Cloud Service Brokerage: Strengthening Service Resilience in Cloud-Based Virtual Enterprises

Simeon Veloudis, Iraklis Paraskakis, Christos Petsos

Cloud Service Brokerage: Strengthening Service Resilience in Cloud-Based Virtual Enterprises

Simeon Veloudis, Iraklis Paraskakis, and Christos Petsos

South East European Research Centre (SEERC), International Faculty of the University of Sheffield, CITY College, 24 Proxenou Koromila St, 54622, Thessaloniki, Greece

{sveloudis, iparaskakis, chrpetsos}@seerc.org

Abstract. We argue that the incorporation of cloud service brokerage (CSB) mechanisms will strengthen the resilience of services in cloud-based VEs. In this respect, we present the Service Completeness-Compliance Checker (SC^3), a mechanism which offers capabilities with respect to the Quality Assurance dimension of CSB. More specifically, SC^3 strengthens the resilience of cloud services by evaluating their compliance with pre-specified policies concerning the business aspects of their delivery. By relying on an ontology-based representation of policies and services, SC^3 achieves a proper separation of concerns between policy definition and policy enforcement. This effectively enables SC^3 to perform policy evaluation in a manner generic and agnostic to the underlying cloud delivery platform utilised by a cloud-based VE.

Keywords: virtual enterprises, cloud computing, cloud service brokerage, governance, quality control, ontologies, Linked USDL

1 Introduction

Cloud computing has evolved out of Grid computing [1,2] as a result of a shift in focus from an infrastructure aiming to deliver mainly storage and compute resources, to an economy-based computing paradigm aiming to deliver a wide range of resources abstracted as services [2]. Such a shift is anticipated to impact the manner in which businesses and organisations share skills and core competencies within a distributed collaborative network [3]. More specifically, activities performed by a dynamic multi-institutional virtual enterprise (VE) may involve the use of heterogeneous, externally-sourced cloud services which span different clouds and capability levels (IaaS, PaaS, and SaaS) [4], and which are entrusted by their users with data, software, and computation; we shall term such a VE a *cloud-based* one.

As an example, consider the following scenario. An industrial consortium is formed to collaboratively process the data produced as part of a seismic survey. Such processing integrates software components, offered as a service, by different consortium participants (SaaS offerings). Each component may be operating on a participant's proprietary infrastructure or, alternatively, on infrastructure provisioned as a cloud service (IaaS offering). At the same time, the simulation requires the develop-

ment of new specialised software components. To this end, the consortium is provisioned the necessary software platform for developing these applications as a service (PaaS offering).

Evidently, the IT environment of a cloud-based VE is transformed into a complex ecosystem of intertwined infrastructure, platform, and application services delivered remotely, over the Internet, by diverse service providers. As the number of services proliferates, it becomes increasingly difficult to keep track of when and how they evolve over time, either through intentional changes, initiated by their providers, or through unintentional changes, such as variations in their performance and availability. Moreover, it becomes increasingly difficult to accurately predict the potential repercussions that such an evolution has with respect to a service's compliance to policies and regulations, and its conformance to service level agreements (SLAs).

In order to strengthen the resilience of the services in such an ecosystem, cloud-based VEs are anticipated to increasingly rely on *cloud service brokerage* (CSB) [5]. In this respect, the work in [3] proposed a conceptual architecture of a framework which offers capabilities with respect to two dimensions of CSB, namely *Quality Assurance Service Brokerage*, and *Service Customisation Brokerage*. These capabilities revolve around the following themes: (i) *governance and quality control*; (ii) *failure prevention and recovery*, and (iii) *optimisation*. The 1st theme is concerned with checking the compliance of services with a set of policies and regulations; it is also concerned with testing services for conformance with their expected behaviour. The 2nd theme is concerned with the reactive and proactive detection of service failures, and the selection of suitable strategies to prevent, or recover from, such failures. The 3rd theme is concerned with continuously identifying opportunities to optimise service consumption.

Continuing the work in [3], this paper reports on the implementation of a particular mechanism of the aforementioned framework, namely the Service Completeness-Compliance Checker (SC^3), which offers capabilities with respect to the *governance and quality control* theme. More specifically, SC^3 strengthens the resilience of cloud services by continuously evaluating their compliance with pre-specified policies concerning their business aspects of delivery. By relying on a *declarative* representation of policies and services, one which is based on an RDF(S) ontology, SC^3 achieves a clear separation of concerns: policies are represented independently of the code that SC^3 employs for enforcing them. SC^3 is thereby kept generic and orthogonal to any underlying cloud delivery platform employed by a cloud-based VE. Such a separation of concerns is generally absent in contemporary governance mechanisms [6], with negative repercussions on their portability, as well as on their ability to represent and reason about, policy interrelations.

