A Generic Mechanism for Cloud Service Governance and Quality Control

Simeon Veloudis
South East European Research Centre (SEERC)
The University of Sheffield International Faculty, CITY College
Thessaloniki, Greece
(+30)2310253477
sveloudis@seerc.org

Iraklis Paraskakis
South East European Research Centre (SEERC)
International Faculty of the University of Sheffield, CITY College
Thessaloniki, Greece
(+30)2310253477
iparaskakis@seerc.org

Christos Petsos
South East European Research Centre (SEERC)
International Faculty of the University of Sheffield, CITY College
Thessaloniki, Greece
(+30)2310253477
chpetsos@seerc.org

ABSTRACT

With the pervasion of cloud computing the enterprise IT environment is progressively transformed into an ecosystem of highly distributed, task-oriented, modular, and collaborative cloud services. In order to deal effectively with the complexity inherent in such ecosystems, future enterprises are anticipated to increasingly rely on cloud service brokerage (CSB). This paper presents a CSB mechanism which offers capabilities with respect to the Quality Assurance dimension of CSB. The proposed mechanism evaluates the compliance of cloud services with pre-specified policies concerning the technical, and mainly the business aspects, of service deployment and delivery. By relying on a declarative representation of both services and policies, the proposed mechanism is kept generic and orthogonal to any underlying cloud delivery platform.

Categories and Subject Descriptors

• **Software and its engineering ~ Cloud computing**
• **Information systems ~ Semantic web description languages**.

Keywords

Cloud computing; cloud service brokerage; service governance; policy-based governance; quality control; service description languages; Linked USDL

1. INTRODUCTION

Cloud computing introduces an economy-based paradigm whereby infrastructure, platform, and application resources are abstracted as services [1]. Its increasing adoption transforms the enterprise IT environment into an ecosystem of wide-ranging, diverse, and interwoven services delivered remotely by a multitude of providers. Nevertheless, despite the significant advantages in terms of cost, flexibility and business agility [2,20], as the number of services proliferates, it becomes increasingly complex to reason with respect to service provision and consumption. For instance, it becomes increasingly complex to reason about certain aspects of service governance such as compliance to policies and regulations, and conformance to service level agreements (SLAs). The situation is further perplexed by the evolution of services, either through intentional changes initiated by service providers, or through unintentional changes, such as variations in service performance and availability.

In order to deal effectively with this complexity, future enterprises are anticipated to increasingly rely on cloud service brokerage (CSB) [1]. In this respect, the work in [22] proposed a conceptual architecture of a general framework which offers capabilities with respect to two dimensions of CSB, namely *Quality Assurance Service Brokerage*, and *Service Customisation Brokerage*. These capabilities revolve around three themes: (i) *governance and quality control*; (ii) *failure prevention and recovery*, and (iii) *optimisation*. The 1st theme is primarily concerned with checking the compliance of services with pre-specified policies concerning their technical and business aspects of delivery. It is also concerned with testing services for conformance with their expected behaviour, and with continuously monitoring their operation for conformance with SLAs. The 2nd theme is concerned with the reactive and proactive detection of service failures, and the selection of suitable adaptation strategies to prevent, or recover, from such failures. The 3rd theme is concerned with continuously identifying opportunities to optimise service consumption with respect to such goals as cost, quality, and functionality.

Continuing the work in [22], this paper presents a high-level account of a CSB mechanism which offers capabilities with respect to the *governance and quality control* theme. This mechanism comprises three main components: (i) the *Service Completeness-Compliance Checker* (SC^3), which is responsible for evaluating the compliance of services with pre-specified policies concerning the technical, and mainly the business aspects, of service deployment and delivery; (ii) a *governance registry system* which is responsible for the lifecycle management of services and policies; (iii) a *messaging system* which is responsible for delivering services to the SC^3 and to the governance registry system.

By adopting a declarative approach to service description, one which is based on an RDF(S) ontology, the presented CSB mechanism models services independently of the code that it employs for checking their compliance with policies. It thus overcomes a shortcoming encountered in current governance mechanisms, namely the lack of separation of concerns between service definition and policy enforcement [13,21]. In this respect, the CSB mechanism is kept generic and orthogonal to any underlying cloud service delivery platform.

