Validating Policies for Dynamic and Heterogeneous Cloud Environments

Simeon Veloudis, Iraklis Paraskakis, Christos Petsos

Validating Policies for Dynamic and Heterogeneous Cloud Environments

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. With the pervasion of cloud computing, virtual enterprises (VEs) are anticipated to increasingly rely on ecosystems of highly distributed, task-oriented, and collaborative cloud services for their operations. In order to manage the complexity inherent in such ecosystems, VEs are expected to increasingly depend upon policies that regulate the deployment and delivery of these services. Nevertheless, the heterogeneity inherent in cloud services hinders the formulation of effective and interoperable such policies. This calls for a policy validation mechanism that is able to automatically evaluate the correctness of these policies. This paper proposes such a validation mechanism, one which is underpinned by a generic representation of the *knowledge* that lurks behind policies and thus is orthogonal to any particular cloud service delivery platform.

Keywords: Virtual enterprises; cloud computing; policies; ontologies; Linked USDL

1 Introduction

Cloud computing introduces a paradigm whereby infrastructure, platform, and application resources are abstracted as services and delivered remotely over the Internet by a multitude of providers [1], [2]. These services allow enterprises to realise significant savings while accelerating, at the same time, the development and deployment of new applications. Naturally, cloud computing is expected to impact the manner in which enterprises cooperate in a distributed collaborative network [3]. In particular, activities accomplished in the frame of a virtual enterprise (VE) may include utilisation of cloud services spanning different administrative domains and levels of capability (IaaS, PaaS and SaaS). For example, consider the scenario (adapted from [4]) whereby a scientific consortium is formed in order to collaboratively run a large cascade of meteorological and hydrological computational models for flood forecasting. Such processing utilises SaaS components offered by different consortium participants. Each such component may be deployed on a participant's proprietary infrastructure, or on infrastructure provisioned as a cloud service (IaaS offering). The computation also requires new specialised software that is developed on a software platform that is too provisioned as a service (PaaS offering).

Cloud-based VEs are therefore rapidly being transformed into complex ecosystems of heterogeneous, externally-sourced services. As the number of these services increases, keeping track of when and how they evolve over time, e.g. as a result of intentional or unintentional changes, becomes an increasingly challenging process. The situation is further perplexed by the dynamic nature of VEs as new enterprises may be added or existing ones leave.

In order to deal with this complexity, VEs are anticipated to rely on *policies* that regulate the deployment and delivery of cloud services. This calls for mechanisms that are able to generically evaluate the conformance of services with these policies, as well as to validate the correctness of the policies. We argue that, in order to deal with the dynamic nature of cloud-based VEs, as well as with the heterogeneity of the cloud services that they employ, these mechanisms must advocate a clear separation of concerns whereby policies are expressed orthogonally to the code used for enforcing them. This brings about a number of positive effects: (i) it keeps the mechanisms independent of any particular cloud delivery platform; (ii) it enables them to deal with a wide range of heterogeneous policies and services; (iii) it allows reasoning about policy interrelations (e.g. conflicting or overlapping policies).

In this respect, in [5] we proposed SC^3: a mechanism for automatically evaluating the compliance of cloud services with preset policies concerning their business aspects of delivery. In this paper, we go a step further and propose a *policy validation* mechanism for evaluating the *correctness* of the policies themselves. The mechanism is underpinned by an ontology able to accurately capture the seminal characteristics of the policies. By shifting the locus of effort from software development to the creation of knowledge structures, our ontology introduces an extra layer of abstraction which enables the formal representation of the concepts that lurk behind business policies. It thereby disentangles the definition of policies from the code employed for enforcing them. The proposed ontology draws upon Linked USDL [6] – an ontological framework which readily provides the necessary structures for accommodating the required concepts. Finally, it is to be noted that the proposed mechanism forms part of a broader mechanism that is being developed as part of the PaaSword project [7] and which aims at validating a wide range of policies, including security policies, in heterogeneous and dynamic cloud environments.

The rest of this paper is structured as follows. Section 2 presents related work. Section 3 outlines a motivating usage scenario and presents a particular categorisation of policies, namely state-oriented policies. Section 4 outlines an ontological model for state-oriented policies and presents the policy validation mechanism. Section 5 presents conclusions and future work.

