realizes the monitoring of the deployed business applications and the corresponding Service Level agreement;
- The Persistence, Execution and Coordination Layer that puts in place the technical infrastructure, e.g. repositories, on top of which the PaaSport marketplace is built, including also the PaaSport Unified PaaS API that is a common API exploited in order to uniformly interact with the heterogeneous PaaS offerings and, in addition, it realize the lifecycle management of the deployed applications.

Our focus in this paper is to present thoroughly the use of semantics in the PaaSport marketplace, as discussed in the next subsection. Besides the PaaSport Semantics models, the architectural layers that mainly deal with semantics are the Offering Recommendation and the Persistence layers.

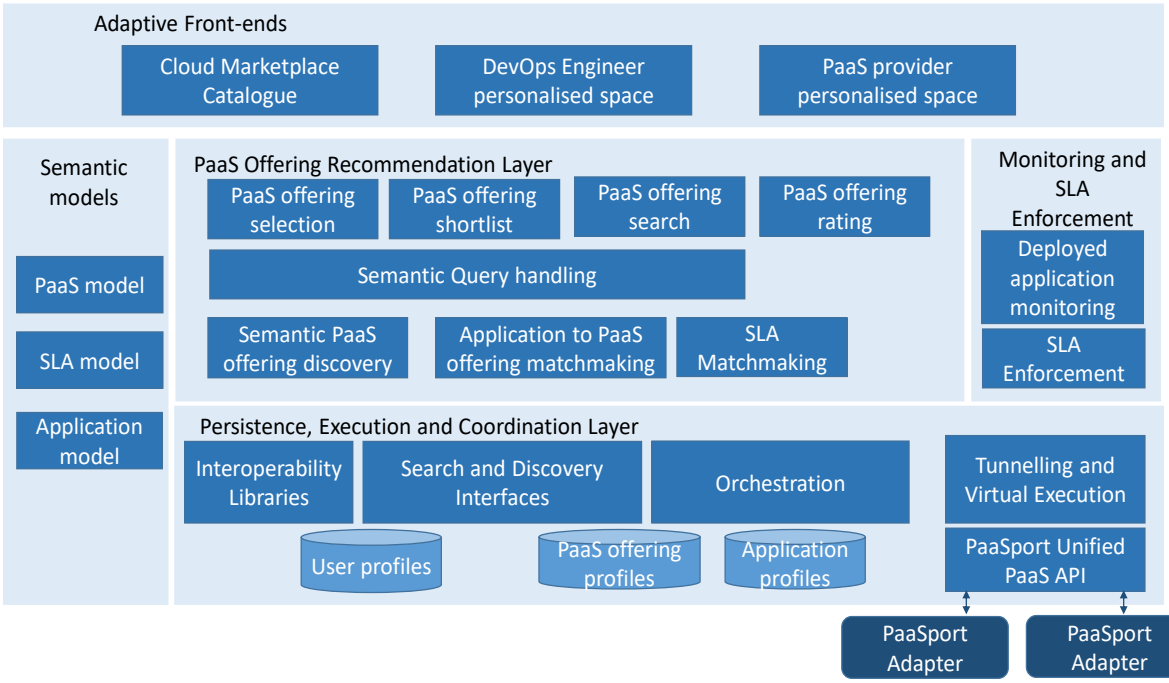

Figure 1. High-level view of the PaaSport Cloud broker Architecture [46].

3.4 The Functionality of Semantic Models in the PaaSport Marketplace

The PaaSport Semantic Models constitute the conceptual and modelling backbone of the marketplace infrastructure and they are used in order to provide a semantic annotation means for the registered PaaS offerings and the deployed applications profiles. Specifically, the functionalities of the PaaSport Semantic Models in the various modules of the PaaSport platform are the following:

a) They provide a common vocabulary for the various modules of the system and for aligning the models of different PaaS offerings, thus resolving semantic interoperability conflicts among heterogeneous Cloud PaaS offerings that exploit diverse platform and application models, offered service descriptions, offered resources, Quality of Service, SLA formats, billing policies and other important issues (such as location of service or service certifications); and,

b) Concerning the PaaSport Offering Recommendation layer, the Semantic Models bridge the gap between business application requirements and PaaS offerings capabilities, thus, facilitating the matchmaking and the identification of the specific PaaS that fulfills the business and technical requirements of a particular application.

c) Concerning the Persistence layer, the database schema follows exactly the conceptual model of the ontologies, in order to avoid syntactic/semantic mismatch between tables/concepts and attributes/properties when the Offering Recommendation layer retrieves PaaS offerings from the database, based on the Semantic Models.

d) Concerning the Adaptive Front-ends layer, the UIs for managing application and PaaS offering semantic profiles use concepts and properties from the semantic models.

e) Concerning the Monitoring and SLA Enforcement layer, the SLA model has been considered for defining the PaaSport SLA policy model used by the SLA Enforcement component, while the monitoring system has been designed so as to support the metrics defined in the semantic SLA model.

The first functionality is achieved by: a) considering and fusing together existing approaches to Cloud computing semantic models, b) studying existing

Cloud computing platforms, c) taking into consideration the functional and non-functional user requirements, as identified in the next sub-section, and finally, d) considering well known ontology frameworks so that interoperability with other similar efforts outside the project boundaries can be achieved. All of these are elaborated in Section 4. The second functionality is achieved by using a common conceptual framework for describing both the platform capabilities and the application requirements, so that application profiles can be matched conceptually, structurally and quantitatively to platform offerings, as explained briefly in Section 5 and elaborate in [5]. The third functionality is achieved by mapping the PaaS Offering profiles stored in the database onto the Semantic Model layer (RDF graph) using a relational-to-ontology mapping tool, namely the D2RQ platform [15] (see section 5). The fourth and fifth functionalities are beyond the scope of this paper and their achievement can be found in the deliverables of the PaaSport project [49].

4 PaaSport Semantic Models

In this section, we describe the development of the PaaSport Semantic Models. Initially, we briefly present how we have acquired the knowledge to be modelled and then we analyze our modelling decisions and justify how they adhere to the ontology requirements set in Section 3.1. We finally present how the PaaSport ontology has been implemented using the descriptions and situations (DnS) ontology pattern of the DOLCE Ultra Lite (DUL) upper level ontology.

4.1 PaaS Domain Models

Before starting Ontology development, we have initially surveyed existing cloud computing platforms and PaaS providers and the respective technical background. We have distinguished several key cloud computing platform manufacturers and providers and

Table 4. Major Cloud Computing Platforms, supported programming languages and services.

Cloud Computing Platforms	Programming Languages	Services
OpenShift Origin	Ruby, Java, Node.js, Python, PHP, Vert.x, Perl	Tomcat (JBoss EWS), Jenkins, PostgreSQL, MySQL, MongoDB
Cloud Foundry	Java, PHP, Python, Play, Node.js, Ruby, Go	MySQL, PostgreSQL, MongoDB
ApacheStratos	Java, PHP	Tomcat, MySQL
HPE Helion Stackato	Clojure, Go, Groovy, Java, Node, Perl, PHP, Python, Ruby, Scala	Apache, JBoss, nginx, Tomcat, MySQL, MongoDB, PostgreSQL

Table 5. Key PaaS Providers and provided services.

PaaS Provider	Pricing Policy (plans)	SLA	Resources	Programming Languages	Services
OpenShift	Free/Bronze/Silver	✓	✓	Ruby, Java, Node.js, Python, PHP, Vert.x, Perl	Tomcat (JBoss EWS), Jenkins, PostgreSQL, MySQL, MongoDB
Heroku	Hobby/Standard/ Premium/Enterprise	✓	✓	Ruby, Java, Node.js, Scala, Clojure, Python, PHP, Perl	MySQL, PostgreSQL, Redis, MongoDB
Cloudbees	Free/Enterprise	✓	✓	Java, Ruby, Node.js, Clojure, PHP, Erlang, Scala	Tomcat, PostgreSQL, MongoDB
AppHarbor	CANOE/CATAMARAN/ YACHT	✓	✓	.NET	MySQL, SQL Server, PostgreSQL, Mongo
CloudControl	Developer/Startup/ Business/Business+	✓	✓	Java, PHP, Python, Ruby, Node.js	PostgreSQL, MySQL, MongoDB
Pivotal Cloud Foundry	Aggregated memory used by applications per month	✓	✓	Java, PHP, Python, Play, Node.js, Ruby, Go	MySQL, PostgreSQL, MongoDB
Amazon Elastic Beanstalk	Free trial and proportional price after that	✓	✓	Java, .NET, PHP, Node.js, Python, Ruby, Go	Tomcat, for database can install the instance on amazon cloud
IBM Bluemix	Free for small resources and proportional price after that	✓	✓	Java, JavaScript, go, PHP, python, ruby	Tomcat, MySQL, PostgreSQL, MongoDB

we tried to record common software / services[1], platform QoS and pricing policy (plans) that a cloud computing platform and/or a PaaS offering can provide. Notice that there are two types of PaaS providers. The first type, such as Heroku and Amazon, are based on their own proprietary cloud computing platform in order to deliver a single, public cloud. On the other hand, there are PaaS providers, such as OpenShift and Cloud Foundry, that they offer the cloud computing platform as an open-source software, so that several, both public and private clouds can be built on it. Moreover, these providers also offer their own public cloud, built around on their software, of course. **Table 4** reviews the most important cloud computing platforms and their supported programming languages and services, whereas **Table 5** reviews several key PaaS providers, either of the first or of the second type.

A PaaS typically resides on top of IaaS providing the ability to access remote computing resources. With IaaS there is the possibility of remotely controlling machines or virtual machines that can be used as necessary. Thus, it was decided to study the OpenShift origins and Cloud Foundry, two of the most popular PaaS platform systems. Cloud Foundry and OpenShift are quite similar in their capabilities and their approach to PaaS. While the terminology they use and the exact deployment methods differ, in essence they are very similar: Each delivers a platform based on the Linux OS with lightweight containers that can run applications against open source languages and frameworks, using common services (software), such as databases. This gives the possibility to describe general PaaS platform via our ontology.

Notice that in order to cover maximally all the platform-offering models that we have reviewed, the PaaSport semantic model is the union of the reviewed offering models, so that no provider can feel left out. This is important for reaching out the providers. In so doing, the PaaSport ontology has many detailed PaaS Characteristics and parameters, as presented in Section 4.3. This means that the providers whose offering model will not match the common "super"-model we have created, will leave many of the offering description characteristics and parameters blank. However, this will not affect the usefulness of the ontology during matchmaking, since it is the DevOps Engineer's request that will guide the search for appropriate platform offerings. Only the parameters filled-in by the DevOps Engineer with application requirements will be used for matching the offerings, regardless if the offerings have all or less-than-all parameters filled-in with values. For example, if the request looks for a general characteristic, e.g. storage up to 100GB, but not for a specific type of disk type (HDD or SSD), then both offerings with specified disk type and the ones with not such a specification, will be considered.

Furthermore, we have studied the semantic layer of Cloud4SOA [34] and considered many relevant concepts and entities from this project. More specifically, Cload4SOA's semantic layer describes some of the entities for a PaaS platform, featuring five (5) distinct layers, each of which describes one separate view of a PaaS platform:

- The **Infrastructure layer** contains definitions for classes used for capturing knowledge related to the infrastructure (hardware and software) utilized by the Platform and Application layers, as well as metrics to measure the values of hardware/software attributes.
- The **Platform layer** contains definitions for classes used for capturing knowledge related to a Cloud-based platform (e.g. supported programming language, offered software/hardware functionalities). The platform is based on the Infrastructure layer in order to operate.
- The **Application layer** contains definitions for classes used for capturing knowledge related to a Cloud-based Application. A Cloud-based Application is developed/deployed/managed in a Cloud Platform.
- The **Enterprise layer** contains definitions for classes used for capturing knowledge related to the enterprises involved in the Cloud (e.g. the PaaS provider, the IaaS provider) and their role in the Cloud.
- The **User layer** contains definitions for classes used for capturing knowledge related to the users of a Cloud4SOA platform. The latter are the Cloud-based application developers and the Cloud PaaS providers.

The Infrastructure layer is the basic layer of the Ontology; it provides a common terminology used by the Application and Platform layer enabling the matching between their instances. The Enterprise layer is correlated with the Platform and Infrastructure layer; it defines the enterprises responsible for the offering of Cloud Infrastructure and Cloud Platforms. Finally, the User layer is the topmost layer of the ontology, it defines the users of the Cloud4SOA platform that are the Cloud-based application developers (correlated with

[1] Notice that in our terminology the concept "service" is a synonym for "software". The services offered by a platform are the pre-installed applications (e.g. databases, web servers, etc.) that come along with the cloud platform.

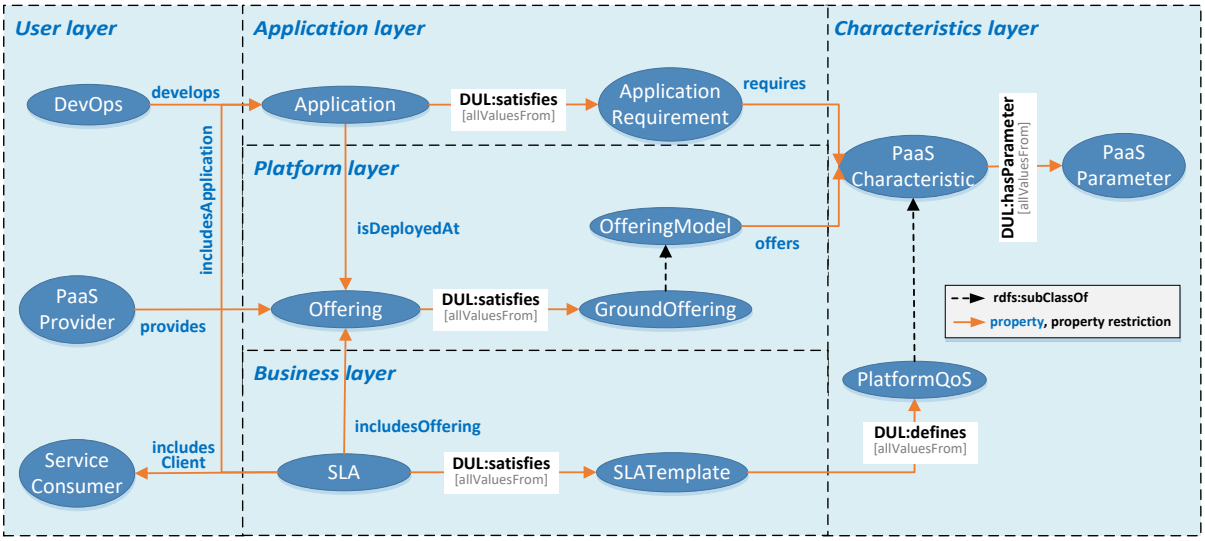

Figure 2. Overview of the PaaSport Semantic Model and its layers.

Application layer) and the Cloud PaaS providers (correlated with Platform layer), it is also correlated with the Enterprise layer since every involved enterprise can have a Cloud4SOA user account. Notice that these layers concern only the semantic representation of the various entities involved in the Cloud and they have nothing to do either with the architecture of the PaaSport broker (presented in Section 3) or the stakeholders of the PaaSport marketplace (described in Section 3.1).

In the PaaSport semantic model, we have followed an almost similar approach concerning the layers (**Figure 2**); however, we have mainly focused on concepts and entities from the first three layers (Infrastructure, Platform and Application), giving emphasis on the Infrastructure layer, since the semantic model of PaaSport is mainly concerned with semantic matchmaking between Cloud platform offering capabilities / characteristics and application requirements from the cloud platform [5].

More specifically, the PaaSport semantic models comprises of the following layers:

- **User layer**: contains definitions of classes about the agents (human or corporation) involved in the PaaSport marketplace, namely PaaS providers, DevOps engineers and service consumers.
- **Application layer**: contains definitions for classes related to an Application deployment at a Cloud Platform, including the set of application requirements from the platform.
- **Platform layer**: contains definitions for classes related to Cloud computing platforms and platform offerings.
- **Business layer**: contains definitions for classes related to the business aspects of the PaaSport marketplace, namely the SLA between marketplace stakeholders and its definition (SLA templates).
- **Characteristics layer**: contains definitions for classes related to the characteristics / capabilities of the platform offerings, namely the infrastructure (hardware and software) utilized by the Platform and Application layers, metrics to measure the values of hardware/software attributes and the platform's QoS, business characteristics of the platform, such as pricing policy and geographical location of services, user characteristics, such as ratings, etc.

After this research, we combined the knowledge of the real market PaaS providers and the existing Cloud4SOA ontology[2], in order to develop an ontology describing all possible components of a PaaS. In addition to the above, we have also used as guides for the development of the PaaSport ontology, some of the major standardization efforts for Cloud computing from OASIS, such as CAMP (Cloud Application

[2] https://github.com/Cloud4SOA/Cloud4SOA/tree/master/semanticModel/C4S_model

Management for Platforms) [10] and TOSCA (Topology and Orchestration Specification for Cloud Applications) [59]. Actually, in [47] we have developed an OWL version of the CAMP meta-model and we have aligned it with the PaaSport ontology. However, this is beyond the scope of this paper.

4.2 PaaSport Semantic Modelling Approach

In the PaaSport project, we have decided to develop the PaaSport semantic models (**Figure 2**) as an extension of the DOLCE+DnS Ultralite (DUL) ontology, which is a simplification and an improvement of some parts of DOLCE Lite-Plus library and Descriptions and Situations ontology (see subsection 4.2.1). The main reasons we have followed this approach is the fact that DUL is based on Ontology Design Patterns (ODP) ensuring a high degree of reusability, modularity and extensibility [24].