The rest of this paper is structured as follows. Section 2 presents a motivating scenario. Section 3 outlines our declarative approach to policy representation and explains how SC^3 extracts from this representation the necessary information for the subsequent policy enforcement process. Section 4 describes this process, and Section 5 outlines related work. Finally, Section 6 presents conclusions and future work.

2 Motivating Scenario

[3] proposed a conceptual architecture of a CSB framework offering capabilities spanning the main phases of a service's lifecycle, namely Service On-boarding, Service Operation, and Service Evolution. SC^3 offers capabilities with respect to the *Service On-boarding* phase[1]. Below we identify these capabilities through the prism of the example of Section 1.

Let CPx (stands for Cloud Platform x) be a cloud delivery platform that hosts various services that are potentially used by the industrial consortium. The platform houses a variety of apps developed by CPx's network of ecosystem partners. CPx also allows advanced users to develop and deploy custom applications on the platform, and to create rich compositions of applications (mash-ups) offered by third-party service providers.

Table 1. Entry-level criteria

Service-level Attribute	Acceptable Values	SLO	Comments
`storage`	[100,1000)	Gold storage	Size in TB
	[10,100)	Silver storage	
	[0,10)	Bronze storage	
`availability`	[0.99999,1)	Gold availability	Total uptime ratio
	[0.9999,1)	Silver availability	
	[0.999,1)	Bronze availability	
`encryption`	256	Gold encryption	Key-length in bits
	192	Silver encryption	
	128	Bronze encryption	

Suppose that an ecosystem partner offers a new service on CPx, call it StoreCloud, which provides an encrypted and versioned persistence layer for storing the results during the various phases of the seismic survey. In order for the new service to be on-boarded on CPx, a number of entry-level criteria must be satisfied. These crucially capture a set of *service-level objectives* (SLOs) expressed in terms of restrictions on relevant *service-level attributes* (see Table 1). These SLOs essentially form CPx's *business* (or *broker*[2]) *policy* (BP) with respect to on-boarding StoreCloud.

We assume that the ecosystem partner who offers StoreCloud, hereafter referred to as the *service provider* (SP), submits a service description (SD) which details the manner in which StoreCloud is to be deployed on CPx. The SC^3 mechanism offers a

[1] SC^3 also offers capabilities with respect to the Service Operation phase and, in particular, with respect to continuously monitoring the behaviour of a service. These capabilities shall not, however, concern us in this paper.

[2] We use the term "*broker*" to emphasise that, in our work, such a business policy is formulated according to the declarative approach of our *brokerage* framework (see Section 3.3).

policy evaluation capability which essentially allows the cloud-based VE to determine whether this SD is compliant with CPx's BP. Such a capability entails two kinds of evaluation: SD *completeness* evaluation and SD c*ompliance* evaluation. The former kind of evaluation aims at determining whether the SD specifies values for *all* required service-level attributes. For example, an SD which does not specify a value for the `encryption` attribute cannot be considered complete. The latter kind of evaluation aims at determining whether the specified attribute values fall within the corresponding ranges prescribed in the BP. For example, an SD which specifies a 64-bit value for the `encryption` attribute cannot be considered compliant. Clearly, the aforementioned evaluations seek to determine whether StoreCloud attains the SLOs specified in CPx's BP. In this respect, they strengthen the service's resilience.

3 Parsing BPs: The SC^3 Approach

This section describes how the SC^3 mechanism parses a BP in order to extract the necessary information for evaluating the completeness and compliance of SDs. A brief description of a conceptual architecture for SC^3 is, however, first in order.

3.1 Conceptual Architecture

As depicted in Figure 1, the SP submits StoreCloud's SD through the SP-facing component – an interface which exposes an editor for facilitating the construction of the SD. The SD is then transported to the SC^3 mechanism, and also stored in the Governance Registry (GReg) depicted in Figure 1; the transportation takes place through a Publish/subscribe (Pub/sub) system. An explanation of the reasons for opting for the open-source WSO_2 Carbon platform [7] (see Figure 1), as well as for advocating a Pub/sub system for transporting SDs, is omitted here due to space limitations; a relevant discussion can be found in [8,9].