2 Related Work

Several works have attempted to address the shortcomings entailed by the lack of separation of concerns between policy specification on the one hand, and policy enforcement on the other [8,9,10,11,12,13,14]. These works generally utilise custom languages, and ontologies, for capturing policies which are subsequently enforced at

run-time through a reference monitor. In [8], the authors propose PONDER – a purpose-made domain specific language for modelling security and management policies; [9,10,11] advocate the use of markup languages for the specification of access control policies. However, such syntactic descriptions fail to capture the knowledge that lurks behind policies: they are simply data models that lack semantic interoperability.

Closer to our work are the approaches reported in [12,13,14]. These are based on Semantic Web representations for representing the knowledge lurking behind *action-oriented* policies, i.e. policies that regulate when an actor can perform a particular action on a particular resource (see Section 3). In [12], the authors present KAoS – a policy enforcement and governance framework which advocates a 3-layered architecture consisting of: i) a human interface layer; ii) a policy management layer, which captures policies in OWL; iii) a policy monitoring and enforcement layer, which converts policies expressed in OWL to a programmatic format suitable for policy enforcement. The work reported in [13] proposes Rei – a rule-based policy framework that permits the declarative specification of heterogeneous policies. Rei policies specify those actions that *can* be performed, and those actions that *should* be performed, by a named entity. It therefore allows the specification of a desirable set of behaviours which are understandable – and enforceable – by autonomous entities in dynamic cloud environments. The work in [14], proposes POLICYTAB for supporting trust negotiation in dynamic Web environments. POLICYTAB embraces an ontology-based approach for capturing policies that guide a trust negotiation process for providing regulated access to Web resources.

Although they do achieve, to different extents, a separation of concerns between policy representation and policy enforcement, the approaches above rely on custom ontologies for modelling policies. Despite the fact that such ontologies may be suitable for capturing action-oriented policies, they generally lack the expressivity for addressing the *state-oriented* business policies on which this work reports (see Section 3). In this respect, instead of extending – or adapting – the ontologies defined in [12,13,14], we opt for Linked USDL: an easily extensible and diffused vocabulary of concepts and their associations that readily provides the necessary constructs for capturing the seminal characteristics of state-oriented policies. In addition, the reliance of the approaches in [12,13,14] on OWL [15], raises concerns about performance when they are used for checking the compliance of increasing loads of cloud services. Our work alleviates these concerns through the use of the lightweight RDF-based [16] vocabulary offered by Linked USDL.

3 Policies For Cloud Service Quality Assurance

We are interested in determining, in a generic manner, a set of *business constraints* that any service aspiring to be used by a VE must satisfy. In our framework, these constraints take the form of *business policies*. Next we demonstrate such policies through the example of Section 1. Let *CPMeteo* be a cloud delivery platform that hosts various services for the flood forecasting computation. These services are

developed either by the scientific consortium that performs these computations, or by third-party providers. Suppose that an ecosystem partner, or third-party provider, offers a new service to CPMeteo, call it *Meteo@Cloud*, which encrypts, stores and provides access to intermediate results of the various phases of the flood forecasting computation. For the new service to become available on CPMeteo, certain criteria must be satisfied. These essentially represent a set of *service-level objectives* (SLOs) that are expressed in terms of constraints on relevant *service-level attributes*; Table 1 summarises the service-level attributes, and their corresponding SLOs, considered for the purposes of this example. These SLOs form CPMeteo's *business policy* (BP) with respect to deploying Meteo@Cloud. The BP additionally incorporates a set of *service-level profile*s (SLPs). These are essentially sets of SLOs that formulate different 'deployment schemes' offered by CPMeteo. For instance, a 'gold' SLP may group together the 'gold' SLOs of each of the service-level attributes of Table 1[1].

Table 1. Entry-level criteria

Service-level Attribute	Acceptable Values	SLO	Comments
`storage`	[100,1000)	Gold storage	Size in TB
	[10,100)	Silver storage	
	[1,10)	Bronze storage	
`encryption`	256	Gold encryption	Key-length in bits
	192	Silver encryption	
	128	Bronze encryption	

We assume that the service provider who offers Meteo@Cloud provides a service description (SD) that specifies the manner in which Meteo@Cloud is to be offered through CPMeteo. This SD must be compliant with CPMeteo's business policy. For instance, an SD which fails to specify a value for the `storage` attribute, or one which specifies a value lesser than 1 TB, cannot be considered compliant with the BP.