The use of DUL (i.e. an upper ontology) ensures better semantic interoperability with other similar projects and research efforts. This is because upper ontologies are supposed to be domain-independent, encompassing very general concepts. Thus, when two ontologies belonging to similar domains are ranked as specializations of the same upper ontology, then most classes of similar meanings belonging to the two different ontologies will be classified under the same general concepts of the common upper ontology. In this way, semantic interoperability is achieved, because even if the two classes are not commonly understood, they could propagate their instances to the common general superclass, thus a minimum level of common understanding is guaranteed. Although not yet a standard, DUL has been used in many projects and offers a very flexible design pattern for defining domain-dependent ontologies.

The DUL ontology is easy to extend by adding e.g. new characteristics and parameters related to PaaSport offerings and applications. Ontologies in RDFS and OWL are generally easy to extend, since new classes and properties can be easily added, without the need to re-configure existing class definitions, since properties are first-class citizens/objects of the ontology. However, the PaaSport semantic model must co-exist and interoperate with the Persistence Storage of the main system. In this case, usually tables correspond to classes and attributes to properties. However, adding a new property to a class roughly corresponds to adding a new attribute to an existing table. This requires schema re-definition. Therefore, extensibility must be exercised very cautiously.

DUL follows a different approach. Properties (i.e. parameters) are defined as new classes and not as OWL properties. In this way, the introduction of new properties/parameters does not require the disturbance of the schema of existing tables but merely the introduction of new tables. This greatly favors extensibility.

Furthermore, this representation of properties also favors the generality and extensibility of the matchmaking / ranking algorithm between application requirements and platform offerings. This type of extensibility is briefly explained in section 5.2 and analyzed more thoroughly in [5].

In **Table 3** (Section 3.1) we briefly explain how the identified ontological requirements have been fulfilled by the above contributions. Notice that some of the requirements are fulfilled by the recommendation algorithm and the integration of the semantic models into the persistence layer of the PaaSport Marketplace, that are presented in sections 4 and 5.

4.2.1 DOLCE and DnS

The **Descriptive Ontology for Linguistic and Cognitive Engineering** (**DOLCE**) aims at capturing the ontological categories underlying natural language and human common sense. **DnS** (**Descriptions and Situations**), is a constructivist ontology that pushes DOLCE's descriptive stance even further [26]. DnS does not put restrictions on the type of entities and relations that one may want to postulate, either as a domain specification, or as an upper ontology, and it allows for context-sensitive "redescriptions" of the types and relations postulated by other given ontologies (or 'ground' vocabularies).

The current OWL encoding of DnS assumes DOLCE as a ground top-level vocabulary. In fact, the two ontologies combined have been deployed for various modelling purposes devoted to the treatment of social entities, such as e.g. organizations, collectives, plans, norms, and information objects. A lighter OWL axiomatization of DOLCE and DnS is available as **DOLCE+DnS-Ultralite** (**DUL**)[3]. This lighter version (see Figure 3) simplifies the names of many classes and properties, adds extensive inline comments, thoroughly aligns to the repository of Content patterns and greatly speeds up consistency checking and classification of OWL domain ontologies that are plugged to it.

The core model of the PaaSport ontology has been developed as a specialized instantiation of the DnS de-

[3] http://www.ontologydesignpatterns.org/ont/dul/DUL.owl

sign pattern. The DnS pattern provides a principled approach to context reification through a clear separation of states-of-affairs, i.e. a set of assertions, and their interpretation based on a non-physical context, called a "**description**". Intuitively, DnS axioms try to capture the notion of *situation* as a unitarian entity out of a state of affairs, with the unity criterion being provided by a *description*. In that way, when a description is applied to a state of affairs, a situation emerges. The core-modeling pattern of DnS allows the representation of the following conceptualisations, as illustrated in Figure 3:

- **Situations**. A situation defines a set of domain entities that are involved in a specific pattern instantiation ("isSettingFor" property) and they are interpreted on the basis of a "description" (through the "satisfies" property). Each situation is also correlated with one user/agent ("dul:includesAgent" property).
- **Descriptions**. An activity description serves as the descriptive context of a situation, defining the concepts ("defines" property) that classify the domain entities of a specific pattern instantiation, creating views on situations.
- **Concepts**. The DUL concepts classify domain entities describing the way they should be interpreted in a particular situation. Each concept may refer to one or more parameters, allowing the enrichment of concepts with additional descriptive context.

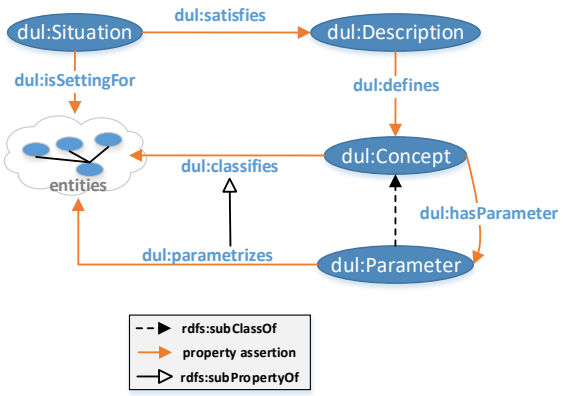

Figure 3. DUL overview.

As we describe in subsection 4.3, the PaaSport semantic models have been defined as extensions to the core DnS pattern (**Figure 4**). More specifically, the situation concept is used as a container for defining the higher-level conceptualizations of the PaaSport domain, such as PaaS offerings, application deployments and SLAs. Offerings are considered as situations, since they represent (possibly different) "views" of a cloud platform, related to (possibly different) provider(s). The SLA is a relational situation between a provider and a client, involving an application that is deployed on a platform offering. Finally, applications are situations as well, since in PaaSport semantic model they actually denote application deployments on a specific platform that may change in the future, and not the application code itself, which might endure.

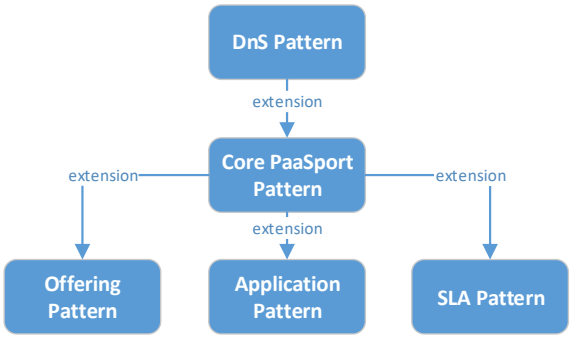

Figure 4. DnS pattern.

The specialized situations are further correlated with specializations of the description concept, allowing the correlation of the situations with additional descriptive context (**Figure 5**). The descriptions in turn define one or more concepts with appropriate domain parameters, resulting in a rich, dynamic and flexible modelling pattern able to address the domain modelling requirements identified so far.

4.3 The PaaSport Ontology

In the following, we describe the core modelling patterns we have developed that serve as the conceptual bases for modelling PaaSport-related concepts. More specifically, the proposed patterns implement the DnS ontology pattern of DOLCE Ultra Lite (DUL) ontology that provides a formal modelling basis and has been used for a number of core ontologies (e.g. the Semantic Sensor Network – SSN ontology [14]), while the pattern-oriented approach of DUL provides native support for modularization and extension by domain specific ontologies. Actually, the "Offering Pattern" is represented through the combination of the OfferingModel and GroundOffering classes, the "Application Pattern" is represented by the ApplicationRequirement class, and finally, the SLA Pattern is represented by the SLATemplate class.

Figure 5 displays the associations between DUL and the PaaSport semantic models. Applications, Offerings and SLAa are PaaS situations (subclass of SUL situation), which satisfy the corresponding PaaS descriptions, namely Application Requirement, Ground Offering and SLA Template. PaaS Characteristics specialize DUL concepts and are defined in PaaS descriptions. The "defines" property is specialized as "requires" for application requirements and as "offers" for PaaS offerings. SLA templates define only platform QoS parameters, aggregated as a PlatformQoS characteristic.

The extensibility of the PaaSport semantic model depends on the parameters' definition. A situation (offering, application, SLA agreement) can have a set of characteristics (description). A characteristic (for example PlatformQoS) can have one or more parameters. Every parameter has a value (dul:hasParameterDataValue) and a "quality" (dul:parametrizes), which is the "physical" or "logical" dimension of the parameter (e.g. storage, duration, etc.) and it is usually (not always) accompanied by measurement units. For this reason, if we know the quality of the value we can easy compare two parameters and by extension two characteristics. If we want to add a new parameter to a characteristic, we have only to declare the quality value of this parameter. Moreover, we could do the same for characteristics, i.e. we can just add a new characteristic to the corresponding description. Furthermore, if we want new characteristic and parameter types, we just add new classes in the corresponding hierarchies, without the need to redefine any ontology properties, since the "requires", "offers" and "hasParameter" properties cover all (sub)classes of PaaS Characteristics and Parameters.

The application example in **Figure 6** describes the logic underlying our proposed model. Specifically, we provide an application which is linked to two different sets of requirements. This is allowed by the PaaSport Semantic Model and its rationale lies on the fact that there might be different sets of requirements e.g. for minimum, moderate or optimal performance for the same application. The first set of requirements is about the programming language (Java ver. 1.6), the database (MySQL with minimum 1GB size for storage) and the platform's QoS, with maximum (network) latency 5ms and minimum uptime 99%. So, according to the DUL pattern, an "Application" (my_app1) "satisfies" an "Application Requirement" (my_app1_req1) which consists of a set of PaaS Characteristics that the "Application" "requires", such as "Programming Language" (Java_1.6.0), "Services" (MySQL_1GB), "PlatformQoS" (qos_1_1), etc. Each PaaS Characteristic consists of one or more PaaS parameters, such as name or version of the programming language, service type (SQLService), DB capacity (Capacity1), etc. The "PlatformQoS" characteristic has several "QoS Parameters", such as service uptime (Uptime99), network latency (Latency5), etc. Each parameter is linked to ("parametrizes") a quality value (MaxMinGB, Maxms) that contains the actual parameter value

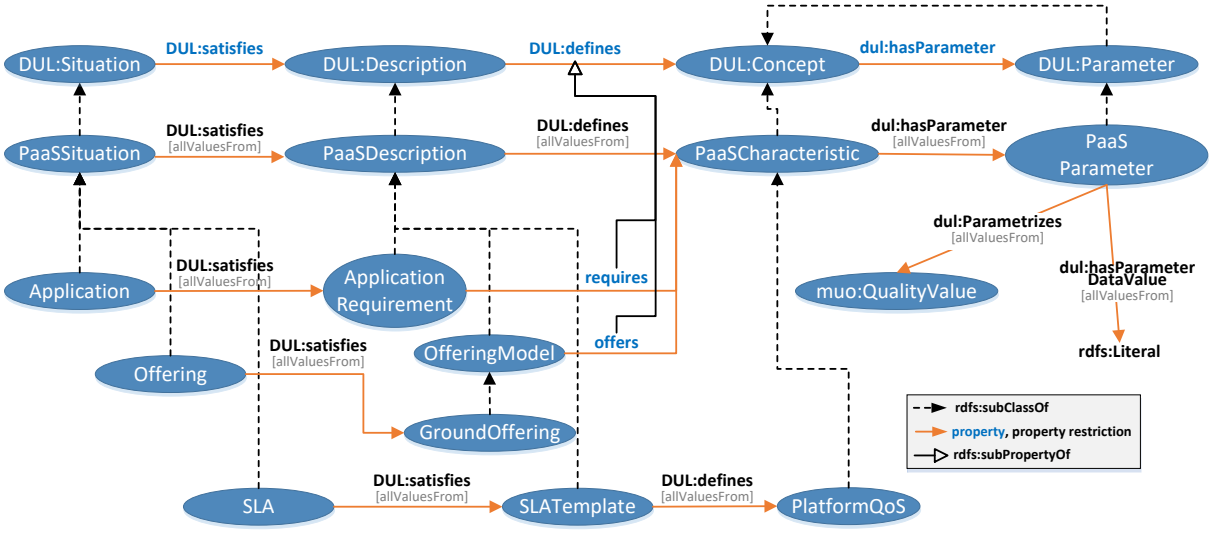

Figure 5. Overview of DUL and PaaSport associations.

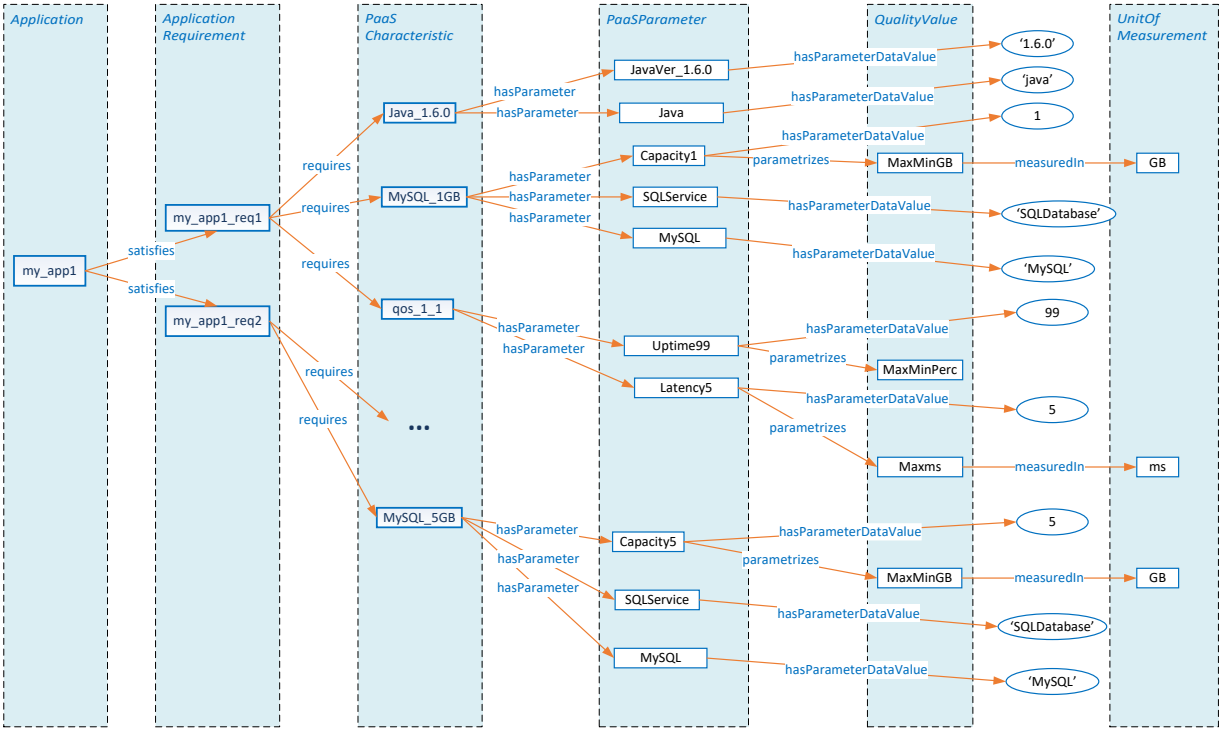

Figure 6. An Application example.

(hasParameterDataValue) and optionally a measurement unit (e.g. "measuredIn" millisecond, GB, etc). The quality values belong to different types (such as single, min, max, etc.) that define how the application requirement value should be compared to the correspondig offering value (exact match, greater than, less than, etc.). More on this are discussed in section 4.3.5. In **Figure 6**, for example, the platform's QoS parameter network latency is shown, which belongs to the Max quality value type, meaning that the application requires a maximum acceptable limit 5ms for the network latency. Notice that the PaaSport semantic model also allows for QoS parameters of individual software/services provided by the infrastructure of the platform through the paas:ResourceParameter class (see Section 4.3.4).

The Offering and Application models are not very different. Only the top-level classes are different, since Offerings and Applications stand conceptually for two very different types of entities: cloud platforms and cloud application deployments, respectively. However, both use the same vocabulary for describing services (PaaS Characteristics) and their parameters, since the restrictions on the dul:defines property are inherited by all subclasses of PaaSDescription, as shown in **Figure 5**. This allows the recommendation algorithm to seamlessly match application requirements to PaaS offerings both syntactically and semantically, which is briefly described in section 5.2. For a full account of the recommendation algorithm see [5].

Actually, PaaSport semantic models follow the "Requirements and Capabilities" model of TOSCA [59] or the "Requirements and Characteristics" model of CAMP [10], which allow for expressing requirements and capabilities of components of a service or an application. In this way, PaaSport semantic models can represent dependencies between components, i.e. when a component depends on (requires) a feature provided by another component, or that a component has certain requirements against the hosting environment such as for the allocation of certain resources. More specifically, the offering model describes the capabilities of the services offered by a certain platform, whereas the application model describes the service, resource or QoS requirements that an application has on the platform, in order to be deployed on it. The SLA model is concerned just with the QoS characteristics of the platform.

In what follows, we first give a brief description of the three PaaSport Semantic Models, i.e. Offering, Application and SLA, and then we describe in detail the PaaS Characteristics and parameters, which is the

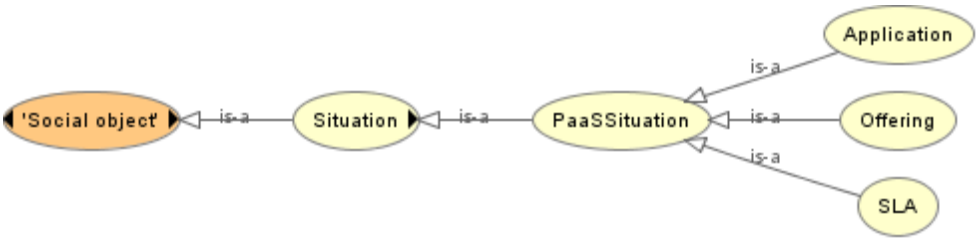

Figure 7. PaaS situations hierarchy.

core part of the ontology that bridges all the semantic models, though the use of common characteristics and parameters for the platform offering capabilities and the application profile requirements.