The rest of this paper is structured as follows. Section 2 presents a motivating scenario. Section 3 outlines a conceptual architecture for the CSB mechanism and presents our declarative approach to

PCI 2015, October 01 - 03, 2015, Athens, Greece
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3551-5/15/10...$15.00
DOI: http://dx.doi.org/10.1145/2801948.2801988

service description; it also provides an overview of the process employed by the SC^3 for evaluating services against policies. Section 4 presents brief accounts of the messaging and governance registry components of the CSB mechanism. Section 5 outlines related work and Section 6 presents conclusions and future work.

2. CASE STUDY SCENARIO

The CSB framework proposed in [22] offers capabilities spanning the main phases of a service's lifecycle, namely Service On-boarding, Operation, and Evolution. This paper focuses on a particular mechanism of this framework which offers capabilities with respect to the *Service On-boarding* phase[1]. Below we identify these capabilities through an imaginary, albeit realistic case study.

Let CMx be a cloud marketplace through which a multitude of services are made available. CMx offers a variety of apps developed, and possibly pre-deployed, by a network of ecosystem partners.

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 pre-deployed service to CMx, call it *StoreCloud*, which provides encrypted and versioned cloud storage. In order for the new service to become available on CMx, 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*; Table 1 summarises the service-level attributes, and their corresponding SLOs, considered for the purposes of this paper. These SLOs essentially form CMx's *business policy* (BP) with respect to deploying *StoreCloud*. The BP additionally incorporates a set of *service-level profiles* (SLPs). SLPs are groupings of SLOs whose purpose is to formulate different 'deployment packages' offered by CMx. For example, a 'gold' SLP may group together the 'gold' SLOs of each of the service-level attributes of Table 1. Of course, the number of SLPs offered by CMx, and the SLOs that these comprise, is an application-specific issue determined by CMx itself. For instance, CMx may choose to define a 'gold' SLP as an SLP that comprises either 'gold'-only SLOs, or two 'gold' SLOs and a 'silver' SLO; alternatively, it may choose to define the latter grouping as a 'silver' SLP[2].

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 deployed. This SD incorporates all those service-level attribute *values* that *StoreCloud* commits to sustain. We term these values, the *service levels* (SLs) that *StoreCloud* offers. The SD also incorporates the SLP according to which *StoreCloud* aspires to be deployed.

Our CSB mechanism provides an SD evaluation capability which essentially allows CMx to determine whether the SD is compliant with the BP. Such a capability entails two kinds of evaluation: SD *completeness* evaluation and SD *compliance* evaluation. The former kind of evaluation aims at determining whether the SD specifies SLs for *all* required service-level attributes. For example, an SD which does not specify an SL for the *availability* attribute (see Table 1) cannot be considered complete. The latter kind of evaluation aims at determining whether the specified SLs fall within the corresponding ranges, or exactly match the values, prescribed in the BP. For example, an SD which specifies a 192-bit value for the *encryption* attribute (see Table 1), whilst it aspires to be deployed according to the 'gold' SLP, cannot be considered compliant.

3. THE SC^3 COMPONENT

The SC^3 lies at the heart of our CSB mechanism. It offers three main functionalities: (i) BP parsing and standalone evaluation; (ii) SD evaluation; (iii) continuous evaluation of the behaviour of a service during its consumption. In this paper we focus on the SD evaluation functionality. The rest of this section is structured as follows. Section 3.1 presents a brief description of a conceptual architecture for the CSB mechanism with reference to the scenario of Section 2. Section 3.2 describes our approach to the declarative representation of SDs – an approach which forms the basis of the SD evaluation process implemented by the SC^3. Section 3.3 provides a high-level account of this evaluation process.

3.1 CSB Mechanism 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 WSO$_2$ Carbon platform [24] (see Figure 1), as well as for advocating the Pub/sub paradigm for transporting SDs, is deferred until Section 4.