3.1 State-oriented Policies vs Action-oriented Policies

The works in [11,12,13] focus on *action-oriented* policies, i.e. policies that control the conditions under which an actor can perform an action on a particular resource. Clearly, an indispensible ingredient of action-oriented policies is the specification of the actor who initiates the action, as well as of the action itself. The ontologies proposed in [11,12,13] naturally provide constructs for the specification of these ingredients. In contrast, as it becomes evident from Section 3.1, the kind of policies that

[1] Of course, the number of SLPs offered by CPMeteo, and the SLOs that these comprise, is an application-specific issue determined by CPMeteo itself. For instance, CPMeteo may choose to define a 'gold' SLP as an SLP that comprises either 'gold'-only SLOs, or two 'gold' and a 'silver' SLO; alternatively, it may choose to define the latter grouping as a 'silver' SLP.

our framework focuses on does not place any emphasis on specifying the actor that potentially initiates a particular action, or on the nature of the action itself: the actor may be *any* service provider, whereas the action is *invariably* the on-boarding of a service on a cloud service delivery platform. Our *business* policies predominantly focus on the conditions that must be met for a service to be on-boarded on such a platform. These conditions entail a set of artefacts that an SD must encompass. In this respect, they essentially formulate a set of *guaranteed states* which are represented through the SLOs and the SLPs that the business policies comprise. We shall term such policies *state-oriented*. Clearly, such a shift in focus from action-oriented to state-oriented business policies calls for a novel approach to the ontological modelling of policies, one which provides the necessary constructs for representing the artefacts that an SD must encompass. In this paper we propose such an approach based on Linked USDL [6]: an ontology framework which provides the necessary concepts for capturing SLPs and SLOs. In the following section we provide an outline of our approach. A brief account of the reasons that lead us to opt for Linked USDL is first in order.

3.2 Linked USDL

Linked USDL is a re-modelled version of USDL [17]. It draws upon the experience gained with USDL, as well as with other research efforts in the realm of Semantic Web Services and business ontologies [18], [19]. It builds upon the principles of Linked Data in order to endorse its use in a 'web of data'. In this respect, it models specifications in an RDF vocabulary that better supports the generic representation of web and cloud services. The adoption of Linked USDL brings about the following advantages [19]. Firstly, its reliance on existing widely-used RDF vocabularies, such as GoodRelations [21] the Simple Knowledge Organization System (SKOS) ontology [22], and FOAF [23]. In this respect, it promotes knowledge sharing whilst it increases the interoperability, and thus the reusability and generality, of our security policies. Secondly, by offering a number of different profiles, Linked USDL provides a holistic and generic solution able to adequately capture a wide range of business details. In addition, Linked USDL is designed to be easily extensible through linking to further existing, or new, ontologies.

4 Representation and Validation of State-oriented Policies

Linked USDL comprises a Core schema which in our model encodes certain invariable characteristics of a BP. From this Core schema a number of extension schemata hinge addressing diverse business aspects of a BP, such as pricing, SLA, security, and IPR; for the purposes of this paper, we focus on the SLA schema. Sections 4.1 and 4.3 below describe, with reference to the scenario of Section 3.1, how state-oriented policies are modelled within the Core and SLA schemata respectively. Sections 4.2 and 4.4 describe how the *correctness* of state-oriented policies is evaluated by the policy validation mechanism.

4.1 Linked USDL Core

We use Linked USDL's Core schema in order to formally capture the following two facts about a BP: (i) the identity of the *business entity* which is responsible for defining the BP (i.e. CPMeteo, in the case of the scenario of Section 3.1); (ii) the *role* in the capacity of which this business entity acts when defining the BP. To this end, we utilise the USDL Core classes and properties depicted in Figure 1[2].