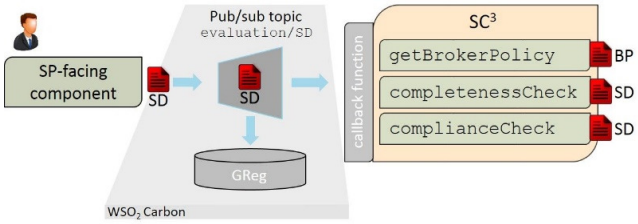

Fig. 1. SC3 conceptual architecture

The SC^3 exposes a callback function for subscribing to the appropriate topic of the Pub/sub system and (asynchronously) receiving the SD. More specifically, this callback function exposes the `EvaluationComponentSDSubscriber` class which is responsible for orchestrating all the actions required for creating connections to the Pub/sub system and subscribing to its topics. This class triggers the SC^3 mecha-

nism when a fresh SD arrives. In particular, it invokes an object of the class `PolicyCompletenessCompliance`, one which is parameterised with the appropriate BP against which the evaluation will take place. The `PolicyCompletenessCompliance` class is one of the core classes of the SC^3 mechanism. It offers three main methods: `getBrokerPolicy`, `completenessCheck`, and `complianceCheck`. The first method extracts all the required information from the BP for the subsequent SD completeness and compliance evaluations to take place (see Section 3.2), whilst the last two implement these evaluations (see Section 4). All three methods are implemented in Java using the Apache Jena (Core and ARQ) APIs [10].

3.2 The `getBrokerPolicy` process

The `getBrokerPolicy` process parses the BP and places the information that it extracts in the `bp` object of the class `BrokerPolicy`. This class encompasses a number of Java HashMap objects as attributes; Table 2 depicts these attributes for the scenario of Section 2. The HashMap objects reflect our *declarative framework* for representing the SLOs incorporated in the BP and, effectively, the BP itself. As already mentioned, through the incorporation of this framework, SC^3 achieves a clear separation of concerns between policy representation and policy enforcement: BPs are represented independently of the code that SC^3 employs for enforcing them. SC^3 is thus kept generic and orthogonal to the underlying cloud delivery platform.

The declarative framework is based upon Linked USDL [11] and, in particular, upon Linked USDL's SLA schema. Linked USDL is a lightweight ontology which provides an RDF vocabulary for the description of the business aspects of policies and services; it draws upon a number of widely-adopted vocabularies such as GoodRelations, SKOS, and FOAF. The reasons for opting for Linked USDL are outlined in Section 5.1; a more complete discussion can be found in [12]. A brief account of our framework is in order; such an account is necessary here for understanding the implementation of `getBrokerPolicy`. The account is based on the scenario of Section 2.

Table 2. HashMap objects

```
Map<String, BrokerPolicyClass> serviceModelMap;
Map<String, BrokerPolicyClass> serviceLevelProfileMap;
Map<String, BrokerPolicyClass> serviceLevelMap;
Map<String, BrokerPolicyClass> serviceLevelExpressionMap;
Map<String, BrokerPolicyClass> expressionVariableMap;
Map<String, BrokerPolicyClass> quantitativeValueFloatMap;
```

3.3 Declarative Representation of BPs and SLOs

We model the SLOs of a BP, hence the BP itself, through a *specialisation process* which constructs a framework of suitable subclasses and sub-properties of the Linked USDL SLA classes and properties depicted in Figure 2. These subclasses are then

populated by instances specified in StoreCloud's SD. Below we outline this process for the 'gold' SLO of StoreCloud's `availability` attribute[3] (see Section 2); a more complete account of this process can be found in [9,12].

SLO Representation.
For each service-level attribute, the BP offers a subclass of the class `ServiceLevel` for accommodating the attribute's SLOs. For example, for accommodating the SLOs of the `availability` attribute (i.e. the 'gold', 'silver', and 'bronze' SLOs of Table 1), it offers the class `SL-Availability` (see Figure 2). SLOs appear as instances of this class – e.g., the `SL-GoldAvailability` instance specified in StoreCloud's SD (see Figure 2).