4.3.1 Offering Model

The Offering Model helps PaaS providers semantically describe their PaaS offerings. Specifically, it contains all available capabilities of a PaaS offering and its services. These capabilities are part of the PaaS hierarchy of characteristics (see section 4.3.4) and can be technical, performance-related, geographical etc.

Figure 7 shows the hierarchy for the Offering class. PaaSSituation is an abstract superclass for all Situation entities of PaaSport, namely Offering, Application and SLA (also shown in **Figure 5**). The core entities of the offering model are the PaaS Offering and its description. The Offering class represents (and is related to) a grounded PaaS Offering. The set of the characteristics / capabilities offered by the platform (e.g. services, resources, platform QoS, etc.), can be found by following the "satisfies" property assertions (**Figure 8**). An offering is linked to its provider as well as to its deployed applications. Notice that although the word "provides" usually cannot be associated with the word "offering", but rather with the word "service", in PaaSport semantic model, class "Offering" actually refers to a "PaaS offering"; therefore, it represents a service.

We define two classes of the offering description:

a) The **OfferingModel** represents the description of a cloud computing platform, i.e. a set of software and tools that can be downloaded and installed on a private or a public cloud, e.g. "OpenShift Origin". The offering model description describes general capabilities of a platform, such as supported programming languages, databases, servers, etc. These capabilities are common in every installation (instantiation) of the PaaS offering model. It is linked through the "offers" property to characteristics describing the capabilities of the offering. Actually, an offering model consists only of the ProgrammingEnvironment and Service characteristics. Characteristics like offered resources and platform QoS are not part of an offering model but of a GroundOffering. In

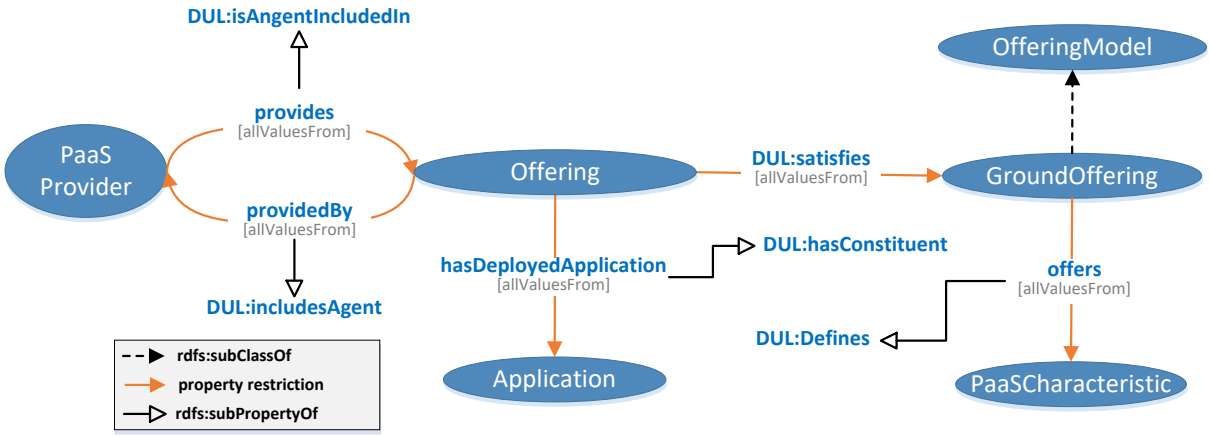

Figure 8. Offering overview.

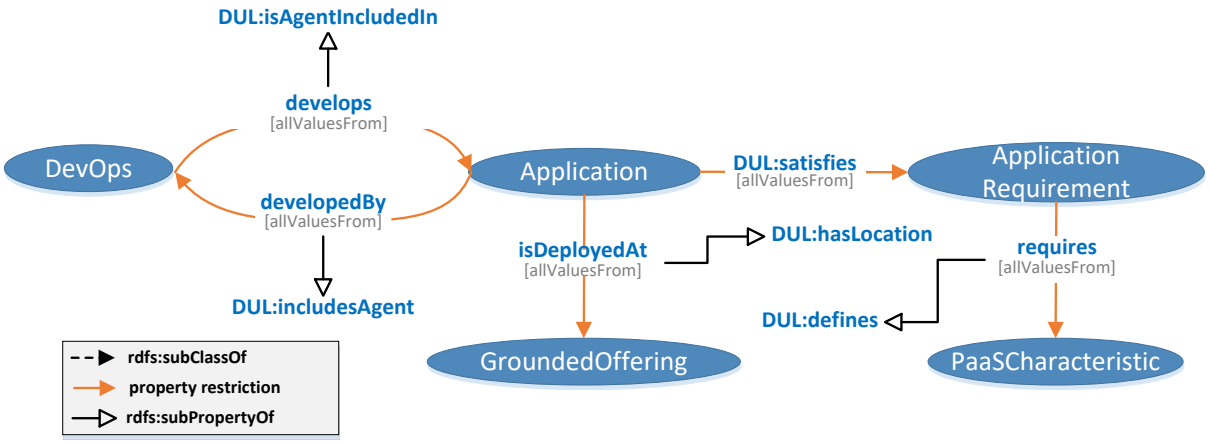

Figure 9. Application overview.

practice, when a user adds a new cloud computing platform in the PaaSport marketplace he/she will first create an instance of this class.

b) The **GroundOffering** represents the description of a grounded PaaS offering and consists of all PaaS capabilities and/or characteristics, such as programming environment, services, resources, QoS, location, pricing etc. The resource, platform QoS, Pricing and location characteristics are defined when an Offering Model becomes a GroundOffering (necessary and sufficient conditions). In practice, when a user adds a new PaaS offering in the PaaSport marketplace he/she will "ground" an existing offering model by creating a new instance of the GroundOffering class; all the characteristics of the corresponding model (service and programming environment) will be copied to the new instance and the capabilities of the grounded offering, such as resources, platform QoS, etc., will be added as well. For example, RedHat's OpenShift[4] is a GroundOffering of the "OpenShift Origin"[5] OfferingModel. There can be several such groundings, especially for open source cloud computing platforms, whereas for several commercial PaaS, such as Heroku[6], there is only one grounding.

4.3.2 Application Model

The Application Model comprises definitions for classes that capture knowledge related to Cloud-based application requirements or software/resource dependencies on the hosting Cloud platform. We designed a simple, open and extendable vocabulary, which allows the semantic annotation of developers' application requirements. Based on the Application Model and the ontology of PaaS Characteristics and parameters (see section 4.3.4), the Cloud-based application developer creates and manages the semantic profile of his/her application deployments (**Figure 9**), regarding software, resource or platform QoS dependencies. The developer can define functional (software dependencies on programming language, servers, database, etc.) and non-functional (resource capacities, performance, price, etc.) requirements for his application. Specifically, these requirements refer to the deployment platform and allow developers to match the PaaS Offerings whose capabilities are the most relevant to their application requirements.

The main class is Application, which represents a Cloud-based application deployed at a PaaS Offering. The set of requirements for the Application from the platform can be found by following the "satisfies" property assertions. An application is linked to its developer. The ApplicationRequirement class is the description of an Application and consists of a set of application requirements, which are PaaS-related Characteristics that can be modeled using "requires" property assertions. Notice that requirements are complex entities, consisting of many parameters, e.g. a database requirement could be MySQL v. 5.7 with minimum storage capacity 5GB.

[4] https://www.openshift.com/
[5] https://www.openshift.org/
[6] https://www.heroku.com/

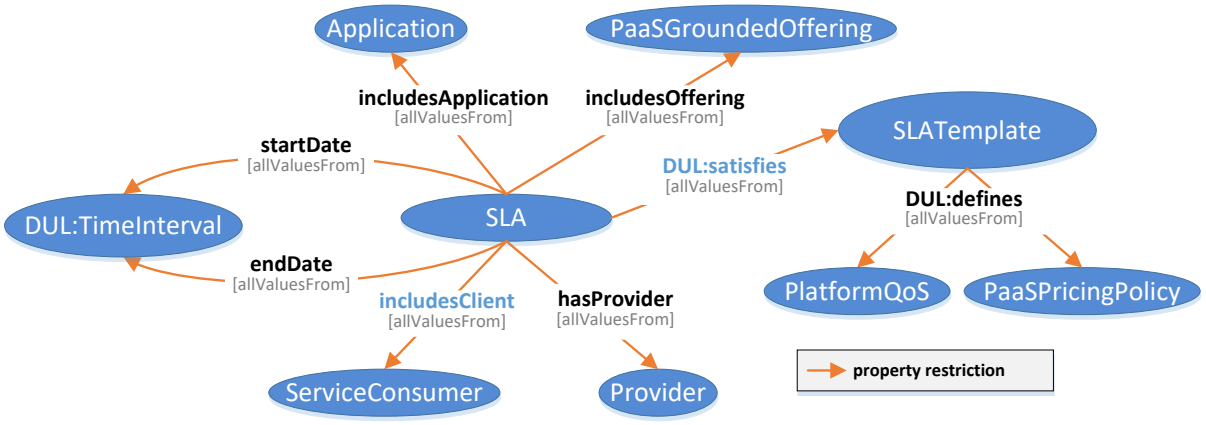

Figure 10. SLA overview.

Notice that the application requirements are added by the DevOps engineer to describe the requirements that the application has from the cloud computing platform to be deployed at. If the provided requirements are not really compatible with (do not "satisfy") the actual requirements, then two things can happen:

a) If the requirements are less than actually required (or "buggy"), then the application will run less than optimal (or not run at all). This will result in the dissatisfaction of the client of the application.
b) If the requirements are more than actually required, the platform offering will probably cost more. This will result in the dissatisfaction of the client of the application.

Therefore, it is in the best interest of the DevOps engineer to report the application requirements truthfully.

4.3.3 SLA Model

SLA (Service Level Agreement) is an agreement between two parties, the Service Consumer (the PaaSport user who deployed an application) and a PaaS Offering Provider. The level of service is formally defined in terms of performance and reliability, through the SLATemplate class, which is a rough schema of the offers the responder is willing to accept, and it also involves the application and the offering. The SLA has a period of validity that is defined in terms of the StartDate and EndDate properties. The performance is described by platform QoS parameters and the pricing by the pricing policy parameters (**Figure 10**), similarly to the platform QoS and pricing parameters of the PaaS Characteristic hierarchy (see section 4.3.4).

4.3.4 PaaS Characteristics and Parameters

This subsection describes in detail the key notions of core PaaSport ontology classes PaaSCharacteristic and PaaSParameter, which are used by all PaaSport semantic models and bridge the gap between them. In PaaSport, a **PaaSCharacteristic** is a basic unit of a PaaS offering capability or characteristic (or of a cloud application requirement) and represents an abstract concept of a cloud platform feature (e.g. programming language, database, uptime, storage, etc.). For the application developer it represents an application requirement about the deployment platform and for the PaaS provider it is a part of the description of the capabilities / characteristics offered by the platform. For example, in the PaaS offering of **Figure 21**, sample instances of PaaS Characteristics include, Java (v. 1.6), instance of the ProgrammingLanguage class (subclass of PaaSCharacteristic), MySQL, PostgreSQL and MongoDB, instances of the Database class, and platform QoS characteristics, such as latency and uptime.

Each PaaSCharacteristic is associated with one or more PaaSParameters, through the DUL:hasParameter property and can be considered as an aggregator object for related parameters. For example, the MySQL database PaaSCharacteristic might be comprised of parameters concerning the database service name (MySQL), the database type (SQL), the version of MySQL (5.7), the provided storage capacity of the database (e.g. 10GB), etc. Characteristics can be backwards compatible with other characteristics (through the isCompatibleWith property). This is mainly used for characteristics such as versions of offered programming languages or services/software.

Figure 11. The PaaS Characteristic hierarchy

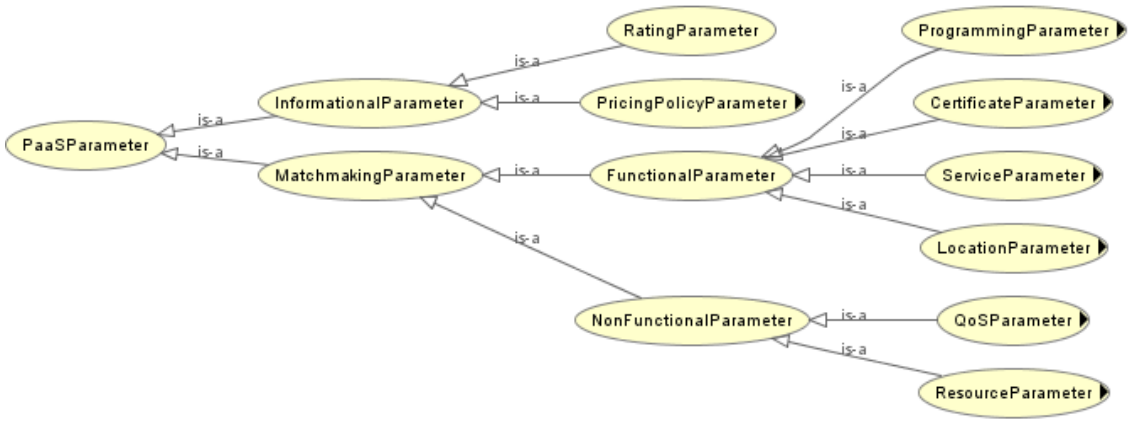

Figure 12. The PaaS Parameter hierarchy

PaaS characteristics are classified into two categories: a) characteristics that deal with the infrastructure needed to deliver the PaaS offering, such as Programming Environment, Service, Resource, and Certificate, and b) characteristics that refer to qualities inhering in a PaaS offering, such as platform QoS, Pricing Policy, Location and Rating. **Figure 11** shows the class hierarchy for PaaS characteristics.

A (PaaS)parameter is a property of a (PaaS)characteristic, enriching it with additional descriptive context. For example, the value "0.09 seconds" refers to the Latency of the provided service. The value of the parameter is defined using the **hasParameterDataValue** property. Through OWL restrictions we associate parameters with specific PaaS characteristics. A PaaSParameter can be either:

- a **MatchmakingParameter**, i.e. a parameter that participates in matchmaking and ranking, or
- an **InformationalParameter** that is used only for informational reasons and can be inspected manually by the application developer for decision-making or any other purpose.

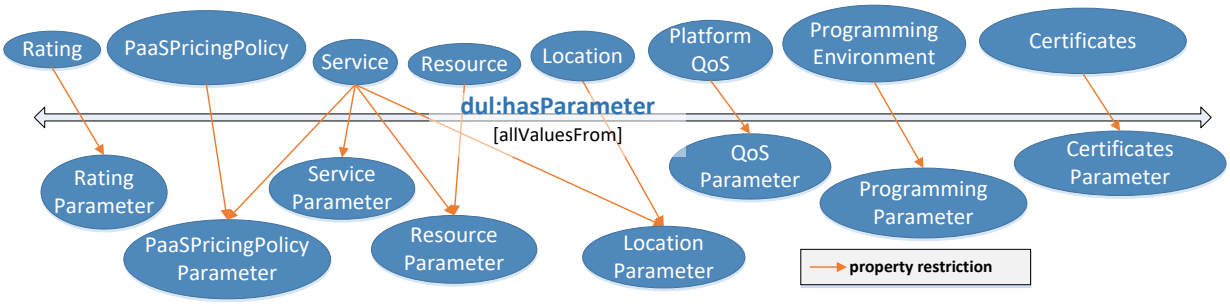

Figure 13. Overview of characteristics and parameters.

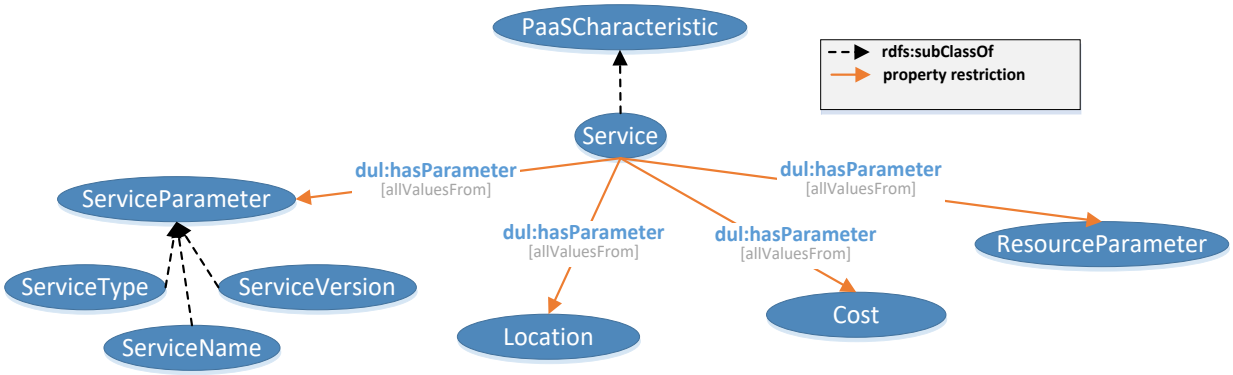

Figure 14. Parameters of a Service.