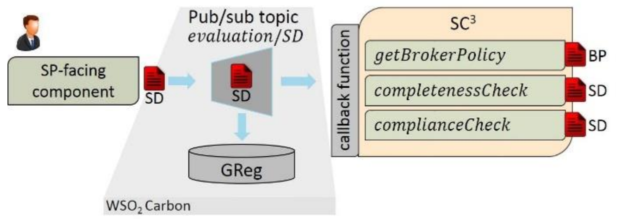

Figure 1: CSB conceptual architecture

The SC^3 exposes a callback function[3] for subscribing to the appropriate topic of the Pub/sub system and receiving the SDs;

[1] The mechanism also offers capabilities with respect to the Service Operation phase, by continuously evaluating the behaviour of a service during its consumption. These capabilities shall not, however, concern us in this paper.

[2] For simplicity, in this paper we assume that a 'gold' SLP comprises only the 'gold' SLOs of Table 1.

[3] The reasons for employing a callback function are explained in Section 4.1.

more specifically, upon the arrival of an SD, the SC^3 triggers three main methods: $getBrokerPolicy$, $completenessCheck$, and $complianceCheck$. The former method extracts all the required information from the BP for the subsequent SD completeness and compliance evaluations, whilst the latter two methods implement these evaluations. All three methods are implemented in Java using the Apache Jena (Core and ARQ) APIs [5]. High-level accounts of the latter two methods are provided in Section 3.3; an account of the former method is beyond the scope of this paper and can be found in [7].

3.2 Declarative Representation of SDs

We represent an SD in terms of a suitable framework of RDF instances. These instances populate the classes of an RDF(S) ontology, one which we employ in order to model a BP. This ontology is based on Linked USDL [14]: a lightweight RDF vocabulary for the description of policies and services with an emphasis on pertinent business aspects. The reasons for opting for Linked USDL are briefly outlined in Section 5; a more complete discussion can be found in [21]. By incorporating an ontology for modelling BPs and SDs, SC^3 achieves a clear separation of concerns: SDs are represented independently of the code that SC^3 employs for evaluating them. In this respect, SC^3 is kept generic and orthogonal to the underlying cloud service delivery platform.

A description of the RDF(S) ontology for modelling BPs is beyond the scope of this paper; the interested reader is referred to [6, 7, 21] for a relevant account. Below we outline our approach to modelling SDs with reference to the $StoreCloud$ example.

3.2.1 *Service-level Representation*

As mentioned in Section 2, the SD incorporates a set of service levels (SLs). Each SL specifies the value of a particular service-level attribute; it is represented in our model in terms of a suitable RDF instance. For example, the SL corresponding to the $availability$ attribute is represented by the instance $SL\text{-}Availability$ of Figure 2.

An SL specifies the value of its service-level attribute in terms of a suitable *service-level expression* (SLE), one which is represented in our model by an RDF instance, e.g. the $SLE\text{-}Availability$ instance of Figure 2. The SL is associated with its corresponding SLE through a suitable RDF property, e.g. the $hasSLEAvail$ property (see Figure 2).

An SLE incorporates a variable along with an appropriate value (or range of values) for this variable. For example, the SLE corresponding to the $availability$ attribute, incorporates the *availability* variable which is expressed by the RDF instance $Var\text{-}Availability$ (see Figure 2). The SLE is associated with this variable via the property $hasVarAvail$. This variable is in turn associated, via the property $hasValAvail$, with a range of values represented by the instance $AvailabilityValue$. This latter instance is associated with its delimiting values through the $hasMaxValueFloat$ and $hasMinValueFloat$ properties[4].

The framework of interconnected instances described above models the SL corresponding to the $availability$ service-level attribure; the SD encompasses analogous frameworks for the rest of the service-level attributes of Table 1 (see Figure 2).

3.2.2 *Service-level Profile Representation*

In addition to SLs, an SD also incorporates the *service-level profile* (SLP) according to which it *aspires* to be deployed. For example, the $StoreCloud$ SD includes the $SLP\text{-}Gold$ instance (see Figure 2) indicating that it aspires to be deployed according to the 'gold' SLP offered by CMx. Of course, it is up to the SC^3 to determine whether such an aspiration can be fulfilled by evaluating – on the basis of the corresponding BP – the SLs that the SD incorporates. These SLs are associated with the SLP instance through object properties such as $hasSLAvail$, $hasSLEncrypt$, and $hasSLStorage$ of Figure 2. The SD refers to the corresponding BP through the $BP\text{-}CPx$ instance (see Figure 2). An account of the ontology framework which is represented by this instance is, as already mentioned, beyond the scope of this paper.