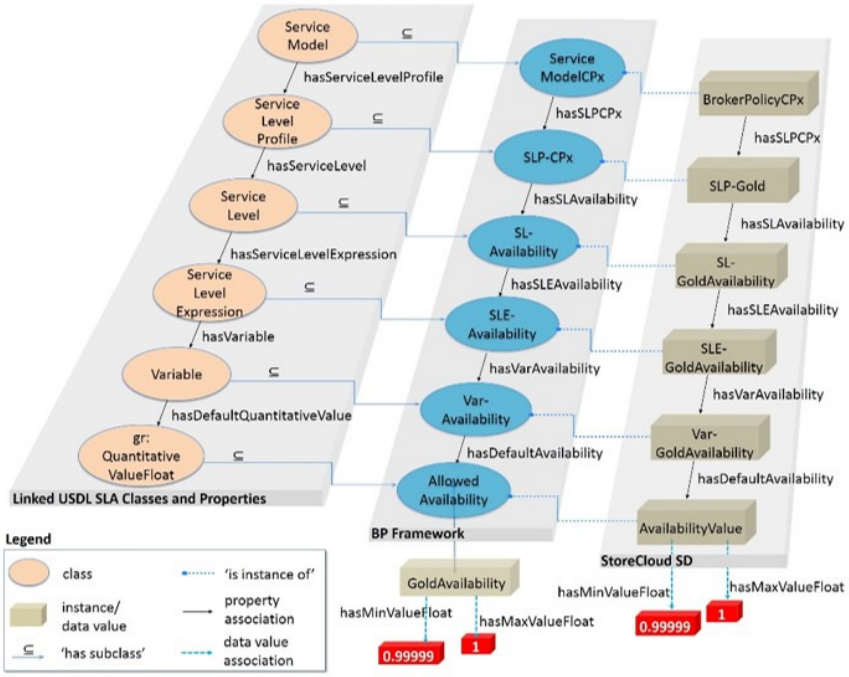

Fig. 2. Linked USDL SLA for the gold availability SLO

Each SLO is defined in terms of a *service-level expression* (SLE) which specifies the conditions that must be satisfied in order for the SLO to be met. SLEs are modelled as instances of suitable subclasses of the `ServiceLevelExpression` class (see Figure 2). For example, the SLEs that correspond to the availability-related SLOs are modelled as instances of the class `SLE-Availability`. SLOs are associated with their corresponding SLEs through appropriate sub-properties of the

[3] Of course, an analogous account applies to the rest of the attributes and SLOs of Table 1.

hasServiceLevelExpression property. In particular, the availability-related SLOs are associated with their SLEs through the hasSLEAvailability property (see Figure 2).

Each SLE binds a *variable* that corresponds to a particular attribute, one which is associated with an allowable range of values. Following an approach entirely symmetrical to the one outlined above for SLOs and SLEs, variables are modelled as instances of appropriate subclacess of the class Variable, whilst value ranges are modelled as instances of appropriate subclasses of the GoodRelations class QuantitativeValueFloat [4]. Figure 2 depicts these subclasses (Var-Availability, AllowedAvailability), and instances (Var-GoldAvailability, AvailabilityValue), for the 'gold' availability attribute. SLEs are associated with their corresponding variables through sub-properties of the hasVariable property (e.g. the hasVarAvailability of Figure 2), and variables are associated with their allowable values through sub-properties of the hasDefaultQuantitativeValue property[5].

Service-level Profiles and BP Representation.
Above we outlined a framework for representing SLOs. We now show how a BP is associated with this framework, hence with its pertinent SLOs[6]. A BP is represented as an instance of a subclass of the ServiceModel class, e.g. as an instance of the class ServiceModelCPx of Figure 2. This instance is associated with its SLOs through one or more *service-level profiles* (SLPs). SLPs are groupings of SLOs whose purpose is to formulate different service deployment 'packages' that are allowable by the BP. For example, in the scenario of Section 2, the 'gold' SLP formulates a deployment package comprising the 'gold' SLOs of the attributes of Table 1. SLPs take the form of instances of appropriate subclasses of the ServiceLevelProfile class, e.g. the SLP-Gold instance of the SLP-CPx subclass of Figure 2. BPs are associated with SLPs through sub-properties of the hasServiceLevelProfile property (e.g. through hasSLPCPx of Figure 2). SLPs are associated with their constituent SLOs through appropriate sub-properties of the hasServiceLevel property – an example is hasSLAvailability of Figure 2 which associates the 'gold' SLP with the 'gold' availability SLO.

3.4 Extracting Information from BPs

Turning now back to BP parsing, the getBrokerPolicy method sets out to construct a programmatic representation of the BP framework outlined above. In this respect, it starts off by discovering, for each Linked USDL SLA class C, those subclasses S of C that appear in the BP. It then instantiates the HashMap attributes of the bp object (see Table 2) with the corresponding subclasses. This instantiation takes

[4] Or of the class QualitativeValue, in case of qualitative variables.
[5] Or through sub-properties of the hasDefaultQualitativeValue, in case of qualitative values.
[6] Recall from Section 2 that a BP is essentially a set of SLOs.

place through the method `getBrokerPolicyClassMap` as indicated in the 1st row of Table 3 for the `ServiceLevel` class (the rest of the instantiations are entirely analogous and thus omitted). The `getBrokerPolicy` method then proceeds to construct a list of string objects holding the URIs of all `QuantitativeValueFloat` instances found in the BP (e.g. the `GoldAvailability` instance depicted in Figure 2); an excerpt of the code that populates this list is shown in the 2nd row of Table 3.