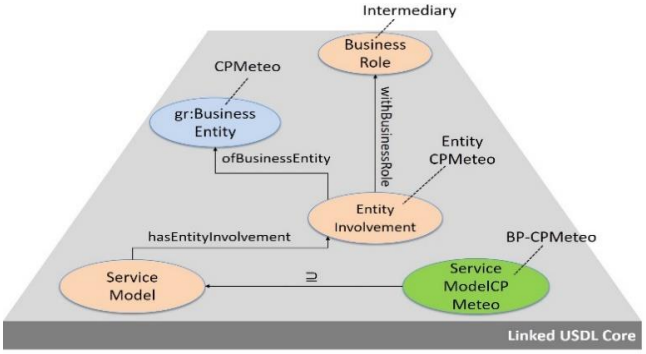

Fig. 1. Linked USDL Core classes, interrelations and instances

More specifically, a BP is identified by an instance of the USDL Core class `ServiceModel`. For example, in the scenario of Section 3.1, the BP is identified by the instance `BP-CPMeteo` depicted in Figure 1. In fact, `BP-CPMeteo` is an instance of a particular subclass of `ServiceModel`, namely of the subclass `ServiceModelCPMeteo`, which is specifically introduced into our model for accommodating all of CPMeteo's BPs[3]. Now, in order to capture the aforementioned two facts we:

1. Associate the instance `BP-CPMeteo` with an instance, say `EntityCPMeteo`, of the class `EntityInlvolvement` through the property `hasEntityInvolvement`.
2. Associate the instance `EntityCPMeteo` with the instance `CPMeteo` of the class `gr:BusinessEntity` via the property `ofBusinessEntity`. Note that the instance `CPMeteo` identifies the business entity which is responsible for defining the BP.

---

[2] This is by no means the complete set of the classes and properties offered by USDL Core, but rather an appropriate subset discerned for the purposes of this paper. Note that in order to reduce notational clutter we avoid specifying namespaces for classes and properties, unless a class or property comes from an external ontology (e.g. the GoodRelations ontology). In addition, the following conventions are used in the figures of this paper: a class is represented by an oval; a property is represented by an arrow decorated with the name of the property; a subclass relation is represented by an arrow decorated with the subset symbol (⊆); instance-class associations are represented with perforated lines.

[3] Although we concentrate here on a single BP, CPMeteo may be associated with a number of different BPs, each represented by a distinct instance of the class `ServiceModelCPMeteo`.

3. Associate the instance `EntityCPMeteo` with the instance `Intermediary` of the class `BusinessRole` via the property `withBusinessRole`. Note that the instance `Intermediary` identifies the role in the capacity of which CPMeteo acts when defining the BP.

By virtue of steps 1 and 2 above, and the transitivity of object properties, the instance `BP-CPMeteo` is associated (indirectly) with the instance `CPMeteo`, thus identifying the business entity responsible for defining `BP-CPMeteo`. Similarly, by virtue of steps 1 and 3 above, `BP-CPMeteo` is associated with the instance `Intermediary`, thus identifying the role in the capacity of which CPMeteo acts when defining `BP-CPMeteo`.

4.2 Validating the Core Portion of a BP

Our aim is to verify that the Core portion of a BP correctly captures the two facts mentioned in Section 4.1, namely the business entity responsible for defining the BP, as well as the role in the capacity of which this business entity acts when defining the BP. First, however, we need to ensure that there indeed exists an instance of the class `ServiceModel` which identifies the BP. To this end, the policy validation mechanism uses the Apache Jena Java API in order to obtain all those resources that are defined as subclasses of the class `ServiceModel`, as well as all the instances that are encompassed in these subclasses. The mechanism then ensures that there exists exactly one instance in the subclass `ServiceModelCPMeteo` (i.e. the `BP-CPMeteo` instance of Figure 1) which identifies the BP.

Turning next to verifying that the BP correctly captures the two facts mentioned in Section 4.1, the validation mechanism uses again the Jena Java API in order to check that: i) `BP-CPMeteo` is associated with exactly one instance of the class `EntityInvolvement` via the `hasEntityInvolvement` property; ii) this `EntityInvolvement` instance is associated with the instance `CPMeteo` of the class `gr:BusinessEntity`, and with no other instances from that class; iii) the same `EntityInvolvement` instance is associated with the instance `Intermediary` of the class `BusinessRole`, and with no other instances from that class. In case any of these checks fails, the validation process fails.

4.3 Linked USDL SLA

USDL SLA provides an adequate ontological framework for modelling the knowledge that lurks behind a BP, i.e. the particular characteristics that a service must exhibit in order to be on-boarded on a cloud platform. As discussed in Section 3, these characteristics are formulated in terms of SLPs and SLOs. Below we outline how these SLPs and SLOs are modelled in USDL SLA[4].