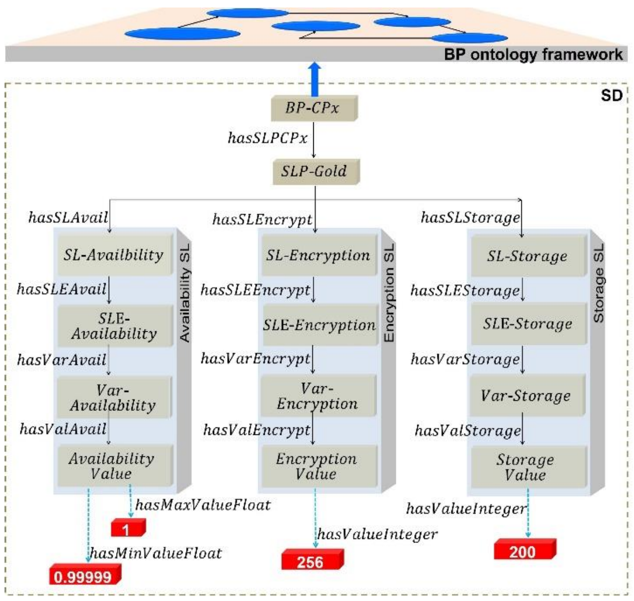

Figure 2: $StoreCloud$ SD representation

3.3 SD Completeness and Compliance Evaluation

In order to evaluate an SD, the SC^3 mechanism constructs a programmatic representation of the SLs outlined in Section 3.2. As mentioned in Section 2, such an evaluation entails a completeness and a compliance evaluation.

3.3.1 *Completeness Evaluation*

The $completenessCheck$ algorithm starts off by determining whether each SL encompasses all required instances. It then checks whether these instances are interconnected through the appropriate RDF object properties. For example, consider the SL for the $encryption$ attribute in Figure 2. The algorithm first checks whether the following instances exist: (i) an instance representing the SL ($SL\text{-}Encryption$); (ii) an instance representing the corresponding SLE ($SLE\text{-}Encryption$); (iii) an instance representing the variable which the SLE binds ($Var\text{-}Encryption$); (iv) an instance representing the value of this variable ($EncryptionValue$). If one or more of these instances are missing, the SL representation – hence the SD – cannot be considered complete. The algorithm then checks whether these instances are interconnected through RDF properties such as the ones depicted in Figure 2. If one or more of these properties are missing the SD cannot be considered complete. In addition, the SL representation cannot be considered complete if erroneous instance interconnections exist, for example an RDF property

[4] These are properties of the GoodRelations ontology (http:// www.heppnetz.de/ontologies/goodrelations/v1.html).

interconnecting the $SL\text{-}Encryption$ instance with the $Var\text{-}Encryption$ instance.

The algorithm also checks whether the SD encompasses an SLP instance and determines whether this instance is associated, through appropriate RDF object properties (such as $hasSLAvail$, $hasSLEncrypt$, and $hasSLStorage$ of Figure 2), with the provided SLs. Finally, the algorithm checks whether the SD encompasses an instance representing the corresponding BP (e.g. $BP\text{-}CPx$).

3.3.2 *Compliance Evaluation*

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 corresponding BP. More specifically, the algorithm starts off by determining the number of data values that are associated, through the properties $hasMaxValueFloat$, $hasMinValueFloat$, $hasValueInteger$, or $hasValueFloat$, with each instance representing the value of a variable (e.g. the $EncryptionValue$ instance of Figure 2). If this number is not equal to 1 (i.e. if a particular instance is associated with two or more data values, or with no data values), the SD cannot be considered compliant.