Table 3. Storing BP subclasses and `QuantitativeValueFloat` instances

```
bp.setServiceLevelMap(getBrokerPolicyClassMap(USDL_SLA,
                                        "ServiceLevel"));
```

```
1. List<String> qvSubclassList = new ArrayList<String>();
2. Iterator<BrokerPolicyClass> iterFl =
   bp.getQuantitativeValueFloatMap().values().iterator();
3. while (iterFl.hasNext()) {
4.   qvSubclassList.add((iterFl.next()).getUri());}
```

Subsequently, the `getBrokerPolicy` method discovers all properties in the BP, along with their corresponding ranges, which have as a domain one of the subclasses *S*; these are effectively all the *sub-properties* that appear in the BP. For each sub-property, an object of the class `Subproperty` is constructed (see Table 4 for an excerpt of the relevant code).

Table 4. Discovering ranges of sub-properties

```
1.while (riIn.hasNext()) {
2.Resource subclassPropertyResource = riIn.next();
3.Subproperty prop =
     new Subpropety(subclassPropertyResource.toString());
4.prop.setDomainUri(subclassResource.toString());
5.Property rangeProperty =
     ResourceFactory.createProperty(RDFS + "range");
6.NodeIterator riIn2 =
     modelMem.listObjectsOfProperty(subclassPropertyResource,ran
       geProperty);
7.prop.setRangeUri(riIn2.next().toString());}
```

4 SD Completeness and Compliance Checking

An SD is effectively a framework of instances that populate the subclasses of the BP framework (see Figure 2). Below we outline the processes that determine whether an SD is *complete* and *compliant* with respect to the BP.

4.1 The `completenessCheck` Method

The `completenessCheck` algorithm starts off by determining whether there is a corresponding instance I_S for each class S discovered by the `getBrokerPolicy` method (see Section 4.2). Then, for each object property P_o discovered by `getBrokerPolicy` such that $\mathrm{dom}(P_o) = S$, it determines if I_S is associated (via P_o) with exactly one instance $I_{S'}$ of the class $S' = \mathrm{ran}(P_o)$. Similarly, for each data property P_d, such that $\mathrm{dom}(P_d) = S$, it determines whether I_S is associated via P_d with a data value from $\mathrm{ran}(P_d)$. An excerpt of the code that checks these associations is shown in Table 5. For example, let $S =$`SL-Availability` (see Figure 2). The algorithm initially checks if the SD contains an instance of S (in this case `SL-GoldAvailability`). Let now $P_o =$ `hasSLEAvailability`. The algorithm checks if `SL-GoldAvailability` is associated, via P_o, with an instance of `SLE-Availability` (an association which exists with `SLE-GoldAvailability`). Analogous checks are performed for the rest of the instances in the SD framework.

Table 5. Sample code for checking instance associations

```
1.Subproperty prop = null;
2.while (propertyIter.hasNext()) {
3.prop = propertyIter.next();
4.int countNli = countQuery("{ <"+instance_uri+">
      <"+prop.getUri()+"> ?var}");
5.String nli_uri = null;
6.if (countNli == 0) {
7.System.err.println("Error no connections");
8.throw new CompletenessException();}
9.if (countNli > 1) {
10.     System.err.println("Error > 1 connections");
11.     throw new CompletenessException();}
12.   else if (countNli == 1) {
13.     System.out.println("OK Instance");}
14.   RDFNode node = oneVarOneSolutionQuery("
      {<"+instance_uri+"> <"+prop.getUri()+"> ?var}");
15.   nli_uri = node.toString();}
```

4.2 The `complianceCheck` Method

The compliance checking algorithm proceeds by determining whether the values, or value ranges, specified in the SD are in accordance with the allowable values, or value ranges, specified in the BP. More specifically, the algorithm starts off by determining the number of values that are associated with a `QuantitativeValueFloat` instance (e.g. the instance `Var-GoldAvailability` of Figure 2) via each of the

properties `hasMinValueFloat` and `hasMaxValueFloat`. If this number is not equal to 1, an exception is raised (see lines 1-3 in the 1st row of Table 6). Otherwise, the algorithm proceeds to check that the range associated with the `QuantitativeValueFloat` instance is indeed subsumed by the corresponding range declared in the BP (see 2nd row of Table 6). For example, the algorithm checks whether the range [0.99999,1) associated with `Var-GoldAvailability`, is subsumed by the range [0.9999,1) associated with the corresponding `AvailabilityValue` instance in the BP (see Figure 2).