[4] The modelling of SLPs and SLOs in Linked USDL was presented in [5]; it is repeated here for completeness. Recall from Section 3 that SLOs express entry-level criteria that must be

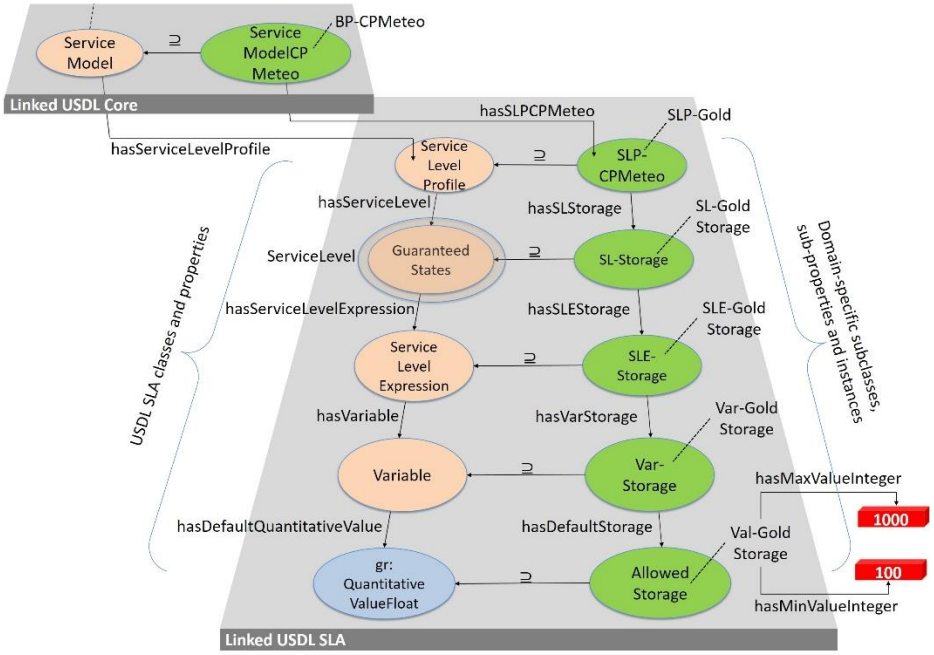

Fig. 2. Linked USDL SLA framework

SLPs are expressed as instances of the USDL SLA `ServiceLevelProfile` class (see Figure 2). For example, the SLPs of the scenario of Section 3.1 are expressed as instances of a particular subclass of the class `ServiceLevelProfile`, namely `SLP-CPMeteo`, which encompasses all SLPs offered by CPMeteo. Thus, the instance `SLP-Gold` depicted in Figure 2, represents CPMeteo's 'gold' profile. The class `SLP-CPMeteo` is associated with the BP (i.e. with the instance `BP-CPMeteo` that identifies the BP) through the property `hasSLPCPMeteo` (see Figure 2). This is a sub-property of the USDL SLA property `hasServiceLevelProfile` which associates the USDL Core class `ServiceModel` with the USDL SLA class `ServiceLevelProfile`.

As reported in Section 3.1, each SLP comprises an SLO for each of the service-level attributes of Table 1. In our framework, the SLOs corresponding to a particular attribute are expressed as instances of a suitable subclass of the USDL SLA `GuaranteedStates` class (see Figure 2). For example, the SLOs of the attribute `storage`, i.e. the 'gold storage', 'silver storage', and 'bronze storage' SLOs of Table 1, appear as instances of the subclass `SL-Storage` (for instance the instance `SL-GoldStorage` depicted in Figure 2)[5]. An SLP is related to the SLOs that it

satisfied in order for a service to be on-boarded on CPMeteo. Also recall that SLPs are groupings of SLOs that represent different 'deployment packages' offered by CPMeteo.

[5] Similarly, the SLOs of the `encryption` attribute, i.e. the 'gold encryption', 'silver encryption', and 'bronze encryption' SLOs of Table 1, appear as instances of an analogous subclass, say `SL-Encryption` (not shown in Figure 2).

comprises through suitable sub-properties of the property `hasServiceLevel`. For instance, an SLP is associated with the SLOs of the `storage` attribute through the sub-property `hasSLStorage`; in Figure 2, this sub-property interrelates the instances `SLP-Gold` and `SL-GoldStorage`.