The algorithm then proceeds to check that the data values associated with such value instances either match with, or are subsumed by (in the case of ranges), the corresponding values in the BP and, in particular, the corresponding values in the SLP that the SD has opted for. For example, the algorithm checks whether the value associated with the $EncryptionValue$ instance is exactly 256, because this is the encryption value demanded by the 'gold' SLP that the SD has opted for. Similarly, it checks whether the range $[0.99999,1)$ associated with the $AvailabilityValue$, and the value 200 associated with the $StorageValue$, are subsumed by the corresponding ranges specified in the 'gold' SLP.

4. THE MESSAGING AND GOVERNANCE REGISTRY COMPONENTS

In addition to the SC^3, our CSB mechanism encompasses a messaging component and a governance registry component. This section describes the technologies that we have utilised in order to implement these components, and justifies our choices by outlining the benefits that these technologies bring to the CSB mechanism.

4.1 The Messaging Component

We advocate the Pub/sub paradigm for transporting SDs to the SC^3 and to the governance registry system (see Figure 1). Below we briefly describe this paradigm and outline the benefits that it brings about.

4.1.1 *The Topic-based Pub/sub Paradigm*

The Pub/sub paradigm typically employs a number of *topics* (for instance the $evaluation/SD$ topic of Figure 1) for transporting messages between sending and receiving entities: the former *publish* messages to topics which are then delivered to those receiving entities that have subscribed to these topics (i.e. to the so-called *subscribers*). Receiving entities employ *callback functions* (see Figure 1) for subscribing to topics and subsequently receiving messages.

The Pub/sub paradigm achieves a three-dimensional *decoupling* [10] between sending and receiving entities: (i) *space decoupling*, whereby sending and receiving entities are agnostic to each other; (ii) *time decoupling*, whereby sending and receiving entities do not (necessarily) participate simultaneously in message exchanges; (iii) *synchronisation decoupling*, whereby sending entities are not blocked when publishing messages, and receiving entities get asynchronously notified of the arrival of new messages (whilst, for example, performing some concurrent activity).

Such a multidimensional decoupling brings the following advantages to our CSB mechanism. Firstly, due to synchronisation and time decoupling, the performance of the SC^3 remains largely unaffected by communication overheads and bottlenecks. Secondly, space and synchronisation decoupling enhance the *scalability* of our mechanism by facilitating its potential for SD evaluation. For example, consider the scenario in which multiple SC^3 instances are deployed for balancing SD evaluation on the basis of, say, pre-defined functional service categories. Each SC^3 instance subscribes to appropriate topics, where each topic transports SDs corresponding to a particular functional category. Each SC^3 instance then automatically (and asynchronously) gets notified every time an SD is published to one of the topics to which it has subscribed. This facilitation is further reinforced by the ability of the Pub/sub mechanism to provide tree-like hierarchical decompositions of topics, reflecting the usual tree-like hierarchical decompositions of functional categories. Last but not least, the space dimension of decoupling allows the SC^3 to be updated, or even completely replaced, without affecting the SP-facing components (see Figure 1). This increases the portability and reusability of our CSB mechanism.

4.1.2 *The WSO$_2$ Message Broker*

We have implemented a Pub/sub system on top of the WSO$_2$ Message Broker (MB) [26], an open source server which provides messaging functionality for the WSO$_2$ Carbon platform. The WSO$_2$ MB brings a number of advantages to our CSB mechanism, besides the ones outlined above. Firstly, in addition to supporting JMS, a robust and mature specification which provides interoperability within the Java Platform, the WSO$_2$ MB also supports AMQP [4]. This significantly increases the generality of the CSB mechanism by enabling interoperability with many languages/platforms (e.g. Java, .Net, C, C++, PHP, Ruby, Erlang and more), and hence imposing virtually no restrictions on the implementation of the SP-facing components of Figure 1. Secondly, the WSO$_2$ MB supports *eventing* (and in particular WS-Eventing) which facilitates the SC^3 in communicating to SPs significant events that occur during service operation[5] in the form of messages. Thirdly, the WSO$_2$ MB increases the scalability and elasticity of our CSB mechanism in several ways: (i) by allowing MB nodes to be added/removed dynamically, at system run-time; (ii) by allowing message storage capacity to scale linearly using the Apache Cassandra database management system; (iii) by intelligently allocating the load between MB nodes using the Apache Zookeeper service. In addition, the WSO$_2$ MB exhibits the necessary reliability features by capitalising on the failure recovery functionality offered by Apache Zookeeper, and the fault-tolerant message persistence offered by Apache Cassandra. Last but not least, the WSO$_2$ MB allows JMS queueing of large messages – a feature which is desirable for our CSB mechanism in the face of large SDs.

