Policy-driven governance in cloud application platforms: an ontology-based approach

Dimitrios KOURTESIS[a,1] Iraklis PARASKAKIS[a], and Anthony J.H. SIMONS[b]
[a] *South-East European Research Centre, International Faculty, The University of Sheffield, Thessaloniki, Greece*
[b] *Department of Computer Science, Engineering Faculty, The University of Sheffield, Sheffield, UK*

Abstract. The emergence of cloud computing is changing the way in which software services are delivered and consumed. In a complex ecosystem of virtualised, interlinked applications and services, there is a greatly increased need for cloud platform operators to control the quality and standards of software offered on their platforms by enforcing different kinds of policy. Existing tools for policy-driven governance in service delivery platforms suffer from important limitations. Such tools do not allow policies to be expressed abstractly and to be maintained separately from the low-level code that is written to enforce them, nor allow the relationships among policies or between policies and their subjects to be captured explicitly. We introduce an alternative approach to governance which aims to address those limitations by allowing policies to be represented on the basis of ontologies, and then enforced by a generic and reusable mechanism that employs off-the-shelf logical reasoning engines. The approach extends to different kinds of governance policy and has the advantage of being formal, declarative and fully standards-based, such that cloud platform operators can automate several policy engineering tasks and easily implement changes in the way software lifecycles and artefacts are governed.

Keywords. Cloud application platform, PaaS, governance, policy, ontology, OWL2 DL, data validation

1. Introduction

Cloud computing offers the prospect of a future market, in which software is created and distributed upon various cloud application platforms by communities of third-party developers. This trend is already seen in the huge popularity of third-party "apps" developed for mobile computing platforms. Cloud platform operators will bear an increased responsibility to control the quality and standards of software offered through their platforms, not only in terms of its correctness and robustness, but also in relation to legal norms and business practices. This gives rise to the notion of *governance*, in which software artefacts and production processes are regulated by rules, expressed as *policies*, which may change over time, or vary according to the locale in which the software is deployed.

[1] Corresponding Author.

One of the greatest challenges in operating a cloud platform which allows third-party extensions is to ensure proper governance by enforcing policies on the software contributed by different parties. Addressing this challenge was one of the main goals in the recently completed research project CAST[2]. The project was set up to create a cloud application platform that enables the development and deployment of on-demand business applications by third parties [1]. The governance tooling in CAST was mainly concerned with two kinds of policy: lifecycle governance, i.e. controlling how third-party extensions evolve on the platform, and artefact governance, i.e. controlling the structure and contents of software artefacts associated with these extensions.

During the CAST project we defined 40 different policies on lifecycle and artefact governance, and developed custom software components which enforce them within an open source registry and repository system. Through this exercise we gained some valuable insights into the limitations of current approaches to policy management, as implemented by today's governance tooling. This paper reports on our ongoing work towards addressing those limitations through an ontology-based approach to policy-driven governance. We are working towards the development of a framework which leverages ontologies as both design-time artefacts for policy modelling (policy definition), as well as run-time artefacts for policy checking (policy enforcement).

In the rest of this paper we explain the motivation for policy-driven governance in cloud application platforms, briefly discuss the limitations of today's governance tooling with regards to the definition and enforcement of policies, provide an overview of our proposed approach, and discuss related work. To the best of our knowledge, this work represents the first attempt to investigate an ontology-based approach to the definition and enforcement of governance policies in open service delivery platforms.

2. The need for policy-driven governance in cloud application platforms

An important trend within the emerging domain of cloud computing is the adoption of *cloud application platforms* — a particular subclass of Platform-as-a-Service (PaaS) offerings supporting the development and delivery of software applications. Cloud application platforms offer a combination of managed computing infrastructure that is made accessible over the internet, with a set of tools and services allowing developers to create applications and have them deployed and executed over that infrastructure. Force.com, LongJump, Engine Yard, and Zoho are some of the currently established service providers in this space.

By design, a cloud application platform is an open environment that is expected to continuously expand through the addition of new applications and/or services by third-party developers. Given this inherent dynamism and tendency for expansion, one of the most challenging objectives for a platform provider is ensuring that the introduction or modification of third-party extensions will not have a negative impact on the platform's stability and reliability. Meeting this objective presents challenges to platform design at many different levels, but is also, fundamentally, a problem of platform *governance* [2]. In the context of this work, we take governance to mean the implementation of *policies* for controlling the *lifecycle* of third-party extensions that are added to the platform, as well as controlling the quality of their associated *artefacts*.