Moreover, the matchmaking parameters are divided into functional and non-functional parameters via the **FunctionalParameter** class: when a parameter is a subclass of FunctionalParameter, then it can only be used as a functional requirement, otherwise it can be used both as a functional and a non-functional requirement. Functional requirements are the requirements that, when not met by an offering, then the offering cannot be considered as a candidate for deploying an application. Non-functional parameters usually measure the quality of a service and are used in order to rank the selected services according to the order of preference. Notice that a non-functional parameter can also be used as a functional one if the user wishes to. For example, if "latency less than 10ms" is absolutely required, offerings that do not satisfy this criterion are not considered at all. This can be declared by the user through the user interface. **Figure 12** illustrates the major subclasses of PaaSParameter and **Figure 13** gives an overview of the association between characteristics and parameters. In the following, we elaborate on some key PaaS Characteristics and parameters.

Service is a piece of software that is part of a platform offering (pre-installed), such as a database server, a web server, etc. Services are related to the location where the servers that provide them are located (especially for cloud databases this is important due to legislation issues), resources (e.g. up to which storage capacity the application can use, either for all its data or just for a single service), and cost (either of the platform as a whole or for a specific service). The service parameters are categorized into four categories (see **Figure 14**):

- **ServiceParameter** describes the basic properties of a service, such as name, type, version etc.
- **Cost** refers to pricing policy of the specific service.
- **ResourceParameter** describes the platform resource-related parameters of a service, such as storage capacity, memory capacity, bandwidth

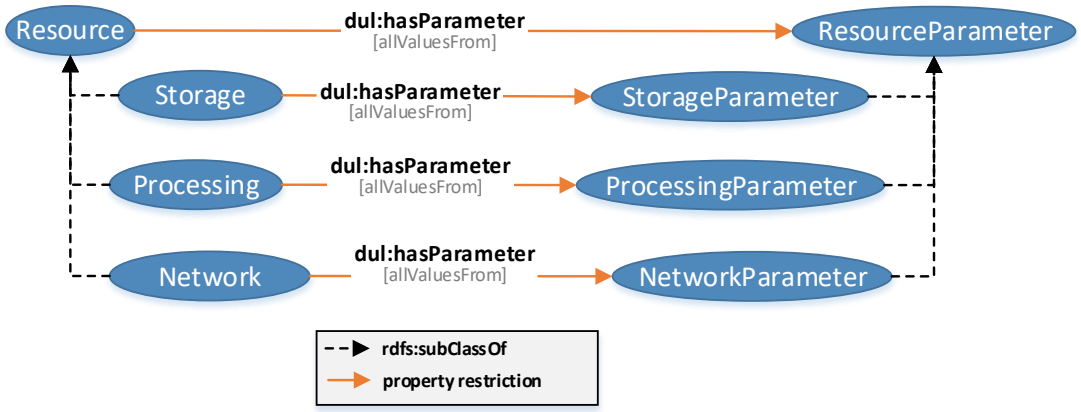

Figure 15. Resource and resource parameters.

etc. Notice that some of these parameters may reflect the QoS for the individual service / software provided by the platform, as opposed to the QoS parameters of the whole Platform-as-a-Service.

– **LocationParameter** describes where the server that provides the service is physically located; for example, a service can be a database server located in Europe.

In addition, the Service class has two subclasses **DB** and **Server** (**Figure 11**). The DB (SQL, NoSQL) class represents a database service provided by a platform offering and the Server class represents a web server or any other type of server that might be offered by the platform. The subclasses of Service are directly related to instances of the ServiceType class through a necessary and sufficient hasValue restriction. Notice that when a user of the PaaSport platform needs to add services that are not in the above two categories, then he/she can create a new direct instance of class Service, without any specific type. However, if this service belongs to a service type that is missing from the PaaSPort taxonomy, then only the PaaSport administrator can evolve the ontology by adding a new subclass of the Service class and the corresponding instances of the ServiceType class. This can only be done offline (e.g. upon platform upgrades) and requires some minor amendments to the ontology, the DB of the persistence layer and the user interface.

An example of a Service (Apache Server 2.2) is presented below:

```
<paas:Server rdf:about="&paas;apache_2.2">
  <DUL:hasParameter>
    <paas:ServiceVersion
rdf:about="&paas;apacheVersion">
      <DUL:hasParameterDataValue
rdf:datatype="&xsd;string">
        2.2</DUL:hasParameterDataValue>
    </paas:ServiceVersion>
  </DUL:hasParameter>
  <DUL:hasParameter>
    <paas:ServiceName
rdf:about="&paas;apacheName">
      <DUL:hasParameterDataValue
rdf:datatype="&xsd;string">
        Apache</DUL:hasParameterDataValue>
    </paas:ServiceName>
  </DUL:hasParameter>
  <DUL:hasParameter rdf:resource="&paas;Serv-
erType"/>
</paas:Server>
```

Class **Location** describes the geographical location of a platform or a service of a platform, since it is usual that due to legislation issues, a DevOps engineer may require the whole platform or a service of the platform to be located somewhere specifically. Location is associated with **LocationParameters**. Currently in PaaSport we support continents and countries. However, this could be refined to smaller granularity if needed. Furthermore, we may link these entities to proper geographical linked open datasets in the future.

Class **Certificate** describes certificates and standards (e.g. involving security) of a platform or certificates/standards required by an application. Certificate is associated with **CertificateParameters**, which can be one of CertificateName, CetificateType, and CertificateVersion.

The **Resource** class describes the hardware-related resources offered by the platform or requested by an application developer, e.g. storage capacity, memory capacity, network bandwidth, etc. Its subclasses are **Storage**, **Network** and **Processing** (see **Figure 15**).

Figure 16. Full Resource Parameters hierarchy.

The actual resources are specified as resource parameters. In a similar manner, the subclasses of the **ResourceParameter** class are **StorageParameter** (e.g. disk type, capacity), **ProcessingParameter** (e.g. CPU architecture, number of cores, memory capacity) and **NetworkParameter** (e.g. bandwidth, latency, concurrent connections) (see **Figure 16**).

Notice that usually, in Service Oriented Architecture, services are described logically and independently from their physical realization. In this vein, the connection of a service to resources might seem inappropriate. However, in the PaaSport ontology we use the term "service" differently, as already explained, namely as a piece of software offered on a Cloud platform by a PaaS provider to the service (PaaS) consumer. Therefore, each platform offering (according to our market research in section 4.1) has different resource settings according to different plans (pricing policies). For example, a free plan may offer very limited memory/storage capacity, whereas a premium plan would allow larger capacities up to a limit, etc. Therefore, the correlation of platform services/software to resources, in PaaSport, is not connected to the physical implementation of the services, but rather to the allowed usage of resources from the deployed application according to the selected pricing policy of the offering.

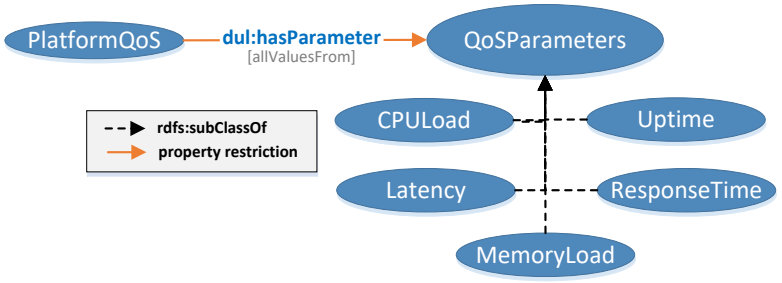

Figure 17. Platform QoS and parameters.

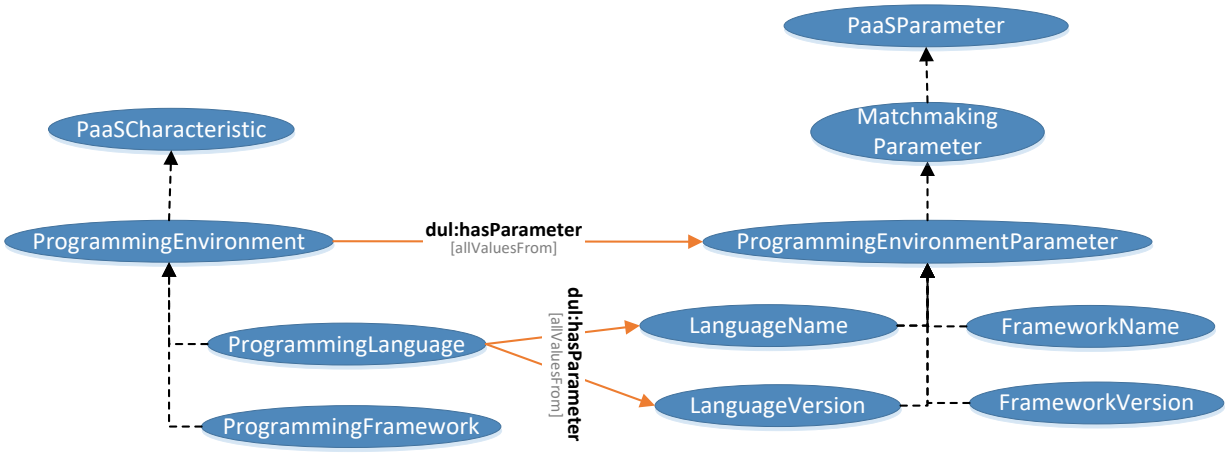

Figure 18. ProgrammingEnvironment and parameters.

Furthermore, concerning resources (such as storage) provided as a service, at a first glance it might seem risky to indicate (or dictate) details about how it is provided (e.g. SSD or HDD disk type), since it might limit the ability of the PaaS provider to effectively manage their available resources. However, the PaaSport ontology aims to aid PaaS providers to properly describe / advertise their platform capabilities and application developers to accurately express their application requirements (either functional or non-functional / performance-related) in order to effectively select the correct and rank / shortlist the best platform offerings to deploy their application. In this vein, the type of resources, which usually dictate both the expected QoS as well as pricing, is definitely relevant. This allows the developer to choose between quality (but expensive) platform plans or inexpensive (but more common) PaaS offers. One such example can be found at the Heroku platform[7], where some of the cheaper plans do not include SSD disks for the storage, whereas the more expensive plans do. This indicates that such resource details are important for selecting the appropriate platform offering.

An example of a storage resource (OpenShift free gear Storage) is presented below:

```
<paas:Storage rdf:about="&paas;gearStorage">
  <DUL:hasParameter>
    <paas:StorageCapacity
rdf:about="&paas;gearStorageCapacity">
      <DUL:parametrizes
     rdf:resource="&paas;StorageCapacityGB"/>
      <DUL:hasParameterDataValue
rdf:datatype="&xsd;integer">
        1
      </DUL:hasParameterDataValue>
    </paas:StorageCapacity>
  </DUL:hasParameter>
</paas:Storage>
```

[7] https://elements.heroku.com/addons/jawsdb

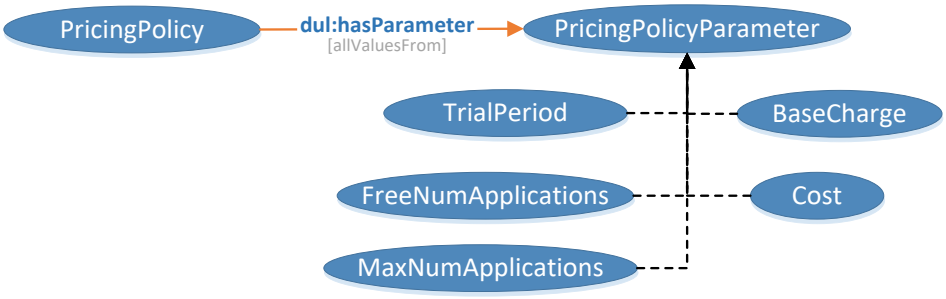

Figure 19. PricingPolicy and parameters.

Class **ProgrammingEnvironment** describes a programming language (e.g. Java, PHP, Python) or a programming language framework (e.g. PHP Zend, Python Django) used for developing Cloud-based applications. Every instance of ProgrammingEnvironment has some parameters, such as language name, language version, framework name, and framework version. These parameters are subclasses of **ProgrammingParameter** (**Figure 17**).

An example of the Java programming language (version 1.6.0) is given below:

```
<paas:ProgrammingLanguage
rdf:about="&paas;java_1.6.0">
  <DUL:hasParameter>
    <paas:LanguageVersion
rdf:about="&paas;javaVersion_1.6.0">
      <DUL:hasParameterDataValue
rdf:datatype="&xsd;string">
        1.6.0
      </DUL:hasParameterDataValue>
    </paas:LanguageVersion>
  </DUL:hasParameter>
  <DUL:hasParameter>
    <paas:LanguageName   rdf:about="&paas;ja-
vaName">
      <DUL:hasParameterDataValue
rdf:datatype="&xsd;string">
        Java
      </DUL:hasParameterDataValue>
    </paas:LanguageName>
  </DUL:hasParameter>
</paas:ProgrammingLanguage>
```

Class **PlatformQoS** represents the platform's Quality of Service metrics and its parameters are subclasses of QoSParameter. These parameters are MaxCPULoad, MinCPULoad, Latency, MaxMemoryLoad, MinMemoryLoad, ResponseTime and Uptime (**Figure 18**).

MaxCPULoad is the upper limit on the percentage of CPU load after which the platform scales up, e.g. "*scale up if CPU load is higher than 90%*". Notice that this limit can be interpreted by the application requirement in two ways:

- If the application profile is similar to a server-like application, then the application requirement for this limit is to be as low as possible, in order to be able to handle as many new requests as possible. So the application requirement treats this as a minimum acceptable limit; a "compatible" offering may offer the same-as-requested or an even lower CPU load.
- If the application profile is similar to a data-intensive application, then the application requirement for this limit is to be as high as possible, in order to utilize the CPU as much as possible. So the application requirement treats this as a maximum acceptable limit; a "compatible" offering may offer the same-as-requested or an even higher CPU load.

In order to comply with both the above application profiles, the DUL:parametrizes property is restricted to RangeValue, i.e. the superclass of the Max and Min classes. When the application requirement is defined, the parameter will become an instance of one of the two most specific classes Max and Min. Max is for data-intensive apps, Min is for server-like apps. Exactly the same behavior is met by **MaxMemoryLoad** parameter, which is the upper limit on the percentage of memory load after which the platform scales up.

MinCPULoad is the lowest limit on the percentage of CPU load below which the platform scales down, e.g. "*scale down when CPU load is below 30%*". Notice that this limit is always interpreted in the same way by the application requirement; it needs to be as high as possible so that the platform will scale-down early enough so that the CPU load is maintained relatively high (e.g. at least 50%). In this ways cost is saved (no need to pay for extra VMs when not really needed). Therefore, it is treated as a Min Range Value. Exactly the same behavior is met by **MinMemoryLoad** parameter, which is the lowest limit on the percentage of memory load below which the platform scales down.

Latency is the maximum latency of an offering (or the maximum latency required by an application). It is the delay incurred due to communications. Furthermore, **ResponseTime** is the time elapsed between sending a request and the reception of the first response, whereas **Uptime** is the percentage of time in a specific period that the application is up and running, or the offering is available.

PricingPolicy contains details about the pricing policy of a PaaS offering (as a whole) or of the cost of a specific service offered by a PaaS offering (e.g. database cost) and can take parameters of the type **PricingPolicyParameter**. Some of the pricing policy parameters are base charge of the offering/service, trial period, extra costs for additional services, the number of free applications that a developer can deploy, and the maximum number of instances that a developer can deploy (**Figure 19**). Finally, class **Rating** represents the rating of a PaaS offering; this value is the average of the users' ratings.

4.3.5 Quality Values and Units

The Measurement Unit Ontology[8] (MUO) has been used for semantically representing the various measurements and units in the PaaSport domain. The ontology can be used for modelling physical properties or qualities. Every unit is related to a particular kind of property. For instance, the Hz unit is uniquely related to the frequency property. Under the provided ontological approach, units are abstract spaces used as a reference metrics for quality spaces, such as physical qualia, and they are counted by some number. For instance, weight-units define some quality spaces for the weight-quality where specific weights of objects, like devices or persons, are located by means of comparisons with the proper weight-value of the selected weight-unit.

In MUO, the class muo:QualityValue (a specialization of dul:Region) is used for representing the values of qualities, for instance, the amount of available memory. Instances of this class are related with: a) exactly one unit, suitable for measuring the physical quality (meters for length, grams for weight, etc), by means of the property muo:measuredIn (a specialization of dul:isPrametrizedBy); b) a number, which expresses the relationship between the value and the unit by means of the rdf:value property; and c) a time, which expresses the quality value along the line of time. Quality values can be temporalized, but this is not always necessary. In PaaSport, we use MUO to represent the units as well as qualitative attributes, whereas values are represented using the DUL vocabulary (dul:hasParameterDataValue).

The reason why we have used the MUO ontology, instead of defining our own units using the DUL class dul:UnitOfMeasure is that MUO has a large set of well-known predefined unit instances, derived from "The Unified Code for Units of Measure (UCUM)"[9]. Almost all physical, chemical and IT units are already defined there. However, we had to extend the unit knowledge base with some derived IT units, such as GB or MB, but we have re-used the basic unit (byte).