4.2 The Governance Registry Component

Cloud service governance refers to policy-based management of cloud services with emphasis on quality assurance [11]. Current practice [15, 27] focuses on the use of registry and repository (RR) systems for storing and managing the lifecycle of services. These

[5] As mentioned at the beginning of Section 3, one of the functionalities offered by the SC^3 (and which is not considered in this paper) is continuous evaluation of the behaviour of a service during its consumption, i.e. during service operation.

are typically combined with purpose-built software to check the conformance of services with relevant policies. In our work, we have opted for the open-source WSO_2 Governance Registry (GReg) system [25]. The main reason that led us to this decision is that are: it provides ample support for SOA governance and service lifecycle management; it provides an extensible OSGi-based architecture [3] which allows the necessary customisation for facilitating the needs of our CSB mechanism.

More specifically, with respect to SOA governance, the WSO_2 GReg provides support for governing all aspects of services, including SDs, policies, and service consumption. In particular, it acts as a policy store for any policy enforcement point such as the SC^3.This naturally facilitates the retrieval of BPs for the evaluation of SDs. In addition, the WSO_2 GReg provides support for a wide range of services (REST, JSON, SOAP, Thrift, etc.) which increases the generality and extensibility of our CSB mechanism.

Moreover, the WSO_2 GReg provides comprehensive lifecycle management including dependency tracking and impact analysis. This increases the potential of the CSB mechanism for scalability: as the number of services proliferates, the impact of the evolution of a service on other dependent services is automatically tracked and dealt with through the provision of appropriate event handlers. Such evolution may be caused either through intentional changes, initiated by SPs (e.g. in the form of service updates), or through unintentional changes during service consumption, such as variations in service performance. In addition, the WSO_2 GReg provides content introspection and validation features which facilitate the standalone BP evaluation functionality offered[6] by the SC^3. Last but not least, the WSO_2 GReg is seamlessly integrated with the WSO_2 MB.

5. RELATED WORK

This section provides an overview of: (i) governance registries (other than the WSO_2 GReg) that have been considered for our mechanism; (ii) works that address the lack of separation of concerns in current governance registry systems; (iii) communication paradigms (other than the Pub/sub) that have been considered for our mechanism.

In addition to the WSO_2 GReg, two popular open-source RR systems have been considered for their suitability for our work: Membrane Registry [16] and Mule Galaxy [17]. The former is more oriented to monitoring consumed services and lacks support for service lifecycle management; it is therefore less appropriate for our CSB mechanism than the WSO_2 GReg. The latter provides support for service lifecycle management but lacks the handlers for triggering the SC^3 when significant events occur during service consumption. In addition, it lacks the extensibility and customisability offered by WSO_2 GReg through its OSGi-based architecture. It too is therefore less appropriate for our CSB mechanism than the WSO_2 GReg.

A general weakness in current governance registry systems, is their inability to achieve a proper separation of concerns between service and policy definition, and policy enforcement. 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 this shortcoming [8,23,11,18]. 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 [23,11,18] which embrace Semantic Web representations for capturing the knowledge encoded in policies. KAoS [23] is a general-purpose policy management framework which exhibits a three-layered architecture including a policy management layer, which uses OWL for encoding and managing policy-related knowledge, and a policy monitoring and enforcement layer. Rei [11] is a policy specification language expressed in OWL-Lite. It allows the declarative representation of a wide range of policies which are purportedly understandable by a wide range of autonomous entities in open, dynamic environments. In [18], POLICYTAB is proposed for supporting trust negotiation in Semantic Web environments. It 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 cloud services. They are therefore inadequate, as they stand, for capturing the SDs on which this work reports. In this respect, we have opted for Linked USDL: a language which readily provides the necessary constructs for capturing the required business policies.