Table 6. Sample code for compliance checking

```
1.int minMaxFloatValueCount = countQuery("{<"+instance_uri +
    ">
                  gr:hasMinValueFloat ?someMinValue; " +
                 "gr:hasMaxValueFloat ?someMaxValue}");
2.if (minMaxFloatValueCount != 1) {
3.System.err.println("Erroneous range association");}
```

```
1.QuantitativeValueInstance bpQvInstance = null;
2.while (bpInstancesIter.hasNext()) {
3.bpQvInstance = bpInstancesIter.next();
4.if (((Float) vObj.getMaxValue() <
            (Float) bpQvInstance.getMaxValue()) &&
           ((Float) vObj.getMinValue() >=
            (Float) bpQvInstance.getMinValue())) {
5.System.out.println("OK - Instance ");
6.         break;}}
```

5 Related Work

To the best of our knowledge, no works other than [4] address the quality assurance dimension of CSB in the context of VEs. [4] recognises the need for frameworks that guide the creation, execution, and management of services in cloud-based VEs; it does not, however, address the quality assurance aspect of such frameworks. The rest of this section outlines work related to service description languages and to ontology-driven policy-based cloud service governance and quality control.

5.1 Service Description Languages

We provide an overview of different strands of service description formalisms. More specifically we outline approaches that: focus on syntactic service descriptions; consider the underlying semantics of web services; capture business aspects of services.

Syntactic service descriptions aim, primarily, at facilitating the interoperable data exchange between service registries (notably UDDI), service providers, and service consumers. The most prominent example is, perhaps, WSDL [13]. Nevertheless, syntactic service descriptions can only aid manual discovery, selection, and composition of services. In an attempt to automate these processes, a new breed of service description languages was introduced that enable Semantic Web Services [14]. These use ontologies in order to capture the functionality of web services in terms of an underlying, domain-specific, vocabulary. The rationale is that since both service descriptions and consumer demands rely on a common semantics, automatic service discovery and composition is, in principle, feasible. Prominent examples of standardisation efforts in this area include WSMO [15], OWL-S [16], SAWSDL [17], and SA-REST [18].

Whilst focusing on aspects which are important for the automatic composition and invocation of web services, the aforementioned approaches neglect any pertinent business details or, at best, address them as non-functional properties. This renders service descriptions cumbersome for service consumers and third-party intermediaries who are often interested in both business details and technical specifications in order to create added value by deploying, aggregating, customising, and integrating services. A third strand of description languages has therefore emerged, one which focuses on the business aspect of services. A prominent example is the Unified Service Description Language (USDL) [19]. USDL aims at unifying the business, operational, and technical aspects of a service in one coherent description. Although it provides a comprehensive framework, USDL has received limited adoption due mainly to its complexity and limited support for extensibility.

To overcome these limitations, Linked USDL [11] has been proposed. Linked USDL is a remodelled version of USDL which offers the following advantages [20,21]. Firstly, it uses a light-weight RDF(S) vocabulary which provides a framework of concepts and properties for modelling, comparing, and trading services and service bundles, as well as for specifying, tracking, and reasoning about the involvement of entities in service delivery chains. This framework can be easily extended through linking to other RDF(S) ontologies. Secondly, it supports large-scale, efficient, multi-party interactions by: i) capitalising on widely-adopted, general-purpose vocabularies such as Dublin Core, GoodRelations, SKOS, and FOAF; ii) embracing Linked Data as the core means for capturing facts about people, organisations, resources, and services. It is to be noted here that these advantanges facilitate the evaluation of the compliance of cloud services with pre-specified policies concerning the business aspects of service delivery and deployment. They are thus particularly significant from the standpoint of strengthening the *resilience* of cloud services.