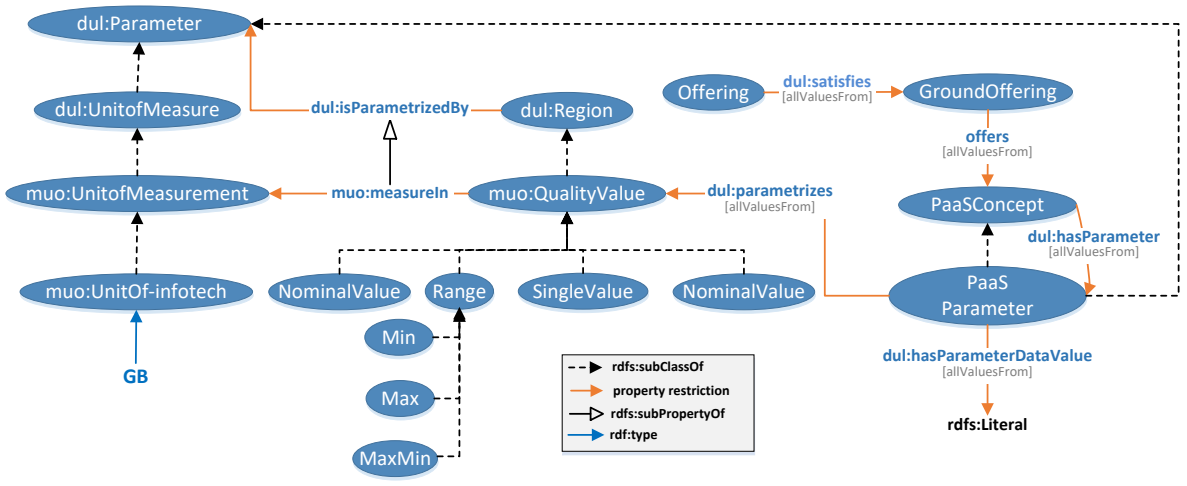

Figure 20. Hierarchy of quality values and units of measurement.

[8] http://idi.fundacionctic.org/muo/

[9] http://unitsofmeasure.org/trac

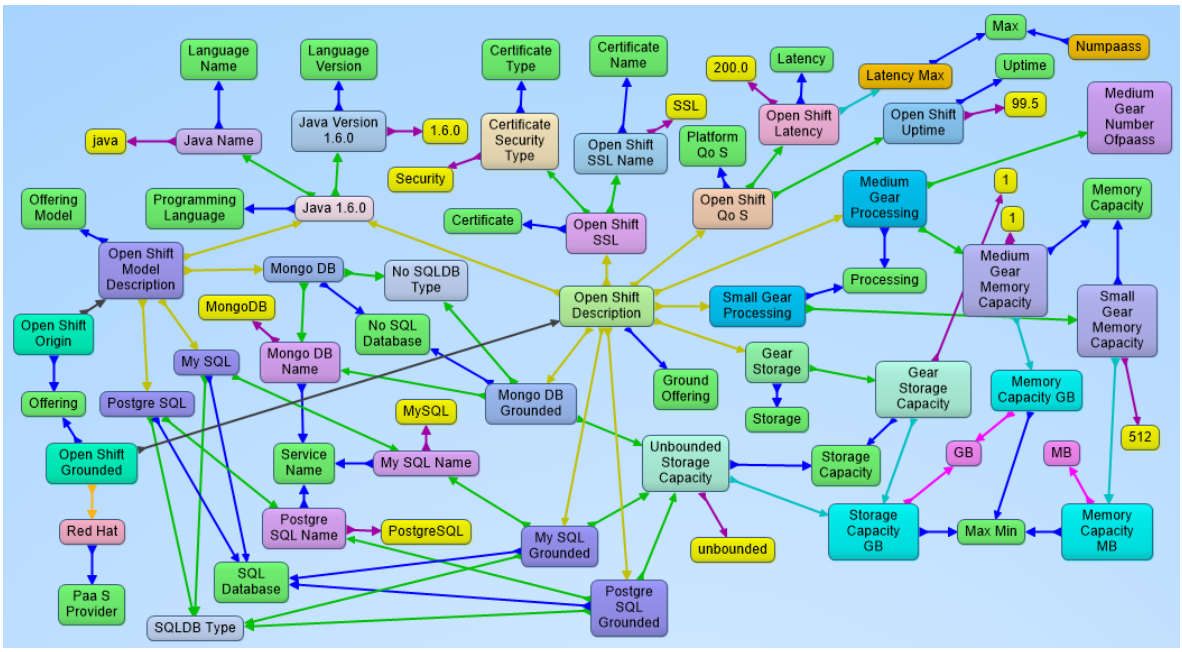

Figure 21. A complete offering example.

Table 6. Legend of colors of **Figure 21**.

Properties	Classes		
Has Parameter	Certificate	Max Min	Qo S
Has Parameter Data Value	Certificate Name	Memory Capacity	Service Name
Measured In	Certificate Type	No SQL Database	Service Type
Offers	Class	Number Of Cores	Simple Derived Unit
Parametrizes	Ground Offering	Offering	SQL Database
Provided By	Language Name	Offering Model	Storage
Satisfies	Language Version	Paa S Provider	Storage Capacity
Type	Latency	Processing	Uptime
	Max	Programming Language	Literal

In PaaSport (**Figure 20**), there are a lot of quality values that represent how a value of a PaaS offering parameter can be compared and matched to the corresponding application requirement parameter:

- Single Values, either symbolic or numeric, that require an exact match.
- Nominal Values, which are enumerated data types and require an exact match.
- Ordinal Values, namely ordered enumerated data types, which also require exact match, but order can be established for better or worse.
- Range Values, which are numeric values that requires range match, e.g. “less than” or “equal”. There are four subclasses of this class, according to the matchmaking profile of each parameter.
  - Max: Range Value with a Max upper limit. Matches less than or equal.
  - Min: Range Value with a Min upper limit. Matches “greater than” or “equal”.
  - MaxMin: Range Value with a limit that is Max for the Offering and Min for the Application. Matches “less than” or “equal”.

- MinMax: Range Value with a limit that is Min for the Offering and Max for the Application. Matches greater than or equal.

4.3.6 Instance Example

In **Figure 21**, we include a complete instantiation example of a PaaS offering, namely OpenShift. Notice that there are two major instances involved, the OpenShift Origin PaaS model, and the grounded OpenShift offering from RedHat. The grounded offering includes two different containers (Gears) with small and medium main memory and storage capacities, the Java language (v. 1.6), MySQL, PostgreSQL and MongoDB with unbounded storage capacity, and platform QoS characteristics, such as latency 200 ms and 99.5% uptime. The color legend for the properties and the classes of **Figure 21** are shown in **Table 6**.

4.3.7 Expressivity and Reasoning

The PaaSport Semantic Models reuse the conceptual model provided by the DOLCE+DnS Ultralight (DUL) foundational ontology and therefore, they inherit all the modelling properties and expressivity characteristics of the upper-level model. More specifically, the expressivity falls under the SHOIN(D) description logic, allowing a) atomic negation, that is, negation of concepts that do not appear on the left hand side of axioms; b) concept intersection; c) universal restrictions; d) existential quantification; e) complex concept negation; f) inverse properties; and g) cardinality restrictions.

Regarding the computation complexity of reasoning, it strongly depends on the OWL 2 reasoning profile that will be used to implement the matchmaking algorithms. An OWL 2 RL-based implementation is NP-COMPLETE, whereas by using an OWL 2 DL reasoner (under direct semantics), the reasoning complexity increases to N2EXPTIME-complete, supporting though higher expressivity. Similarly, the computational complexity of SPARQL (SPARQL Protocol and RDF Query Language) that will be used to query the data strongly depends on the language constructs used for defining queries. The full SPARQL (e.g. using FILTER, UNION, OPTIONAL operators) is PSPACE-complete, whereas all OPTIONAL-free graph patterns are either NP-COMPLETE (whenever operator AND co-occurs with UNION or SELECT/CONSTRUCT) or in PTime.

5 Interaction with the Recommendation and Persistence Layers

The PaaSport Semantic Model strongly interacts mainly with the Recommendation and the Persistence layers of the PaaSport Broker (**Figure 22**).

5.1 Persistence Layer

The Persistence (or Repository) Layer is used in order to (a) persist the various PaaSPort data models that are mapped to the semantic model, (b) persist other entities that are needed for the proper function of PaaSPort Marketplace, and (c) offer search and discovery interfaces that allow the usage of persisted information from other components. The main component of the persistence layer is a Relational database that is used to store the data that are necessary for the operation of the platform. The repository contains the three main types of data objects that PaaSport Marketplace needs to store: a) the PaaS Offering Profiles constituting the semantic profiles of the PaaS offerings advertised in the PaaSport Marketplace; b) the Deployed Application Profiles constituting the semantic profiles of the deployed business software applications; and c) the User Profiles constituting the semantic representation of the profiles that Software SMEs Engineers and PaaS Providers maintain on the PaaSport Marketplace.

A crucial aspect of the repository layer is the ability to expose a specific part of its data in RDF format in order for the semantic matchmaking to take place (**Figure 22**), in a process called as RDFization. Data stored in relational systems can be extracted via queries, stored procedures, or any other process that will extract the data from the database. Table columns and rows have to be mapped to concepts and attributes defined in an ontological model.

In order to achieve the RDFization, we use the D2RQ Platform [15], which is a system for accessing relational databases as virtual, read-only RDF graphs. It offers RDF-based access to the content of relational databases without having to replicate it into an RDF store and provides access to the content of the database as Linked Data over the Web. The D2RQ Platform comprises of a set of tools, which offer SPARQL access, a Linked Data server, an RDF dump generator, a simple HTML interface, and Jena API [3] access to D2RQ-mapped databases (**Figure 23**). The D2RQ Platform provides a declarative language, the D2RQ Mapping Language, for mapping relational database schemas to RDF vocabularies and OWL ontologies. A

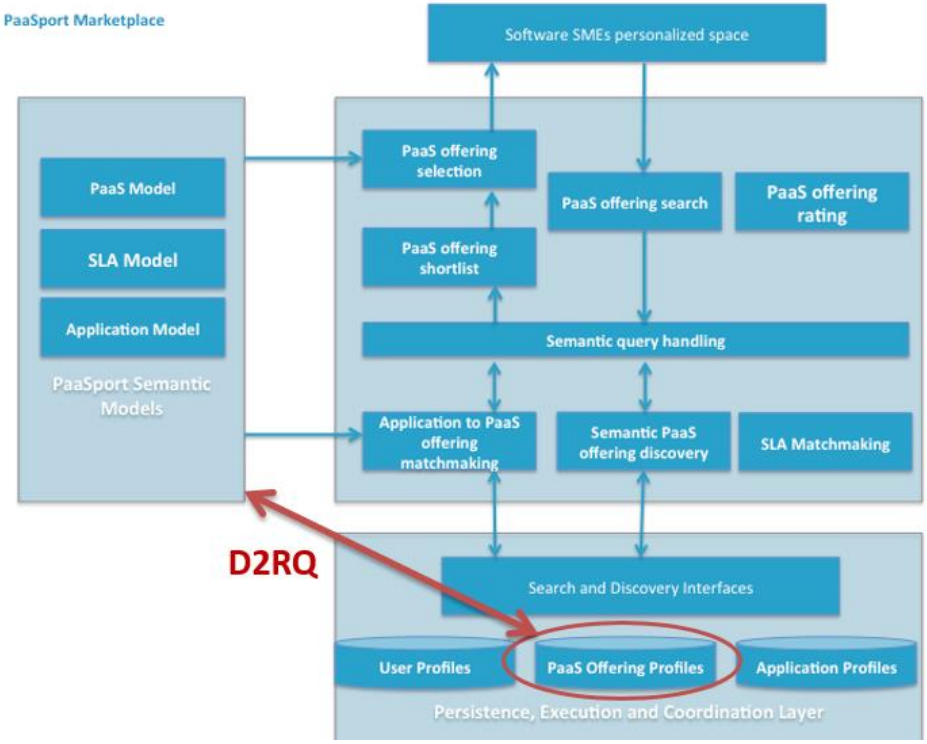

Figure 22. Relational to RDF mapping

D2RQ mapping is an RDF document written in Turtle syntax. The D2RQ mapping defines a virtual RDF graph that contains information from the database. The virtual RDF graph can be accessed in various ways. Furthermore, the D2RQ platform comprises of the D2RQ Engine, a plug-in for the Jena Semantic Web toolkit, which uses the mappings to rewrite Jena API calls to SQL queries against the database and passes query results up to the higher layers of the frameworks. Finally, the D2RQ Server is a publishing tool for the content of relational databases.

In PaaSport we use the D2RQ mapping language in order to export the offering profiles from the relational database of the persistence layer to an RDF format, so that they can be used by the matchmaking and ranking algorithm of the recommendation layer (**Figure 27**). The D2RQ language is also used to create all essential mapping rules that are stored in a file. Every time that a PaaS provider inserts a new offering instance in the database, a script is responsible to recreate the PaaSport ontology file.

The D2RQ language can connect RDF triples to database tuples and attributes. At the example below (**Figure 24**), the mapping rule for the grounded offerings is presented. First, there is a rule for connecting to the database, so it defines the port of the database endpoint, username and password. After that, a mapping rule is defined, with the name of the rule-triplet, the defined data storage, the URI pattern of the triplets and the name of the class. Thus, for every tuple in the table *groundedpaasoffering* the mapping rule creates a new triplet.

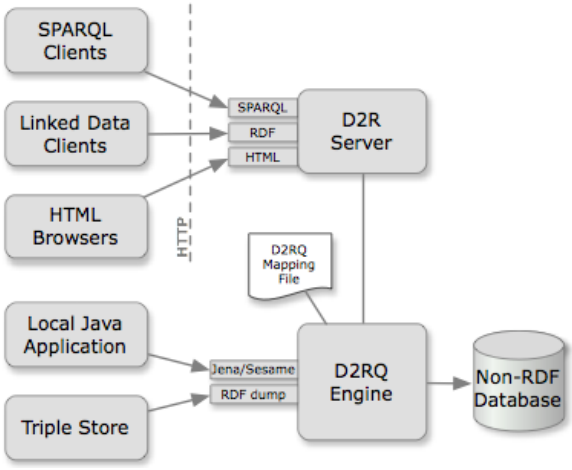

Figure 23. Architecture of the D2RQ Platform (taken from [15]).

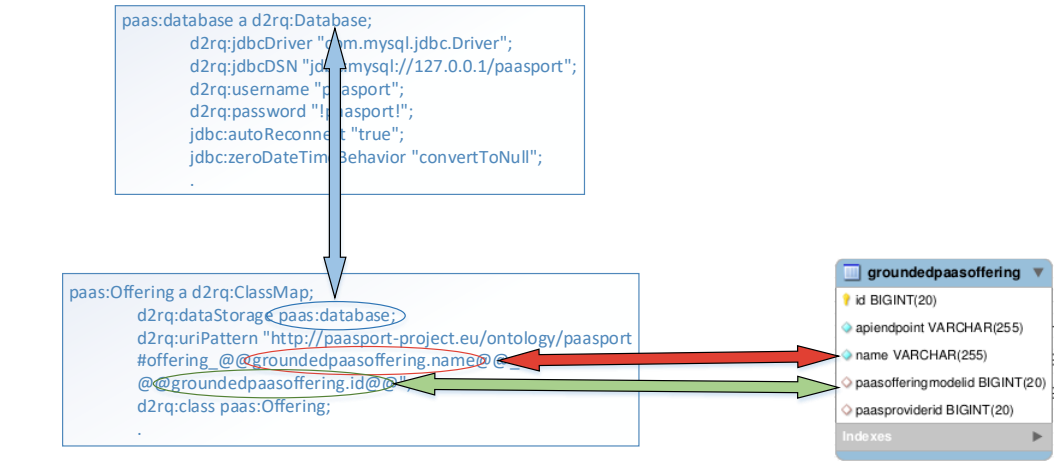

Figure 24. D2RQ example; connecting to the database and mapping a table

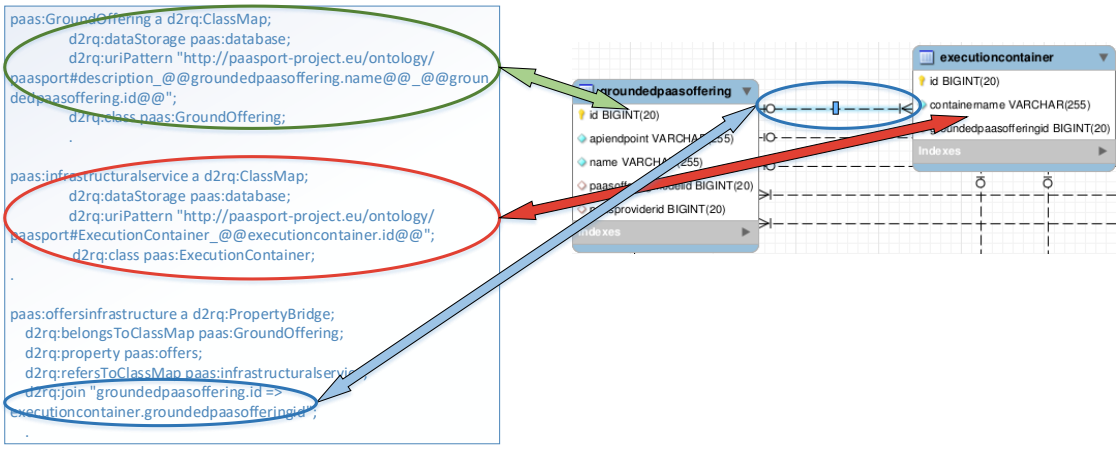

Figure 25. D2RQ example; mapping a property.