Now, each SLO is expressed in terms of a *service-level expression* (SLE) which articulates those conditions that must be met for the SLO to be achieved. An SLE is represented as an instance of an appropriate subclass of the class `ServiceLevelExpression` (see Figure 2). For instance, the SLEs corresponding to the storage SLOs take the form of instances of the class `SLE-Storage` (cf. `SLE-GoldStorage` of Figure 2). SLOs are related to their SLEs through suitable sub-properties of the property `hasServiceLevelExpression`. For instance, the storage SLOs are related to their SLEs via the property `hasSLEStorage` depicted in Figure 2.

Finally, each SLE is associated with a *variable* that represents a certain attribute and is linked with an allowable range of values. Following a symmetrical approach to the one provided for SLOs and SLEs, variables take the form of instances of suitable subclasess of the class USDL SLA `Variable` class. Similarly, value ranges take the form of instances of suitable subclasses of the class `gr:QuantitativeValueInteger`. Figure 2 illustrates these subclasses (`Var-Storage`, `AllowedStorage`), and instances (`Var-GoldStorage`, `Val-GoldStorage`), for the 'gold' storage attribute. SLEs bind their constituent variables through sub-properties of the property `hasVariable` (e.g. the `hasVarStorage` sub-property of Figure 2). Similarly, variables are linked to their allowable value ranges through sub-properties of the property `hasDefaultQuantitativeValue` (e.g. the `hasDefaultStorage` sub-property of Figure 2).

4.4 Validating the SLA Portion of a BP

Two main methods are involved in the validation of the SLA portion of a BP: (i) a parsing method which constructs a programmatic representation of the framework described in Section 4.3; (ii) a policy validation method which performs the necessary correctness checks.

Programmatically Representing the SLA Portion of a BP. Each of the USDL-SLA classes depicted in Figure 2 is represented programmatically in terms of an appropriate in-memory data structure. Subsequently, for each such class S, the validation mechanism discovers all the domain-specific subclasses C of S that are encompassed in the BP. It then populates the data structure corresponding to S with these subclasses. For example, it populates the data structure corresponding to the USDL-SLA class `ServiceLevelProfile` with the `SLP-CPMeteo` subclass and the data structure corresponding to the USDL-SLA class `ServiceLevel` with the `SL-Storage` subclass. The same applies for the data structures corresponding to the rest of the USDL SLA classes depicted in Figure 2. Each discovered subclass C is also represented programmatically in terms of an appropriate in-memory structure. For

each such subclass C, the validation mechanism proceeds to discover all those properties that are defined in the BP, along with their linked ranges, which happen to have as the subclass C as their domain. It then populates the data structure corresponding to C with these properties. These properties are effectively all the *sub-properties* of the USDL SLA properties that are encountered in the BP. By the end of this process, we have a complete in-memory representation of the model depicted in Figure 2.

Validating the SLA Portion of a BP. The policy validation method operates on the in-memory data structures constructed by the parsing method above. More specifically, it checks that: (i) at least one instance exists in each of the subclasses of the `ServiceLevelProfile` class (e.g. the instance `SLP-Gold` depicted in Figure 2); (ii) the instance `BP-CPMeteo` (that identifies the BP) is associated with such an instance through a sub-property of the property `hasServiceLevelProfile`. Subsequently, the validation method checks that: (i) each SLP instance is associated with at least one SLO instance, i.e. with at least one instance from the subclasses of the `GuaranteedStates` class that were discovered by the parsing process; (ii) these associations take place through distinct sub-properties of the USDL SLA property `hasServiceLevel`. In an entirely symmetrical manner, the validation mechanism then checks that: (i) each SLO (i.e. each subclass of the class `GuaranteedStates`) is associated with at least one SLE (i.e. with at least one subclass of `ServiceLevelExpression`) via a distinct sub-property of `hasServiceLevelExpression`; (ii) each SLO instance is associated with at least one SLE instance. An analogous set of checks applies for the associations between SLEs and variables. Furthermore, the mechanism checks that there exists an one-to-one correspondence between variables and quantitative (or qualitative) values. In addition, it checks that: (i) all quantitative value instances are associated with minimum and maximum values via the properties `gr:hasMinValueInteger`, `gr:hasMaxValueInteger`; (ii) each quantitative value instance is associated with an appropriate unit of measurement via `gr:hasUnitOfMeasurement`.