5.2 Ontology-driven Policy-based Cloud Service Governance and Quality Control

Cloud service governance refers to policy-based management of cloud services with emphasis on quality assurance [22]. Current practice [23,24] focuses on the use of registry and repository systems combined with purpose-built software to check the conformance of services with relevant policies [25]. A major weakness in these systems is failure to achieve a separation of concerns between defining policies and evaluating data against these policies [22,25]. This has a number of negative repercussions such as lack of portability and lack of explicit representation of policy interrelations. Several works have attempted to address these shortcomings [26,27,28,29]. These generally employ bespoke languages, and ontologies, for capturing policies; the policies are then enforced at run-time typically through the use of a reference monitor. Closer to our approach are the works in [27,28,29] which embrace Semantic Web representations for capturing the knowledge encoded in policies.

In [27], the authors present KAoS – a general-purpose policy management framework which exhibits a three-layered architecture comprising: i) a *human interface layer*, which provides a graphical interface for policy specification; ii) a *policy management layer*, which uses OWL to encode and manage policy-related knowledge; iii) a *policy monitoring and enforcement layer*, which automatically grounds OWL policies to a programmatic format suitable for policy-based monitoring and policy enforcement. In [28] the authors propose Rei – a policy specification language expressed in OWL-Lite. It allows the declarative representation of a wide range of policies which are purportedly understandable – hence enforceable – by a wide range of autonomous entities in open, dynamic environments. In [29], POLICYTAB is proposed for supporting trust negotiation in Semantic Web environments. POLICYTAB advocates an ontology-based approach for describing policies that drive a trust negotiation process aiming at providing controlled access to Web resources.

Whilst achieving a proper separation of concerns between policy specification and policy enforcement, the aforementioned semantically-enhanced approaches rely on bespoke, non-standards-based, ontologies for the representation of policies. Such ontologies generally lack the expressivity for addressing the business details that characterise web and cloud services. They are therefore inadequate, as they stand, for capturing the business policies on which this work reports. In this respect, in our work, we have opted for Linked USDL: a language which readily provides the necessary constructs for capturing the required business policies.

6 Conclusions and Future Work

We have presented SC^3, a mechanism which offers capabilities with respect to the *Quality Assurance* dimension of CSB. SC^3 strengthens the resilience of services in cloud-based VE by evaluating their compliance with pre-specified policies concerning the business aspects of their delivery. By representing policies declaratively, in terms of an ontology, and not as part of the code of SC^3, a separation of concerns between

policy definition and policy enforcement is achieved. This effectively enables SC^3 to operate in a manner generic and agnostic to the underlying cloud delivery platform.

This paper has focused on the policy evaluation capabilities of SC^3 during the On-boarding phase of a service's lifecycle (see Section 2). Nevertheless, SC^3 offers a number of additional relevant capabilities which have been omitted here for reasons of space. These are: (i) an SLA monitoring capability which enables SC^3 to *continuously* check whether a service complies with the BP during the service Operation phase (i.e. during its consumption); (ii) a *standalone policy evaluation* capability, which assesses the correctness of a BP. In addition, SC^3 can be used for strengthening the resilience not only of application services (as demonstrated in this paper), but also of infrastructure and platform services.

SC^3 has been successfully used, in the frame of EU's Broker@Cloud project [30], for evaluating the quality of CRM services that are on-boarded on an exising commercial cloud application platform – namely the CAS Open [31] platform. In the future we intend to further assess the effectiveness of SC^3 by incorporating it in a number of additional cloud platforms, in particular platforms that provide infrastructure and platform services.

Acknowledgements. This research is funded by the EU 7th Framework Programme under the Broker@Cloud project (www.broker-cloud.eu), grant agreement n°328392.

7 References

1. Vaquero, L.M., Rodero-Merino, L., Caceres, J., Lindner, M.: A break in the clouds: Towards a cloud definition. SIGCOMM Comput. Commun. Rev., 39 (1), 50--55 (2008)
2. Foster, I., Zhao, Y., Raicu, I. Lu, S.: Cloud Computing and Grid Computing 360-Degree Compared. IEEE Grid Computing Workshop 2008. IEEE, pp.1--10, (2008)
3. Veloudis, S., Paraskakis, I., Friesen, A., Verginadis, Y., Patiniotakis, I., Rossini, A.: Continuous Quality Assurance and Optimisation in Cloud-based Virtual Enterprises. In Camarinha-Matos, L.M., Afsarmanesh, H. (eds.) PRO-VE 2014. LNCS, vol 434, pp. 621--632, Springer, Heidelberg (2014)
4. Cretu, L.G.: Cloud-based Virtual Organization Engineering. Informatica Economică., 16 (1), 98--109 (2012)
5. Cloud Computing Reference Architecture. Technical report, NIST (2011)
6. Veloudis, S., Friesen, A., Paraskakis, I., Verginadis, Y., Patiniotakis, I.: Underpinning a Cloud Brokerage Service Framework for Quality Assurance and Optimization. 6th IEEE International Conference on Cloud Computing Technology and Science, pp. 660—663, IEEE Press, New York (2014)
7. WSO_2 Carbon – 100% Open Source Middleware Platform, http://wso2.com/products/carbon/
8. Broker@Cloud project Deliverable 30.3, http://www.broker-cloud.eu/documents
9. Broker@Cloud project Deliverable 40.1, http://www.broker-cloud.eu/documents
10. Apache Jena, https://jena.apache.org/
11. Linked USDL, http://www.linked-usdl.org/
12. Broker@Cloud project Deliverable 30.2, http://www.broker-cloud.eu/documents