Turning now to SD transportation, in addition to the Pub/sub paradigm, we have considered the following paradigms: *message passing*, *remote procedure calls*, and the *observer design* pattern. These paradigms offer synchronisation decoupling, but no decoupling in the time or space dimensions [10]; they therefore fail to bring about the advantages discussed in Section 4.1.1. Similarly, we have considered the *distributed shared memory* and *point-to-point message queuing* paradigms which provide space and time decoupling, but not synchronisation decoupling [10]. In addition, we have considered the *content-based* Pub/sub [19] and *type-based* Pub/sub [9] paradigms. In the former, subscribers receive events not by subscribing to predefined topics, but by determining the properties of the events that they wish to receive. In the latter, an analogous scheme is offered based on the type, rather than the content of events. Both of these paradigms provide complex abstractions which are generally not required in our CSB mechanism which is mainly interested in the exchange of a single type of message – namely SDs.

Finally, it is to be noted here that it is not within the scope of this paper to provide a comparison between different available open-source message brokers that could be used as a replacement of the WSO_2 MB. Such comparisons are often futile as most configurations, features, or protocols differ widely.

6. CONCLUSIONS AND FUTURE WORK

We have presented a high-level description of our CSB mechanism for cloud service governance and quality control. Our mechanism encompasses three main components: the SC^3, the WSO_2 GReg, and the WSO_2 MB. With respect to the former, we have outlined the declarative SD representation that it advocates, and the two kinds of SD evaluation that it offers, namely completeness and compliance evaluation. The SC^3 achieves a clear separation of concerns allowing SDs to be represented independently of the code employed for evaluating them. In this respect, it is kept generic and orthogonal to the underlying cloud service delivery platform. With respect to the latter two components, we have outlined the benefits that the chosen technologies bring to the CSB mechanism.

[6] As mentioned in Section 3, this is one of the functionalities offered by the SC^3 which is not considered in this paper.

Currently our CSB mechanism is being successfully used in the frame of EU's Broker@Cloud project (www.broker-cloud.eu) for evaluating the CRM services that are on-boarded on an exising commercial cloud application platform – namely the CAS Open platform (http://www.cas-crm.com/). 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 IaaS and PaaS platforms.

7. ACKNOWLEDGMENTS

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

8. REFERENCES

[1] 2011. *Cloud Computing Reference Architecture*. NIST.

[2] 2011. *Cloud: What an Enterprise Must Know*. Cisco.

[3] 2011. *Governance Registry brings integrity to SaaS platform*. Industrial Case Study Report. WSO$_2$.

[4] AMQP. 2015. https://www.amqp.org. Accessed: 2015-04-22.

[5] Apache Jena, https://jena.apache.org. Accessed: 2015-04-22.

[6] Broker@Cloud. 2014. *Deliverable 30.2 - Methods and tools for brokerage-enabling description of enterprise cloud services.* http://www.broker-cloud.eu.

[7] Broker@Cloud. 2014. *Deliverable 40.1 - Methods and mechanisms for cloud service governance and quality control.* http://www.broker-cloud.eu.

[8] Damianou, N., Dulay, N., Lupu, E., and Sloman, M. 2001. The Ponder Policy Specification Language. In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks* (Bristol, UK, January 29-31 2001). LNCS 1995. Springer-Verlag, London, UK,18-38

[9] Eugster, P., Guerraoui, R., and Damm, C. 2001 On objects and events. In *Proceedings of the 16th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* (Florida, USA, October 14-18, 2001). OOPSLA '01. ACM, New York, NY, USA, 254-269. DOI=10.1145/504282.504301

[10] Eugster, P.T., Felber, P.A., Guerraoui, R., and Kermarrec, A. 2003. The Many Faces of Publish/Subscribe. *ACM Comput Surv.* 35, 2 (March 2003), 114-131. DOI= 10.1145/857076.857078

[11] 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 (Lake Como, Italy, June 4-6 2003). POLICY '03. IEEE Computer Society, Washington, DC, USA, 63-74

[12] Kourtesis, D., Parakakis, I., and Simons, A.J.H. 2012. Policy-driven governance in cloud application platforms: an ontology-based approach. In *Proceedings of the 4th International Workshop on Ontology-Driven Information Systems Engineering* (Graz, Austria, July 24-27 2012). ODISE'12. IOS Press, Amsterdam, The Netherlands.