[2] http://www.cast-project.eu

2.1. Lifecycle governance

Lifecycle governance is concerned with ensuring a structured and disciplined approach to introducing third-party extensions, deploying them to the platform’s execution environment, modifying them, or removing them. Central to lifecycle governance is the notion of a lifecycle model defining the phases that every different managed software entity is obliged to proceed through, as well as the preconditions associated with the transition from one lifecycle phase to the next. For example, one of the lifecycle governance policies defined for the CAST platform states that a precondition for allowing an app to proceed from the review phase to the beta testing phase, is for the app to be associated with a quality review report that contains a positive evaluation. In addition to this precondition, the app must continue to satisfy all preconditions defined for previous transitions (i.e. the transition from local development to sandboxed testing and from sandboxed testing to review).

2.2. Artefact governance

Artefact governance is concerned with ensuring that all artefacts associated with managed entities are conformant to a set of technical, business or legal constraints defined by the platform provider. Central to artefact governance is the notion of artefact specifications which place constraints on the structure and contents that different kinds of configuration, specification or code artefacts are allowed to have. For example, one of the artefact governance policies defined for the CAST platform states that the interface specification (WSDL) of every external web service used by one or more apps, should contain exactly two non-identical endpoint URLs, which point to different servers on which the service is deployed (primary and backup endpoints). The rationale is to provide a failover alternative in case the primary server that hosts the service becomes unavailable.

3. Limitations of policy management in state of the art governance tooling

Many policies, like the above examples from the CAST project, are amenable to automated checking. Other policy checks may not be feasible to automate, or, in some cases, may explicitly be required to be carried out in a manual fashion. In this work we focus on policies whose enforcement can, and should be, fully automated.

Software tools supporting governance in open service delivery platforms have been commercially available for several years now. Typically, such tools come in the form of an integrated registry and repository system. Their most common usage scenario is to support the management of service-oriented enterprise infrastructures [3]. A feature that is central in such governance tools is providing users with some way of checking whether the data in the registry/repository system conforms to relevant policies. Each governance tool achieves this through a different approach to policy definition and enforcement.

In the scope of the CAST project, we analysed and compared two open source registry and repository systems in order to understand how they allow policies to be defined and enforced. Namely, we analysed Mule Galaxy [4] and WSO2 Governance Registry [5]. We found many similarities among the two systems, as well as similarities with other commercial governance tools from vendors like IBM [6] and Oracle [7].

Their main limitations with regards to the definition and enforcement of policies, as observed though our study, can be summarised in the following.

Lack of separation of concerns between definition and enforcement of policy: Policy definition and policy checking are entangled within the same software unit. Policy authors write custom code that interfaces with the registry/repository system through an API, and checks if some data of interest conforms to certain constraints. Those constraints are defined implicitly as part of the same code that checks for data conformance. Except for the case where such constraints are defined in an explicit way (e.g. in a separate XML schema document) there is no differentiation between *what* a policy is about, and *how* data can be checked for conformance to that policy. Typically, the only machine-readable representation of a policy is the code that enforces it.

Lack of abstraction in policy representation: Because of the above, policy logic is represented at the same level of abstraction as the implementation of the registry and repository system. The rules or constraints that a policy comprises are encoded in an imperative style, as part of the same low-level logic that queries databases and parses files to check instance data for violations. The encoding of a policy is therefore disconnected from the high-level domain concepts that one would use to communicate its purpose and the policy author's intent.

Lack of formal representation of policy constraints and relationships. The relationships among policies, as well as between policies and their subjects (i.e. the logical entities in the governance domain) are not captured explicitly. Tracing the association of an operational-level policy to other policies at the same level or a higher (strategic) level is not possible. The same holds for tracing the relationships between a particular platform resource and all policies directly or indirectly related to it. Last but not least, the absence of any formal encoding of policies makes it difficult to analyse them, to reason how policies may affect other policies and to perform automated verification and validation.

The above limitations have negative implications with respect to policy maintainability, comprehensibility, verifiability, traceability, interoperability, and with respect to the overall agility of platform governance.

4. An ontology-based approach to policy-driven governance

The ongoing research reported in this paper aims to investigate how to overcome the above limitations by a new approach to the definition and enforcement of governance policies, where ontology-based knowledge representation and reasoning will be central.

The benefits of applying ontologies and related Semantic Web technologies to policy engineering have already been explored in earlier work [8, 9, 10]. Tonti et al [9] have reported reduced human error, simplified policy analysis, reduced policy conflicts, and increased interoperability. Uszok et al [11] have emphasised the benefits of reusability, extensibility, verifiability, safety, and "reasonability".

Past research on this topic has focused mostly on policies relating to security, privacy, trust management, quality of service, or business norms [12]. In our work, we are concerned primarily with policies for lifecycle governance (regulating the evolution of software on a platform) and artefact governance (regulating the configuration of deployed software and associated artefacts). Notwithstanding these differences, we believe that an ontology-based approach to policy-driven governance can result in many analogous improvements.