In **Figure 25**, a property-mapping example is presented. First, there is a rule that creates the property domain class (*GroundOffering* class). Then, there is a rule that creates the property range class (*ExecutionContainer* class). Finally, there is a rule that makes the property mapping, in our case, the property *paas:offers*. The connection is based on a join between the tables of the two classes above on the *id* attribute. Thus, every *GroundOffering* instance connects to a corresponding *ExecutionContainer* instance through the property offers, based on the id of the *groundpaasoffering* tuple.

At this point, it is worth mentioning that, alternatively, the platform offerings could have been stored directly into a native RDF database (i.e. a triplestore), so that all the above effort on converting relational data to RDF data could have been avoided. Notice, however, that the technology stack of the PaaSport broker is based on the Spring Framework [57], Spring Data [56], and Hibernate ORM [32], which conjunctively offer fast development times [48]; therefore, it is inevitable to use a relational database system to store the data objects that are required in order for all PaaSport layers to interoperate. Therefore, if we have chosen to duplicate data about platform offerings into an RDF database, we would have the additional complication of synchronizing data between the two database systems, which would be even worse than the complication of extracting data in an RDF format.

5.2 Recommendation Layer

The Recommendation Layer of the PaaSport Reference Architecture (Section 3) involves the development of algorithms and software for supporting the selection of the most appropriate PaaS offering that best

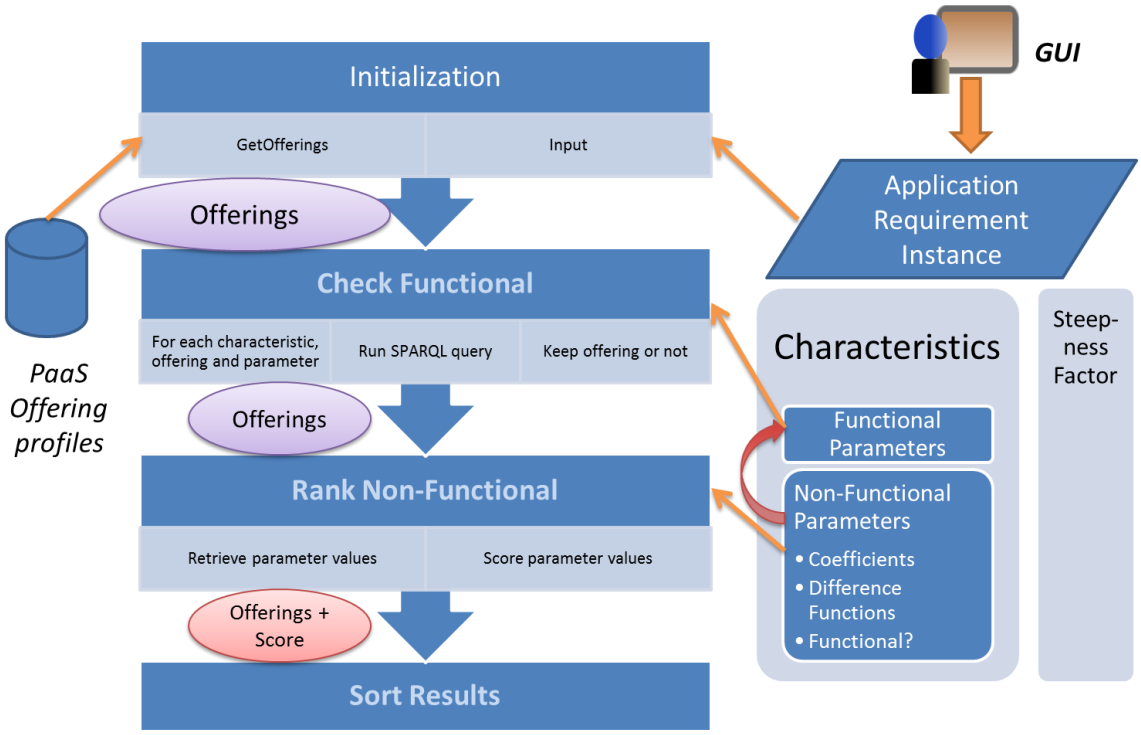

Figure 26. Overview of the PaaSport Matchmaking and Recommendation algorithm

matches the requirements of the application a developer wants to deploy. Under this context, the PaaSport recommendation algorithms and models are aimed at providing the necessary semantic layer on top of the offering and application model descriptions, solving interoperability issues and improving the quality of the recommendations. To this end, standard vocabularies and ontology languages are used for capturing the structural and semantic characteristics of the various entities involved in the PaaSport domain, whereas the underlying conceptual models facilitate the use of lightweight reasoning during the matchmaking process.

At the heart of the PaaS Offering Recommendation Layer there is a recommendation algorithm that selects and scores-ranks the most appropriate PaaS offerings that best match the requirements of the application a developer wants to deploy. The matchmaking and ranking algorithm consists of two steps (**Figure 26**):

a) Selection of those offerings that satisfy the functional parameters.
b) Scoring of the remaining (from step a) offerings using an aggregation scoring function on all the non-functional parameters.

Note that when talking about functional and non-functional parameters, we refer to the parameters that the DevOps Engineer has set as application requirements through the corresponding GUI. Also, note that parameters are classified as either functional or non-functional from the PaaS Semantic Model (Section 4). Therefore, the user interface can be constrained by the Model on which parameters can be used as functional or non-functional. However, non-functional parameters can be used both as non-functional and functional. For example, one might require that the storage capacity of the offering should be no less than 10GB and that he/she is not willing to consider offerings in the final ranked list with less storage, even with a lower score than the others are. In this case, the parameter will be included twice in the list of parameters retrieved by the GUI, once in the functional parameters list and once in the non-functional parameters list. However, the opposite is not allowed, i.e. a functional parameter (set by the Semantic Model, e.g. the Programming Language) can never be treated as non-functional.

Table 7. SPARQL template for checking a functional parameter with a MinMax range numerical value.

```
1.  ASK {
2.      <offering.ID> DUL:satisfies / paas:offers ?characteristic .
3.      ?characteristic rdf:type <characteristic.type> .
4.      {   ?characteristic DUL:hasParameter ?par .
5.      } UNION
6.      {   ?characteristic paas:hasCompatibilityWith+ / DUL:hasParameter ?par .
7.      }
8.      ?par rdf:type <par.type> .
9.      ?par DUL:hasParameterDataValue ?Value .
10.     {   ?par DUL:parametrizes <par.qualityValue> .
11.         BIND (1 AS ?Factor1)
12.         BIND (1 AS ?Factor2)
13.     } UNION
14.     {   ?par DUL:parametrizes / uomvocab:measuredIn ?Units .
15.         ?Units rdf:type <par.qualityValue.MeasureUnit.Type> .
16.         <par.qualityValue.MeasureUnit> rdf:type uomvocab:BaseUnit .
17.         ?Units rdf:type uomvocab:SimpleDerivedUnit .
18.         ?Units uomvocab:derivesFrom <par.qualityValue.MeasureUnit> .
19.         ?Units uomvocab:modifierPrefix / uomvocab:factor ?Factor2 .
20.         BIND (1 AS ?Factor1)
21.     } UNION
22.     {   ?par DUL:parametrizes / uomvocab:measuredIn ?Units .
23.         ?Units rdf:type <par.qualityValue.MeasureUnit.Type> .
24.         <par.qualityValue.MeasureUnit> rdf:type uomvocab:SimpleDerivedUnit .
25.         ?Units rdf:type uomvocab:BaseUnit .
26.         <par.qualityValue.MeasureUnit> uomvocab:derivesFrom ?Units .
27.         <par.qualityValue.MeasureUnit> uomvocab:modifierPrefix / uomvocab:factor ?Factor1 .
28.         BIND (1 AS ?Factor2)
29.     } UNION
30.     {   ?par DUL:parametrizes / uomvocab:measuredIn ?Units .
31.         ?Units rdf:type <par.qualityValue.MeasureUnit.Type> .
32.         <par.qualityValue.MeasureUnit> rdf:type uomvocab:SimpleDerivedUnit .
33.         ?Units rdf:type uomvocab:SimpleDerivedUnit .
34.         ?Units uomvocab:derivesFrom <par.qualityValue.MeasureUnit.BasicUnit> .
35.         <par.qualityValue.MeasureUnit> uomvocab:modifierPrefix / uomvocab:factor ?Factor1 .
36.         ?Units uomvocab:modifierPrefix / uomvocab:factor ?Factor2 .
37.     }
38.     FILTER( xsd:double(?Factor2)*?Value >= xsd:double(?Factor1)*<par.value>)
39. }
```

The algorithm implemented in the PaaS Offering Recommendation Layer accesses the PaaS Offerings profiles stored in the Persistence and Execution Layer. The algorithm uses predefined SPARQL query templates, according to the parameter type. **Table 7** shows such a typical SPARQL template for checking if a functional parameter of the application requirement is less than or equal with the corresponding parameter of an offering (quality value MinMax). The value of the parameter is a numerical value that may be characterized by measurement units (e.g. 2GB of memory). Similar such templates exist for any of the quality values presented in Section 4.3.5.

The SPARQL query navigates from the initial offering (<*offering.ID*>) to its characteristics (?*charac-*

teristic) that share the same characteristic type (<*characteristic.type*>) with the application profile (lines 2-3). Then all the parameters (?*par*) of the characteristic of the characteristic instance that share the same parameter type (<*par.type*>) with the application profile (lines 4-8) are retrieved. Note the search is not restricted only to parameters of the current characteristic instances, but also to parameters of compatible characteristics, through the *paas:hasCompatibilityWith* transitive property (line 6). This could be used for cases e.g. of different programming language versions that are backwards compatible.

In line 9 the value of the parameter is retrieved and lines 10-38 check if the value of the parameter of the offering is equal to corresponding value of the application requirement. The comparison for equality should also consider equality of units or equivalence of units, when units are convertible to each other. For example, 1024MB is equivalent to 1GB, although neither the value nor the measurement unit are equal. There are several alternative cases for equality; each one of them is one of the 4 graph patterns that are connected through the UNION operator.

The first case (lines 10-13) is where the quality value of the offering parameter is identical to the one of the application requirement (both values have identical units). Thus the values of the offering and the application will be compared directly, without unit conversions; that is why both variables ?*Factor1* and ?*Factor2* are set to 1. In the second case (lines 14-22), the measurement unit of the parameter of the application profile is a basic unit (line 16), so it cannot be converted (?*Factor1* set to 1), whereas the measurement unit of the parameter of the offering is a derived unit (line 17) that can be converted to the basic unit using a modifier ?*Factor2* (e.g. Giga) (line 19). For example, 20KB will be converted to 20*1024=20480 bytes. The third case (lines 22-30) is the exact symmetrical one. Finally, the fourth case (lines 30-37) is when both the offering and the application profile have parameters of derived units (lines 32-33). Instead of converting the one unit to the other, we convert both of them (lines 35-36) to the basic unit they originate from (line 34), so that they become comparable. Line 38 at the end is the filtering expression that compares the two converted (or not) values, using the alternative ?$Factor_i$ multipliers. Notice that this case also covers the case where both values are expressed in the same derived measurement unit.

The above SPARQL template is agnostic to the domain model; therefore, the algorithm will remain unchanged even if the PaaS ontology evolves in the future. More details on the recommendation algorithm and the SPARQL templates can be found at [5]. Furthermore, notice that the business model of the PaaSport Marketplace and the technical solution for interoperability / portability implemented by the PaaSport Broker imposes to the recommendation algorithm to search for a single platform offering of a single cloud provider, among multiple cloud providers, that offers all the compatible requested services. Usually PaaS providers charge for a platform as a whole and not as single provided services. Therefore, when a DevOps engineer wants to deploy / migrate an application from one cloud provider to another, usually searches for a better (more cost efficient) platform. This offers a cleaner deployment and maintenance solution. However, the recommendation algorithm can be easily extended to multiple cloud providers at the service level.

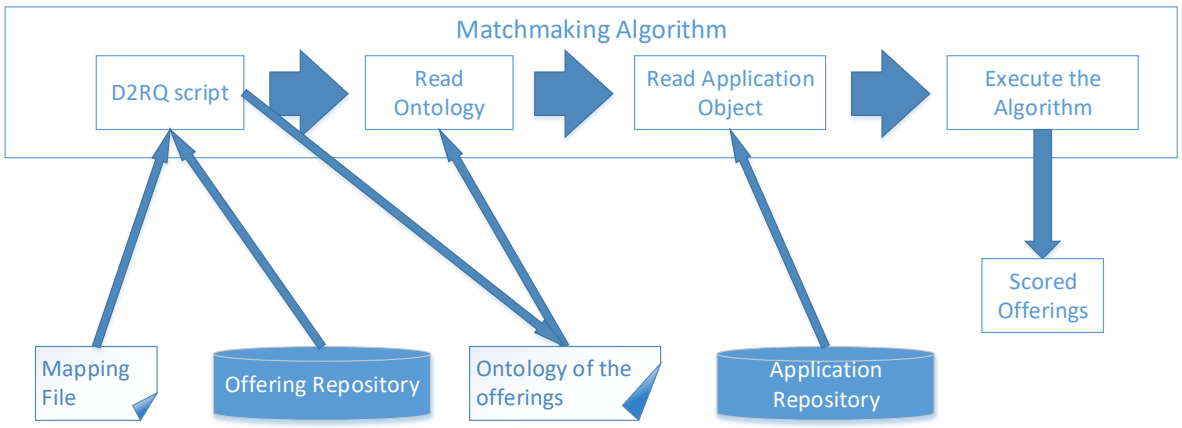

Figure 27. Workflow of Data for the Recommendation layer.

The Recommendation layer interacts with the persistence layer, through the matchmaking algorithm to:

a) Retrieve the PaaS offerings stored in the PaaSport Marketplace database;
b) Get the application requirements that the DevOps engineer has posted through the user interface.

Figure 27 shows the workflow of data from the persistence layer to the recommendation layer, concerning the inputs needed for the matchmaking/recommendation algorithm. The PaaS offerings are stored in the relational database of the persistence layer and they are mapped to RDF data, using the characteristics and properties of the Semantic Models, using the D2RQ platform presented above. Then the offerings are fed into the matchmaking algorithm. After that, the application requirements are queried from the persistence layer and they are used to construct an application object that is also used as input of the matchmaking/recommendation algorithm, which then proceeds as already described above.

6 Evaluation

In this section we evaluate the PaaSport semantic model in two different directions. First, we report on the metrics of the Ontology as well as how it was verified. Then, we evaluate the scalability of the ontology regarding size and query performance and we compare it with the ontology of Cloud4SOA [34].

6.1 Ontology Verification and Metrics

The PaaSport ontology[10] was developed with the TobBraid Composer Free Edition [60] from TopQuadrant. In this subsection we present the verification methodology we have used for the ontology, as well as some metrics associated with it, which have been provided by a different ontology editor (Protégé [52]). A literature review of ontology metrics reveals a variety of such metrics aiming to assess and qualify an ontology. A good overview of ontology evaluation methods is given in [25] and in [7]. An ontology evaluation process may target different qualitative or quantitative criteria. Such techniques help uncover errors in implementation and inefficiencies regarding the modelling, complexity and size of the ontologies. Nevertheless, none of the evaluation methods, neither alone nor in combination, can guarantee a "good" ontology, but can surely help identify problematic parts [63]. Any given approach may address more or less specific issues; therefore, evaluation methodologies partially clarify the problems at stake [25]. For verifying the PaaSport ontology, we have selected two evaluation methods, details of which are given in the following: (a) an automated ontology evaluation tool named OOPS! and (b) Protégé metrics.

OOPS! (OntOlogy Pitfall Scanner) is a web application [43] that helps detect some of the most common pitfalls when developing ontologies [51]. For example, OOPS! warns you when:

– The domain or range of a relationship is defined as the intersection of two or more classes. This warning could avoid reasoning problems in case those classes could not share instances.
– No naming convention is used in the identifiers of the ontology elements. In this case the maintainability, the accessibility and the clarity of the ontology could be improve.
– A cycle between two classes in the hierarchy is included in the ontology. Detecting this situation could avoid modelling and reasoning problems.

The generated results suggest how the ontology could be modified to improve its quality. Nevertheless, these suggestions should be manually interpreted and revised properly by the knowledge engineer. Three levels of importance have been identified in the evaluation process via OOPS!:

– critical pitfalls, that affect the ontology's consistency and reasoning,
– important pitfalls, which are not critical, but are important to fix, and
– minor pitfalls, which do not cause any practical problem, but correcting them will make the ontology clearer and more compact.

We have evaluated the PaaSport ontology by submitting it to OOPS!. Most of the pitfalls detected concern the imported ontologies, namely DUL and MUO, and they will not be reported here. **Table 8** includes the PaaSport ontology's pitfalls detected by OOPS!, along with a brief description. For missing annotations, the actions are trivial so we do not report them further. For the cases of missing inverse relationships, two of them involved the relationships "requires" and "offers" that relate an application requirement and an offering model to PaaS Characteristics. The inverse relationships "isRequiredBy" and "isOfferedBy" have been introduced as subproperties of the DUL property

[10] The ontology can be found at: http://lpis.csd.auth.gr/ontologies/paasport/paasport.owl

"isDefinedIn". The third case involved the isCompatibleWith property between PaaS Characteristics, which was turned into transitive and symmetric instead. Finally, the recursive definition involved the "PaaSCharacteristic" class and the isCompatibleWith property. The recursion was due to the fact that despite class PaaSCharacteristic was the domain and range of the property, we have included also a redundant local range property (allValuesFrom restriction) for this property at class PaaSCharacteristic, restricting it again to PaaSCharacteristic. This restriction was just removed.

Table 8. PaaSport Ontology's pitfalls detected by OOPS!