[13] Kourtesis, D. and Paraskakis, I. 2011. A registry and repository system supporting cloud application platform governance. In *Proceedings of the 4th International Conference on Service Oriented Computing* (Paphos, Cyprus, December 5-8, 2011). LNCS vol. 7221. Springer, Berlin/Heidelberg, 255-256. DOI= 10.1007/978-3-642-31875-7_36

[14] Linked USDL. http://linked-usdl.org/. Accessed: 2015-04-22.

[15] Marks, E.A. 2008. *Service-Oriented Architecture Governance for the Services Driven Enterprise*. Wiley.

[16] Membrane Open Source SOA Registry and Web Services Monitoring. http://www.membrane-soa.org/soa-registry. Accessed: 2015-04-22.

[17] MuleSoft Service Registry and Repository. http://www.mulesoft.com/resources/esb/service-registry-repository. Accessed: 2015-04-22.

[18] Nejdl, W., Olmedilla, D., Winslett, M., and Zhang. C.C. 2005. Ontology-Based policy specification and management. In *Proceedings 2nd European Semantic Web Conference* (Heraklion, Greece, May 29th – June 1st, 2005). ESWC'05. LNCS 3532. Springer-Verlag, Berlin, Heidelberg, 290-302

[19] Rosenblum, D. and Wolf, A. 1997. A design framework for Internet-scale event observation and notification. In *Proceedings of the 6th European Software Engineering Conference/ACM SIGSOFT 5th Symposium on the Foundations of Software Engineering*. ACM Press, New York, NY, 344–360.O

[20] Vaquero, L.M., Rodero-Merino, L., Caceres, J., Lindner, M. 2008. A break in the clouds: Towards a cloud definition. *SIGCOMM Comput. Commun. Rev.* 39, 1 (December 2008) 50-55. DOI = 10.1145/1496091.1496100.

[21] Veloudis S., Paraskakis, I., Friesen, A., Verginadis, Y., Patiniotakis, I. 2014. Underpinning a Cloud Brokerage Service Framework for Quality Assurance and Optimisation, In *Proceedings of the 6th IEEE International Conference on Cloud Computing Technology and Science* (Singapore, December 15 – 18, 2014), CloudCom'2014, IEEE, New York, NY, 660-663. DOI = 10.1109/CloudCom.2014.146.

[22] Veloudis, S., Paraskakis, I., Friesen, A., Verginadis, Y., Patiniotakis, I., and Rossini, A. 2014. Continuous Quality Assurance and Optimisation in Cloud-based Virtual Enterprises. In *Proceedings of the 15th International Working Conference on Virtual Enterprises* (Amsterdam, The Netherlands, October 5-7 2014). PRO-VE'14. LNCS 434, Springer, Heidelberg, 621-632, DOI= 10.1007/978-3-662-44745-1_61

[23] Uszok, A., Bradshaw, J., Jeffers, R., Johnson, M., Tate, A., Dalton, J., and Aitken, S. 2004. KAoS Policy Management for Semantic Web Services. IEEE Intell. Syst. 19, 4, 32-41

[24] WSO$_2$ Carbon – 100% Open Source Middleware Platform. http://wso2.com/products/carbon. Accessed: 2015-04-22.

[25] WSO$_2$ Governance Registry, http://wso2.com/products/ governance –registry. Accessed: 2015-04-22.

[26] WSO$_2$ Message Broker, http://wso2.com/products/message-broker. Accessed: 2015-04-22.

[27] Zhang, L.J. and Zhou, Q. 2009. CCOA: cloud computing open architecture. In *Proceedings of the IEEE International Conference on Web Services* (Los Angeles, CA, USA, July 6-10 2009). ICWS 2009, IEEE Press, New York, 607-616