4.1. Overview of the proposed approach

In abstract terms, the approach we put forward comprises three major components. Firstly, a platform governance ontology to provide the basic vocabulary and modelling constructs for describing platform resources and governance policies. Secondly, a set of mechanisms to generate abstract, ontology-based descriptions of different kinds of platform resources by means of transformation from their native representation into Linked Data [13]. Thirdly, a methodology to encode all of the different kinds of platform policies in some appropriate logic-based form, based on the same ontology, and a generic and reusable infrastructure to check if the abstract descriptions of platform resources are conformant to those policies. Figure 1 illustrates the concept.

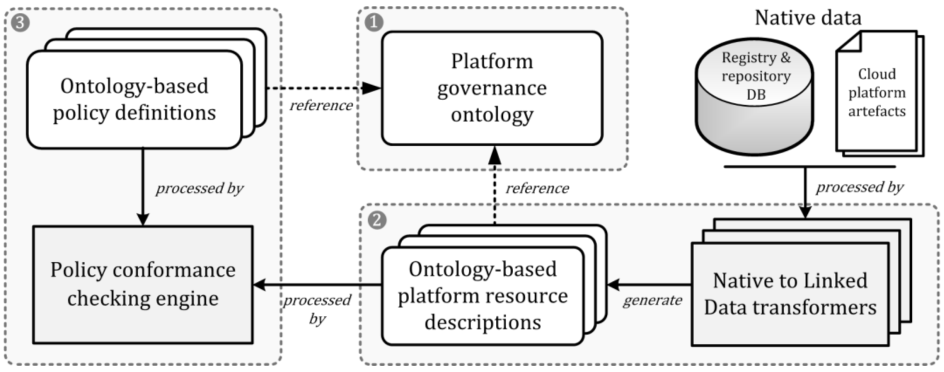
3
Ontology-based policy definitions
reference
processed by
Policy conformance checking engine
1
Platform governance ontology
reference
Native data
Registry & repository DB
Cloud platform artefacts
processed by
2
Ontology-based platform resource descriptions
processed by
generate
Native to Linked Data transformers

Figure 1. Overview of approach for ontology-based definition and enforcement of governance policies.

In the following subsections we discuss some interesting aspects of those components in greater detail, placing more emphasis on the third one.

4.2. Platform governance ontology

The foundation for policy representation is our *platform governance ontology*, which defines modelling constructs corresponding to the different types of logical entities found on the CAST platform, such as different kinds of software units (solutions, apps and services), different kinds of software artefacts (e.g. deployment descriptors, interface definitions, pricing specifications, localisation files, images), artefact collections, lifecycle states (development, testing, review, beta, production, deprecation, end-of-life), and many more. In addition, it defines concepts corresponding to the various attributes of platform resources as well as the ways those resources are interrelated. For reasons of interoperability and tool support, the language we have adopted for developing the platform governance ontology is OWL2 [14].

4.3. Ontology-based platform resource description

For policy conformance checking to be feasible through a generic and universal method, the heterogeneous platform resources that are subject to governance must be described in an abstract and homogeneous manner. Descriptions are extracted from the multiple forms in which platform resources are natively represented to create Linked Data, using the platform governance ontology as the main reference vocabulary.

The term Linked Data refers to a set of best practices for publishing and connecting structured data using key Web technologies: URI, HTTP, and RDF [15]. The way in which Linked Data are represented is not determined by how policies are encoded or how policy conformance checking algorithms operate. In fact, the usage scenarios for the Linked Data produced by this process can include much more than just policy enforcement.

4.4. Ontology-based policy definition and conformance checking

Checking the above ontology-based descriptions of platform resources against policies is a task which can generally be viewed as (at least) two different kinds of computational problem: a problem of integrity constraint validation on ontology objects, or a problem of ontology object classification. Depending on the adopted approach one must implement the appropriate strategy for defining policies and checking data against these policies.

When policy checking is cast as an integrity constraint validation problem, the strategy is to define governance policies in the form of first-order logic queries using an ontology-based query language such as SPARQL [16] or SQWRL [17]. Conformance checking can thus be reduced to query answering. If a query returns a non-empty result set, it means that the returned ontology objects violate the integrity constraints specified in the query, i.e. they do not conform to the respective policy.

When policy checking is cast as an object classification problem, the strategy is to define governance policies as Description Logic (DL) class axioms, so as to reduce the task of conformance checking to instance checking with an OWL DL reasoner. Instance checking is a basic service provided by every DL reasoner to answer if a given individual is an instance of a specified class [18]. As we discuss next, due to certain characteristics of the OWL language this strategy requires an additional pre-processing step before instance checking is actually applied on any particular ontology object.