Pitfall	Number of cases
P08: Missing annotations (Minor) Ontology terms lack annotation properties, either rdfs:label or rdfs:comment, that would improve the ontology understanding and usability from a user point of view.	113
P13: Missing inverse relationships (Minor) There are relationships (except for symmetric ones) that do not have an inverse relationship defined within the ontology.	3
P24: Using recursive definition (Important) An ontology element is used in its own definition.	1

Table 9. PaaSport Ontology metrics by Protégé

Metric	Count	Metric	Count
Axioms	528	SubObjectPropertyOf axioms	20
Logical axioms	282	ObjectPropertyDomain axioms	19
Classes	112	ObjectPropertyRange axioms	17
Object properties	33	InverseObjectProperties axioms	7
Data properties	1	DataPropertyAssertion axioms	4
Individuals	4	AnnotationAssertion axioms	112
SubClassOf axioms	196	FunctionalObjectProperty axioms	2
EquivalentClasses axioms	5	ClassAssertion axioms	12
DL expressivity		SHOIN(D)	

Furthermore, we also present the ontology metrics provided by Protégé [52] that are based on the general structure of the ontology and are classified into the following general groups [27]:

- General metrics, such as counters for classes, object/data properties and individuals.
- Class axioms, such as subclass axioms, equivalent class axioms, disjoint class axioms, etc.
- Object property axioms, which include counters for object properties axioms such as total values of sub-object properties, equivalent, inverse, disjoint, functional, transitive, symmetric, antisymmetric, reflexive and irreflexive object properties, as well as counters for data properties domain and range.
- Data property axioms, including datatype properties counters, meaning total values of sub-datatype properties, equivalent, disjoint and functional datatype properties, as well as counters for data properties domain and range.
- Individual axioms, with counters for class assertions and same or different individual axioms.
- Annotation axioms, which includes counters for annotation assertions and for annotation property domain and range.

Table 9 includes the respective metrics for the PaaSport ontology, as provided by the 'ontology metrics' view in Protégé. Notice that only non-zero metrics are reported. As can be observed, the ontology is quite rich in classes, which are made subclasses of the DUL upper level ontology, but quite fewer new object properties are introduced, as subproperties of DUL properties, meaning that many DUL properties have been re-used. There is only one new datatype property. This is because parameters of the PaaS offerings are not directly represented as datatype properties, but instead they are made first class (reified) objects that actually need only one datatype property (hasParameterDataValue). This feature of the DnS ontology design property is very useful for the extensibility of the ontology. Specifically, when new features are added, either PaaS Characteristics or characteristic properties, one has to add only new classes as subclasses of already existing classes. The individuals defined in the ontology are instances of the MUO ontology, namely the KB, MB, GB, TB instances for measuring memory / disk capacities. These are all derived units of measurement, derived from 'byte'. The last row of **Table 9** refers to the DL (Description Logic) expressivity of the PaaSport ontology; DL provides the logical formalism underlying OWL 2. The PaaSport ontology has a DL expressivity level of SHOIN(D) (see section 4.3.7), therefore it is equivalent to OWL 2 DL.

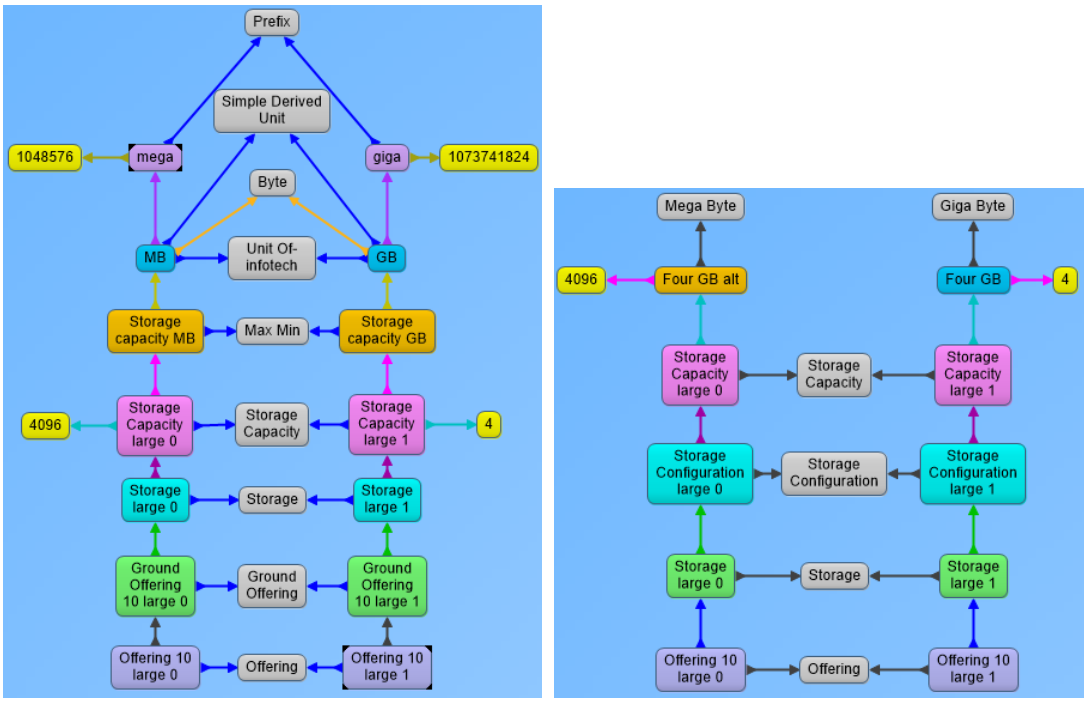

Figure 28. The offerings of the PaaSport semantic model (left) and the Cloud4SOA model (right).

6.2 Ontology Scalability

In order to evaluate the scalability of the ontology regarding size and query performance and compare it with "competing" ontologies, such as Cloud4SOA [34], we have performed the following experiments and comparisons:

a) Scalability of performance for queries over platform characteristics that bear different measurement units vs. queries that are not concerned with units, using the PaaSport semantic model. In this way, we will be able to evaluate the performance burden of having different measurement units.

b) Scalability of performance for queries (with and without measurement units) over the PaaSport semantic model vs. similar queries over a Cloud4SOA-like semantic model. In this way, we will be able to evaluate the performance burden of having an extensible semantic model on top of DUL.

In order to achieve these, we have generated three types of PaaS offerings multiple times and we measured the size and the response time of corresponding SPARQL queries that return all the offerings that exceed the minimum applications requirements set for a single platform characteristic. In this case, the characteristic is the storage capacity offered by the PaaS in three different settings 1, 2 and 4 GBs. In order to check just the scalability, we have also restricted the description of the offerings knowledge bases to just a single platform characteristic (storage capacity).

Figure 28 (left) shows two such offerings, using the PaaSport semantic model, used in the experiments, whereas **Figure 28** (right) shows the corresponding offerings using the Cloud4SOA-like semantic model. For all the experiments, we have created five different knowledge bases with 300 up to 3 million offerings. The ratio of the storage capacities among these offerings was 1/3 from each of the capacities. Furthermore, we have used two different settings for the experiment knowledge bases; in the first setting half of the offerings have their storage capacities expressed in GB, whereas the other half in MB, whereas in the second setting all offerings have their storage capacities expressed in GB, which makes querying easier. This makes in total 4 different types of knowledge bases (PaaSport model with GB and MB, PaaSport model with GB only, Cloud4SOA-like model with GB and MB, Cloud4SOA-like model with GB only), each in 5 different sizes (300, 3K, 30K, 300K, 3M offerings).

All knowledge bases (20 in total) were uploaded at corresponding repositories into Ontotext's GraphDB

Free[11] triplestore. The sizes of the knowledge bases are shown in **Table 10** and **Figure 29** (in KB) and they clearly scale linearly. Furthermore, there is not any notable difference between the PaaSport and the Cloud4SOA-like semantic models. Notice that only sizes of mixed GB-MB knowledge bases are reported; the corresponding knowledge bases with only GB have similar sizes.

Table 10. Size of offerings knowledge bases (in KB)

	300	3K	30K	300K	3M
PaaSport model	172	1,725	17,435	176,386	1,784,345
Cloud4SOA model	240	2,403	24,253	244,859	2,472,007

The application request tested was for offerings with storage capacity at least 2GB, so the expected result set is comprised of 2/3 of the total number of offerings with 2GB and 4 GB storage capacity. The SPARQL queries for the four different types of knowledge bases are shown in **Table 11**, **Table 12**, **Table 13**, and **Table 14**. The SPARQL queries for the PaaSport model in **Table 11** and **Table 12** are simplified versions of the query template presented in **Table 7**, adapted to the MinMax quality value of the storage capacity (see section 4.3.5). Actually, they represent two of the multiple UNION queries; the one that takes into account the measurement units (because some of the storage capacities are in GB and some in MB, whereas the application request is expressed in GB) and the one that does not need to take into account the measurement unit, because the query designer knows in advance that all offerings use exactly the same measurement unit, in this case GB.

Table 11. SPARQL query for checking the storage capacity in the PaaSport semantic model (GB and MB).

```
SELECT ?offering ?Value WHERE {
      ?offering rdf:type  paasport:Offering .
      ?offering DUL:satisfies ?gd .
      ?gd  paasport:offers ?characteristic .
      ?characteristic rdf:type paasport:Storage  .
      ?characteristic DUL:hasParameter ?par .
      ?par rdf:type paasport:StorageCapacity .
      ?par DUL:hasParameterDataValue ?Value .
      ?par DUL:parametrizes ?qualityVal .
      ?qualityVal  uomvocab:measuredIn ?Units .
      ?Units rdf:type ucum:UnitOf-infotech .
      ucum:GB rdf:type uomvocab:SimpleDerivedUnit .
      ?Units rdf:type uomvocab:SimpleDerivedUnit .
      ?Units uomvocab:derivesFrom ucum:byte .
      ucum:GB uomvocab:modifierPrefix ?prefix1 .
      ?prefix1 uomvocab:factor ?Factor1 .
      ?Units uomvocab:modifierPrefix ?prefix2 .
      ?prefix2 uomvocab:factor ?Factor2 .
      FILTER ( ?Factor2*?Value >= ?Factor1*2 )
}
```

In the case of the Cloud4SOA-like semantic model, the corresponding SPARQL queries (**Table 13**, **Table 14**) navigate the graph of **Figure 28** (right). The first of the two queries needs a UNION in order to retrieve offerings both in GB and MB.

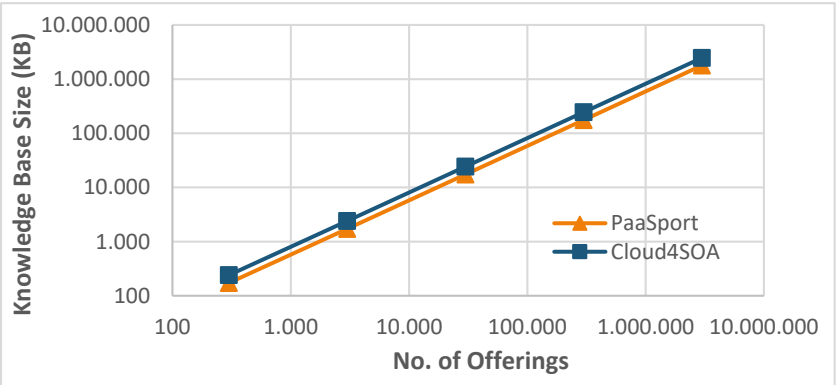

Figure 29. Scaling of offerings knowledge bases size.

[11] http://graphdb.ontotext.com/

Table 12. SPARQL query response times (in sec).

	300	3K	30K	300K	3M
PaaSport (GB-MB)	0.028	0.117	1.025	10.589	110.158
PaaSport (GB only)	0.022	0.068	0.522	5.162	52.790
Cloud4SOA (GB-MB)	0.026	0.063	0.454	4.471	44.855
Cloud4SOA (GB only)	0.019	0.058	0.433	4.200	43.103

Table 12 includes the response times of all queries executed over all repositories. Each query was executed 30 times on a PC with i7 at 3.4 GHz CPU, 16 GB main memory and the average time is reported. **Figure 30** shows how the queries scale over the size of the triplestore (log-log scale). Results show that the query response time of the PaaSport semantic model with both GB and MB is approximately double compared to the other three types of models at most of the knowledge base sizes, except the "small" one with 300 offerings. This was expected since the PaaSport semantic model with alternative measurement units is more complex than the rest of the settings, therefore a more complex graph pattern is needed to retrieve the correct information. This is evident by just looking at the 4 different SPARQL queries. However, the burden of having alternative measurement units is not very big and the scaling of query performance of all model types is almost similar. In case only the GB measurement unit is involved, the comparison between the query performance of the PaaSport model and the Cloud4SOA-like model is even better (only ~22% worse).

From the above results it is evident that the performance of the PaaSport model is inferior to that of a Cloud4SOA-like model, even if not significantly. Therefore, we should justify what the PaaSport model is better at. Notice that in all queries, the resources and properties indicated in bold are specific to the storage capacity request and need to be filled-in the query template for each different application request submitted by the user. These request-specific elements can be easily derived from the application request in the case of the PaaSport semantic model (**Table 11**, **Table 12**), whereas the rest of the query remains intact. This is due to the fact that we have developed the PaaSport ontology as an extension of the DUL upper ontology, as we elaborated in section 4.2. However, this is not the case for the queries that use the Cloud4SOA-like model. The parts of the queries in **Table 13** and **Table 14** that are highlighted need to be manually edited by the user each time a different characteristic is used as

Table 13. SPARQL query for checking the storage capacity in the PaaSport semantic model (only GB).

```
SELECT ?offering ?Value WHERE {
    ?offering rdf:type paasport:Offering .
    ?offering DUL:satisfies ?gd .
    ?gd  paasport:offers ?characteristic .
    ?characteristic rdf:type paasport:Storage  .
    ?characteristic DUL:hasParameter ?par .
    ?par rdf:type paasport:StorageCapacity .
    ?par DUL:hasParameterDataValue ?Value .
    ?par DUL:parametrizes paasport:Storage_capacity_GB .
    FILTER ( ?Value >= 2 )
}
```

Table 14. SPARQL query for checking the storage capacity in the Cloud4SOA-like semantic model (GB and MB).

```
SELECT ?offering ?Value WHERE {
    ?offering rdf:type  c4s:Offering .
    ?offering c4s:offerStorage ?s.
    ?s c4s:hasStorageConfiguration ?sc .
    ?sc c4s:hasStorageCapacity ?par .
    ?par c4s:hasMaxStorageValue ?qualityValue .
    ?qualityValue c4s:hasValue ?Value .
    { ?qualityValue rdf:type c4s:GigaByte .
      FILTER( ?Value >= 2 )
    } UNION
    { ?qualityValue rdf:type c4s:MegaByte .
      FILTER ( ?Value >= 2048 )
    }
}
```

Table 15. SPARQL query for checking the storage capacity in the Cloud4SOA-like semantic model (only GB).

```
SELECT ?offering ?Value WHERE {
    ?offering rdf:type  c4s:Offering .
    ?offering c4s:offerStorage ?s.
    ?s c4s:hasStorageConfiguration ?sc .
    ?sc c4s:hasStorageCapacity ?par .
    ?par c4s:hasMaxStorageValue ?qualityValue .
    ?par c4s:hasMaxStorageValue ?qualityValue .
    ?qualityValue paasport:hasValue ?Value .
    ?qualityValue rdf:type c4s:GigaByte .
    FILTER ( ?Value >= 2 )
}
```

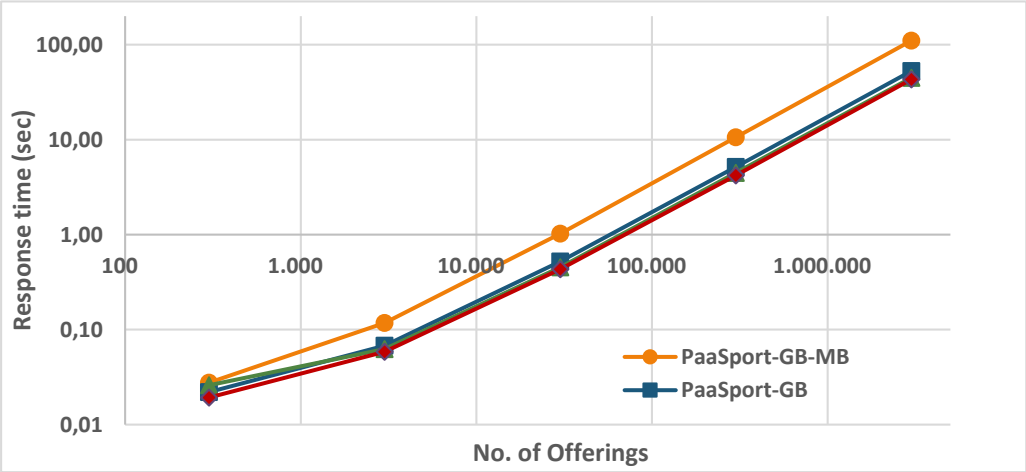

Figure 30. Scaling of query response times.

an application request. This is due to the fact that the Cloud4SOA ontology does not follow a regular structure as the PaaSport ontology and the number, the names and the semantics of classes and properties of platform characteristics vary in an ad-hoc manner. This discussion also includes measurement units. For example, in **Table 13** the transformation of the limit of 2GB to 20148MB needs to be done manually by the user, simply because there are no basic and derived units in Cloud4SOA and there is no way to automate the conversion between them. In case there are more alternative measurement units in the knowledge base, the query should re manually reformulated with more UNIONs, so there is a need from the user to know the contents of the knowledge base before submitting the query. In contrast, the query for the PaaSport model in **Table 11** works in any case of measurement units and for every measurable platform characteristic without any need for manual intervention. Concluding, PaaSport trades flexibility and generality for performance. Since the number of PaaS offerings in a typical installation of the PaaSport marketplace is not expected to exceed the order of few thousands, we believe that this trade-off pays off.