Testing the Validation Mechanism

In order to increase our assurance on the policy validation mechanism, a *test harness* has been developed for potentially discovering invalid BPs that may go undetected by the mechanism. The harness automatically injects errors into the BP and observes whether these are detected by the validation mechanism. More specifically, for every triple T in the BP, it extracts the subject T_S, the predicate T_P, and the object T_O, of T. It then generates a new triple T' by introducing an error in T_S (e.g. a typo) and runs the validation mechanism with T being replaced by T'. It follows the same procedure for T_P and T_O. The test harness revealed a number of bugs that led to failure in detecting certain erroneous BPs. For example, a significant bug that was discovered was that the validation mechanism failed to check whether certain object properties interconnecting instances in the BP were in fact sub-properties of the correct Linked USDL properties.

5 Conclusions and Future Work

We have presented a *policy validation* mechanism which automatically assesses the *correctness* of *state-oriented policies*. This ensures that cloud services are checked for compliance against *correctly-formed* policies. We believe that such a mechanism is a vital ingredient of any framework aspiring to provide policy-based quality assurance of cloud services. Our policy validation mechanism has been used in conjunction with the SC^3 mechanism that we have developed for evaluating the quality of cloud services. Both mechanisms have been successfully used for assessing CRM services on-boarded on an exising commercial cloud application platform – namely the CAS Open platform [18].

As part of future work, we intend to extend our mechanism to cover a wider range of policies including action-oriented policies. Such an extension will enable us to model policies articulating the actions that need to be taken in case one or more SLOs are violated (or are about to be violated) *during* the consumption of a service. In addition, we are already extending our mechanism to cope with context-aware security policies in highly-dynamic and heterogeneous cloud environments.

Acknowledgements. The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 644814.

6 References

1. 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)
2. Cloud Computing Reference Architecture. Technical report, NIST (2011)
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. Yi, S.: Virtual Organization Based Distributed Environmental Spatial Decision Support Systems: Applications in Watershed Management. PhD Thesis, Michigan State University, 3348219 (2008)
5. Veloudis, S., Paraskakis, I., Petsos, C.: Cloud Service Brokerage: Strengthening Service Resilience in Cloud-Based Virtual Enterprises. In Camarinha-Matos et al. (eds.) PRO-VE 2015. LNCS, vol 463, pp. 122--135, Springer, Heidelberg (2015)
6. Linked USDL, http://www.linked-usdl.org/
7. PaaSword project, http://www.paasword.eu/
8. 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)
9. eXtensible Access Control Markup Language (XACML) Version 3.0. 22 January 2013. OASIS Standard. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

10. Security Assertions Markup Language (SAML) Version 2.0. Technical Overview 25 March 2008. OASIS Standard. https://www.oasis-open.org/committees/download.php/27819/sstc-saml-tech-overview-2.0-cd-02.pdf (2008)
11. WS-Trust 1.3. 19 March 2007. OASIS Standard. http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.doc (2007)
12. 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)
13. 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)
14. 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)
15. OWL Web Ontology Language Reference. W3C Recommendation. 10 February 2004. http://www.w3.org/TR/owl-ref/
16. RDF 1.1 XML Syntax. W3C Recommendation. 10 February 2004. http://www.w3.org/TR/rdf-syntax-grammar/
17. Barros, A. and Oberle, D.: Handbook of Service Description: USDL and its Methods, Springer (2012)
18. Cardoso, J., Pedrinaci, C., Leidig, T., Rupino P. and Leenheer, P.: Foundations of Open Semantic Service Networks. *International Journal of Service Science, Management, Engineering, and Technology,* vol. 4, no. 2, 1-16 (2013)
19. Cardoso, J., Pedrinaci, C., Leidig, T.: Linked USDL: a Vocabulary for Web-scale Service Trading. In *11th Extended Semantic Web Conference (ESWC)* (2014)
20. GoodRelations: The Professional Web Vocabulary for E-Commerce. http://www.heppnetz.de/projects/goodrelations/
21. SKOS Simple Knowledge Organization System. http://www.w3.org/2004/02/skos/
22. The FOAF Project. http://www.foaf-project.org
23. CAS CRM, http://www.cas-crm.com/