13. W3C Recommendation. 2001. *Web Services Description Language* (*WSDL*) 1.1. http://www.w3.org/TR/wsdl
14. McIlraith, S. A., Son, T. C. and Zeng, H. 2001. Semantic Web Services. *IEEE Intel. Syst.* 16, 2 (2001) 46-53. DOI=10.1109/5254.920599
15. W3C Member Submission. 2005. *Web Service Modelling Ontology* (*WSMO*). http://www.w3.org/Submission/WSMO
16. W3C Member Submission. 2004. *OWL-S: Semantic Markup for Web Languages*. http://www.w3.org/Submission/OWL-S
17. W3C Recommendation. 2007. *Semantic Annotation s for WSDL and XML Schema*. http://www.w3.org/TR/sawsdl
18. W3C Member Submission. 2010. *SA-REST: Semantic Annotations for Web Resources*. http://www.w3.org/Submission/SA-REST
19. Oberle, D., Barros, A., Kylau, U., and Heinzl, S. 2013. A unified description language for human to automated services. *Inf. Syst.* 38, 1 (March 2013), 155-181. DOI = 10.1016/j.is.2012.06.004
20. Pedrinaci, C., Cardoso, J., and Leidig, T. 2014. Linked USDL: a Vocabulary for Web-scale Service Trading. In *Proceedings of the 11th Extended Semantic Web Conference* (Crete, Greece, May 25-29, 2014). ESWC'14, Springer. ISBN: 978-3-319-12023-2
21. Cardoso, J., Pedrinaci, C., Leidig, T., Rupino, P., and De Leenheer, P. 2013. Foundations of Open Semantic Service Networks. *International Journal of Service Science, Management, Engineering, and Technology*, 4, 2 (April-June 2013), 1-16. DOI=10.4018/jssmet.2013040101
22. Kourtesis, D., Parakakis, I., Simons, A.J.H.: Policy-driven governance in cloud application platforms: an ontology-based approach. In: 4th. Int. Workshop on Ontology-Driven Information Systems Engineering (2012)
23. Marks, E.A.: Service-Oriented Architecture Governance for the Services Driven Enterprise. John Wiley & Sons (2008)
24. Zhang, L.J., Zhou, Q.: CCOA: cloud computing open architecture. In: IEEE International Conference on Web Services, pp. 607--616. IEEE Press, New York (2009)
25. Kourtesis, D., Paraskakis, I.: A registry and repository system supporting cloud application platform governance. In: 9th Int. Conf. on Service Oriented Computing. LNCS, vol. 7221, pp. 255--256, Springer, Berlin/Heidelberg (2011)
26. Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The Ponder Policy Specification Language. In Sloman, M., Lobo, J., Lupu, E. (eds.) *Proceedings of the International Workshop on Policies for Distributed Systems and Networks* (POLICY '01), pp. 18--38, Springer-Verlag, London (2000)
27. Uszok, A., Bradshaw, J., Jeffers, R., Johnson, M., Tate, A., Dalton, J., and Aitken, S.: KAoS Policy Management for Semantic Web Services. IEEE Intel. Sys. 19, 4, 32--41 (2004)
28. Kagal, L., Finin, T., Joshi, A.: A Policy Language for a Pervasive Computing Environment. In 4th IEEE Int. Workshop on Policies for Distributed Systems and Networks (POLICY '03), pp. 63--74, IEEE Computer Society, Washington, DC (2003)
29. Nejdl, W., Olmedilla, D., Winslett, M, Zhang. C.C.: Ontology-Based policy specification and management. In Gómez-Pérez, A. and Euzenat, J. (eds.) ESWC'05, pp. 290-302, Springer-Verlag, Berlin, Heidelberg (2005)
30. Broker@Cloud project, http://www.broker-cloud.eu/
31. CAS CRM, http://www.cas-crm.com/