7 Conclusions and Future Work

The PaaSport project tries to avoid the Cloud provider lock-in problem that many software SMEs are having, by (a) enabling platform provider SMEs to roll out semantically interoperable PaaS offerings, and, (b) facilitating software SMEs to seamlessly deploy (or migrate) business applications on the best-matching Cloud PaaS offering. To this end, PaaSport combined Cloud PaaS technologies with lightweight semantics in order to specify and deliver a thin, non-intrusive Cloud broker, in the form of a Cloud PaaS Marketplace.

In this paper, we have presented the semantical aspects of the PaaSport Cloud broker / marketplace, focusing on an OWL ontology we have developed. The PaaSport ontology represents the necessary Platform-as-a-Service characteristics and attributes for semantically annotating: (a) capabilities of PaaS offerings, (b) requirements of applications to be deployed on one of the Cloud platform offerings, through a Cloud Broker, and (c) Service Level Agreements to be established between offering providers and application owners. The ontology has been designed in order to efficiently support a semantic matchmaking and ranking algorithm for recommending the best-matching Cloud PaaS offering to the application developer, which uses SPARQL queries for retrieving relevant data from the semantic repository.

The PaaSport ontology has been defined as an extension of the DOLCE+DnS Ultralight (DUL) ontology design pattern [26]. This offers extensibility, since both PaaS characteristics and parameters are defined as classes, so extending the ontology with new characteristics requires just adding new classes as subclasses of existing ones, which is less complicated than adding properties. This extensibility advantage reflects also on the persistence layer of the PaaSport marketplace that is based on a relational database system and it can be easily extended, as the semantic

model evolves, without the need to change existing tables. Finally, this feature also proves to be advantageous in terms querying the knowledge base to retrieve PaaS offerings compatible with application requests. Specifically, general templates can be used to pose queries on any PaaS characteristic requested, including also arbitrary measurement units. This flexibility comes at a small performance price.

Future development plans for the PaaS ontology include its extension with the representation of intricate PaaS pricing models and plans. Furthermore, a transformation methodology between these models is needed, so that these models can be comparable and useful in the decision-making about the Cloud platform to deploy an application, in addition to the recommendation algorithm [5]. This transformation can be based rules of inference, e.g. SWRL (Semantic Web Rule Language) [33] or SPIN (SPARQL Inferencing Notation) [35]. Finally, an interesting research direction would be to integrate the recommendation algorithm within the ontology in the form of SPIN / SPARQL [35] rules and constraints, since the algorithm itself is mostly based on SPARQL query templates. In this way, the business logic of PaaSport recommendation would be integrated with the ontology itself, making it transparent, modifiable, extensible and portable.

Acknowledgments

This work is fully funded by the EU FP7-SME-2013-2-605193 PaaSport project. The authors would like also to thank our project partners Giannis Ledakis, Andreas Papadopoulos, Demetris Trihinas, George Pallis, Gerald Hübsch, Fatemeh Ahmadi Zeleti and Lukasz Porwol for their valuable comments.

References

[1] Androcec, D., Vrcek, N., Seva, J. Cloud Computing Ontologies: A Systematic Review. Proc. 3rd Int. Conf. on Models and Ontology-based Design of Protocols, Architectures and Services (MOPAS 2012), pp. 9-14, 2012.

[2] Androcec, D.; Vrcek, N.; Kungas, P., "Service-Level Interoperability Issues of Platform as a Service," 2015 IEEE World Congress on Services (SERVICES), pp.349-356, June 27 2015-July 2 2015.

[3] Apache Jena, https://jena.apache.org/ (last checked on 06-Jan-2016)

[4] ARTIST FP7 project: http://www.artist-project.eu (last checked on 06-Jan-2016)

[5] Bassiliades N., Symeonidis M., Meditskos G., Kontopoulos E., Gouvas P., Vlahavas I., A Semantic Recommendation Algorithm for the PaaSport Platform-as-a-Service Marketplace, Expert Systems with Applications, *accepted for publication*, http://dx.doi.org/10.1016/j.eswa.2016.09.032.

[6] Borenstein, N.; Blake, J., "Cloud Computing Standards: Where's the Beef?," in Internet Computing, IEEE , vol.15, no.3, pp.74-78, May-June 2011

[7] Brank, J., Grobelnik, M. and Mladenic, D. (2005), A survey of Ontology Evaluation Techniques, in Proceedings of the Conference on Data Mining and Data Warehouses (SiKDD 2005), Ljubljana, Slovenia.

[8] Carvalho L., Mahowald R. P., McGrath B., Fleming M., Hilwa A. Worldwide Competitive Public Cloud Platform as a Service Forecast, 2015–2019. Jul 2015. Doc # 257391. Market Forecast. Available at: https://www.idc.com/getdoc.jsp?containerId=257391 (last accessed: January 2016)

[9] Chen F., Bai X., and Liu B. Efficient Service Discovery for Cloud Computing Environments. *Advanced Research on Computer Science and Information Engineering*, G. Shen and X. Huang (Eds.), Springer Berlin Heidelberg, 2011, pp. 443-448.

[10] Cloud Application Management for Platforms Version 1.1. Edited by Jacques Durand, Adrian Otto, Gilbert Pilz, and Tom Rutt. 09 November 2014. OASIS Committee Specification. http://docs.oasis-open.org/camp/camp-spec/v1.1/camp-spec-v1.1.html.

[11] Cloud Service Measurement Index Consortium: http://csmic.org/ (last checked on 06-Jan-2016)

[12] Cloud4SOA FP7 project: http://www.cloud4soa.com (last checked on 05-June-2016)

[13] Columbus L., 451 Research: Platform-as-a-Service (PaaS) Fastest Growing Area of Cloud Computing, Available at: http://www.forbes.com/sites/louiscolumbus/2013/08/20/451-research-platform-as-a-service-paas-fastest-growing-area-of-cloud-computing/

[14] Compton Michael, Barnaghi Payam, Bermudez Luis, García-Castro Raúl, Corcho Oscar, Cox Simon, Graybeal John, Hauswirth Manfred, Henson Cory, Herzog Arthur, Huang Vincent, Janowicz Krzysztof, Kelsey W. David, Le Phuoc Danh, Lefort Laurent, Leggieri Myriam, Neuhaus Holger, Nikolov Andriy, Page Kevin, Passant Alexandre, Sheth Amit, Taylor Kerry, The SSN ontology of the W3C semantic sensor network incubator group, Web Semantics: Science, Services and Agents on the World Wide Web, Vol. 17, 2012, pp. 25-32.

[15] D2RQ Platform, http://d2rq.org/ (last checked on 06-Jan-2016)

[16] Dastjerdi A. V., Tabatabaei S. G. H., and Buyya R. An Effective Architecture for Automated Appliance Management System Applying Ontology-Based Cloud Discovery. *Proc. 10th IEEE/ACM Int. Conf. on Cluster, Cloud and Grid Computing*, pp. 104-112, May 2010.

[17] Deng Y., Head M. R., Kochut A., Munson J., Sailer A., and Shaikh H. Introducing Semantics to Cloud Services Catalogs. Proc. 2011 IEEE Int. Conf. on Services Computing, pp. 24-31, July 2011.

[18] Dukaric R., Juric M. B., Towards a unified taxonomy and architecture of cloud frameworks, Future Generation Computer Systems, 29(5), 2013, pp. 1196-1210.

[19] European Commission, Unleashing the Potential of Cloud Computing in Europe, Communication From The Commission To The European Parliament, The Council, The European Economic And Social Committee And The Committee Of The Regions, Brussels, COM(2012) 529, 27/9/2012. In: http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=COM:2012:0529:FIN:EN:PDF

[20] Flahive A., Taniar D., and Rahayu W. Ontology as a Service (OaaS): A Case for Sub-ontology Merging on the Cloud. Journal of Supercomputing, pp. 1-32, October 2011.
[21] Fortis TF, Munteanu VI, Negru V (2012) Towards an ontology for cloud services. In: 6th International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS 2012). IEEE Computer Society Press, Washington, pp 787–792.
[22] Foster I, Zhao Y, Raicu I and Lu S 2008 Cloud Computing and Grid Computing 360-Degree Compared. In: Proceedings of the IEEE Grid Computing Environments Workshop, pp 1-10
[23] Gagliardi F and Muscella S 2010 Cloud computing: data confidentiality and interoperability challenges. In: Antonopoulos N, Gillam L, editors. Cloud Computing: Principles, Systems and Applications (Computer Communications and Networks). London: Springer; 2010:257-270.
[24] Gangemi A., Presutti V. Ontology Design Patterns, in Staab S. et al. (eds.): Handbook of Ontologies (2nd edition), Springer, 2009.
[25] Gangemi, A., Catenacci, C., Ciaramita, M. and Lehmann, J. (2005), A theoretical framework for ontology evaluation and validation, Proceedings of the Semantic Web Applications and Perspectives (SWAP) – 2nd Italian Semantic Web Workshop, Trento, Italy.
[26] Gangemi, A., Mika, P. Understanding the semantic web through descriptions and situations. In: Proceedings of the International Conference on Ontologies, Databases and Applications of Semantics. pp. 689-706 (2003).
[27] García, J., García-Peñalvo, F. J., and Therón, R. (2010), A Survey on Ontology Metrics. Knowledge Management, Information Systems, E-Learning, and Sustainability Research, Communications in Computer and Information Science, Vol. 111, pp. 22-27.
[28] Gardner D. 2010 Cloud computing's ultimate value depends on open PaaS models to avoid applications and data lock-in. In: http://www.zdnet.com/blog/gardner/cloud-computings-ultimate-value-depends-on-open-paas-models-to-avoidapplications-and-data-lock-in/3794
[29] Garg S. K., Versteeg S., Rajkumar B., A framework for ranking of cloud computing services, Future Generation Computer Systems, Volume 29, Issue 4, June 2013, Pages 1012-1023, ISSN 0167-739X, http://dx.doi.org/10.1016/j.future.2012.06.006.
[30] Han T., Sim K. M. An Ontology-enhanced Cloud Service Discovery System. *Proc. Int. MultiConference of Engineers and Computer Scientists 2010*, Vol I, March 2010.
[31] He K.-Q., Wang J., and Liang P. Semantic Interoperability Aggregation in Service Requirements Refinement. *Journal of Computer Science and Technology*, vol. 25, pp. 1103-1117, November 2010.
[32] Hibernate ORM, http://hibernate.org/
[33] Horrocks Ian, Patel-Schneider Peter F., Boley Harold, Tabet Said, Grosof Benjamin, Dean Mike, SWRL: A Semantic Web Rule Language Combining OWL and RuleML, W3C Member Submission 21 May 2004. Available at: http://www.w3.org/Submission/SWRL/
[34] Kamateri, E., Loutas, N., Zeginis, N., Ahtes, J., D'Andria, F., Bocconi, S., Gouvas, P., Ledakis, G., Ravagli, F., Lobunets, O. & Tarabanis, K. (2013) Cloud4SOA: A semantic-interoperability PaaS solution for multi-Cloud platform management and portability. In Service-Oriented and Cloud Computing, Lecture Notes in Computer Science Volume 8135, 2013, pp 64-78.
[35] Knublauch H., Hendler J. A., Idehen K., SPIN - Overview and Motivation, W3C Member Submission 22 February 2011. Available at: https://www.w3.org/Submission/spin-overview/
[36] Kourtesis D., Alvarez-Rodríguez J. M., Paraskakis I., Semantic-based QoS management in cloud systems: Current status and future challenges, Future Generation Computer Systems, Vol. 32, 2014, pp. 307-323.
[37] Langer P., Mayerhofer T., Kappel G., "Semantic Model Differencing Utilizing Behavioral Semantics Specifications", Proceedings 17th International Conference, MODELS 2014, Model-Driven Engineering Languages and Systems, Volume 8767 of the series Lecture Notes in Computer Science, pp. 116-132, Springer International Publishing, Valencia, Spain, 2014.
[38] Ma Y. B., Jang S. H., and Lee J. S. Ontology-Based Resource Management for Cloud Computing. Intelligent Information and Database Systems, N. Nguyen, C.-G. Kim, and A. Janiak (Eds.), Springer Berlin Heidelberg, 2011, pp. 343-352.
[39] Martinez C. A., Echeverri G. I., and Sanz A. G. C. Malware detection based on Cloud Computing integrating Intrusion Ontology Representation. Proc. 2010 IEEE Latin-American Conference on Communications (LATINCOM), pp. 1-6, September 2010.
[40] Mell Peter, Grance Timothy, "The NIST Definition of Cloud Computing", NIST Special Publication 800-145, September 2011. Available at: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf
[41] mOSAIC FP7 project: http://www.mosaic-cloud.eu (last checked on 06-Jan-2016)
[42] Moscato F., Aversa R., Di Martino B., Fortis T.-F., and Munteanu V. An Analysis of mOSAIC ontology for Cloud Resources Annotation. Proc. Federated Conf. on Computer Science and Information Systems, pp. 983-990, 2011.
[43] OntOlogy Pitfall Scanner!, http://oops.linkeddata.es/ (last checked on 06-Jan-2016)
[44] OpenCrowd. *Cloud Computing Vendors Taxonomy*. Available at: http://cloudtaxonomy.opencrowd.com/, last access: December 2015.
[45] PaaSport Consortium, Deliverable 1.1: PaaSport Requirements Analysis Report, November 2014. Available at: http://paasport-project.eu/deliverables.php
[46] PaaSport Consortium, Deliverable 1.2: PaaSport Reference Architecture, October 2014. Available at: http://paasport-project.eu/deliverables.php
[47] PaaSport Consortium, Deliverable 1.3: PaaSport Semantic Models, November 2014. Available at: http://paasport-project.eu/deliverables.php
[48] PaaSport Deliverable D4.2: "Persistence and Execution Layer – First Release", PaaSport project, FP7-605193. Available at: http://paasport-project.eu/deliverables.php
[49] PaaSPort FP7 project: http://paasport-project.eu (last checked on 06-Jan-2016)
[50] Petcu D., Di Martino B., Venticinque S., Rak M., Máhr T., Lopez G. E., Brito F., Cossu R., Stopar M., Šperka S. and Stankovski V., Experiences in building a mOSAIC of clouds, Journal of Cloud Computing: Advances, Systems and Applications, vol. 2 (1), p. 12, 2013.
[51] Poveda-Villalón, M., Gómez-Pérez, A., & Suárez-Figueroa, M. C. (2014). OOPS! (OntOlogy Pitfall Scanner!): An On-line Tool for Ontology Evaluation. International Journal on Semantic Web and Information Systems (IJSWIS), 10(2), 7-34. doi:10.4018/ijswis.2014040102
[52] Protégé, http://protege.stanford.edu/ (last checked on 06-Jan-2016)
[53] REMICS FP7 project: http://www.remics.eu (last checked on 06-Jan-2016)
[54] Rimal B P, Jukan A, Katsaros D and Goeleven Y. 2011. Architectural Requirements for Cloud Computing Systems: An Enterprise Cloud Approach. Journal of Grid Computing. 9. 3-26
[55] Rodríguez-García M. A., Valencia-García R., García-Sánchez F., Samper-Zapater J. J., Creating a semantically-enhanced cloud services environment through ontology evolution, Future

Generation Computer Systems, Vol. 32, 2014, pp. 295-306, 2014.
[56] Spring Data, http://projects.spring.io/spring-data/
[57] Spring Framework, https://projects.spring.io/spring-framework/
[58] Takahashi T., Kadobayashi Y., and Fujiwara H. Ontological Approach toward Cyber-security in Cloud Computing. Proc. 3rd Int. Conf. on Security of Information and Networks, pp. 100-109, September 2010.
[59] Topology and Orchestration Specification for Cloud Applications, Version 1.0. 25 November 2013. OASIS Standard. http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html.
[60] TopQuadrant, TopBraid Platform Overview, in: http://www.topquadrant.com/technology/topbraid-platform-overview/
[61] Transparency Market Research, Platform as a Service (PaaS) (Public, Private and Hybrid Cloud) Market - Global Industry Analysis, Size, Share, Growth, Trends and Forecast 2014 – 2020. Dec 2014. Available at: http://www.transparencymarketresearch.com/platform-as-a-service.html
[62] Tsai, W.-T., Sun, X., Balasooriya, J., 2010. Service-oriented cloud computing architecture. In: 2010 Seventh International Conference on Information Technology New Generations, pp. 684–689.
[63] Vrandečić, D. (2009), Ontology Evaluation, Handbook on Ontologies, International Handbooks on Information Systems, pp. 293-313.
[64] Wang J., Zhang J., Hung P. C. K., Li Z., Liu J., He K. Leveraging Fragmental Semantic Data to Enhance Services Discovery. Proc. 2011 IEEE Int. Conf. on High Performance Computing and Communications, pp. 687-694, September 2011.
[65] Weinhardt C., Anandasivam A., Blau B., and Stosser J. Business Models in the Service World, IT Professional, vol. 11, pp. 28-33, March-April 2009.
[66] Youseff L., Butrico M., and Da Silva D. Toward a Unified Ontology of Cloud Computing. Proc. Grid Computing Environments Workshop (GCE '08), pp. 1-10, November 2008.