The relevant literature on OWL and RDF data validation provides examples of both approaches in use, such as [19] and [20], which discuss the first and the second approach, respectively. Naturally, each of the two approaches has its own advantages and disadvantages for particular application domains, discussing which is beyond the scope of this paper. In our work so far we have primarily been investigating the second approach; however, we aim to eventually incorporate both strategies in our framework for cloud platform governance.

One of the questions that surfaced on the outset of this work was whether the expressivity of OWL2 DL (i.e. $\mathcal{SROIQ}^{(\mathcal{D})}$) would prove sufficient for representing all of the policies in the CAST project dataset as OWL class axioms, or whether it would be necessary to step outside OWL2 DL boundaries. Indeed, the representation of some CAST policies proved to be demanding in terms of expressivity, and required the use of some SWRL rules [21]. For the rest of the policies the less expressive DL $\mathcal{ALCOIQ}^{(\mathcal{D})}$ has been sufficient. The need to step outside DL and include Horn-clauses in SWRL was mitigated by the easy integration of SWRL with OWL. SWRL rules and OWL ontologies share a common semantics, and can be serialised together. Furthermore, SWRL is a de facto standard supported by many reasoning engines (such as Pellet and HermiT), so combining OWL DL axioms and SWRL rules does not present any serious practical or theoretical obstacles.

A more serious challenge was posed by OWL's Open World Assumption (OWA) and the lack of any Unique Name Assumption (UNA) in its standard semantics, since these features militate against using a standard OWL reasoner to perform data validation. Consistently with the OWA, a standard OWL reasoner will never infer that some statement is false, simply because there is no evidence to support the truth of that statement. By contrast, data validation requires reasoning under a Closed World Assumption (CWA), whereby a reasoner would conclude that a statement is false when there is no evidence to support it. The absence of a UNA allows an OWL reasoner to infer that two differently-named objects may in fact be identical, which may have unexpected consequences where distinctness is desired. As a result, employing a standard OWL reasoner "out of the box" to perform policy conformance checking is not feasible. These issues are well known to the Semantic Web community and have been recognised as obstacles to using OWL for data validation purposes [22, 23, 24].

A solution is to enable some form of local closed world (LCW) reasoning, i.e. to close the world relative only to the descriptions of platform resources we are interested in checking, while leaving the rest of the KB to be processed under the standard OWA. As mentioned in [25], this can be accomplished by adding extra assertions on an object of interest, to state that all the information relevant to that object is known.

To illustrate how we approximate local closed world reasoning to overcome the absence of CWA and UNA in OWL, let us consider the example policy mentioned in section 2 of this paper: the artefact governance policy stating that the interface description document of every external web service (i.e. its WSDL file) should contain exactly two non-identical endpoints. Let this policy be represented by an equivalence class axiom as in (1):

$$\text{ValidServiceInterface} \equiv \text{ServiceInterface} \sqcap (= 2\ \text{contains.Endpoint}) \qquad (1)$$

Let us further assume a Knowledge Base $\mathcal{K}$ containing ontology instance data assertions as in (2):

$$\mathcal{K} = \{\text{ServiceInterface}\,(s), \text{Endpoint}(e_1), \text{Endpoint}(e_2), \text{contains}(s, e_1), \text{contains}(s, e_2)\} \qquad (2)$$

We would like to have the reasoner infer $\text{ValidServiceInterface}(s)$, which is a way of saying that individual s belongs to the class of valid WSDL documents. However, without any UNA, the combination of (1) and (2) does not entail this. Despite the fact that s is known to contain the endpoints e_1 and e_2, there is nothing to preclude that e_1 and e_2 is the same individual. To compensate for OWL's lack of a Unique Name Assumption, we need to extend $\mathcal{K}$ by asserting explicitly that e_1 and e_2 are different individuals (3):

$$e_1 \neq e_2 \qquad (3)$$

Even with this addition, it is still not yet possible to have s classified under the anonymous class of things that contain exactly two endpoints $(= 2\ \text{contains.Endpoint})$. Under OWL's Open World Assumption, the cardinality restriction in the class expression can be matched only if we have explicit knowledge that e_1 and e_2 are in fact the only objects related to s along the *contains* property (otherwise, s could be related to more, as yet unseen objects). The way to achieve this is by extending $\mathcal{K}$ with an anonymous type assertion as in (4):

$$(= 2\ \text{contains}.\top)\ (s) \qquad (4)$$

The addition of (3) and (4) to the KB is a way of closing the world relative to part of the KB (i.e. relative to s only) and in isolation from other ontology individuals. The addition of such special-purpose assertions does not need to be permanent — they can be discarded as soon as conformance checking for *s* has been completed. Moreover, the assertions do not need to be custom-coded or predefined in templates. They can be dynamically generated, as a pre-processing step within the policy checking engine. This is achieved by an algorithm which examines the equivalence class axiom representing a policy of interest, determines which (asserted or inferred) properties are relevant for classification, constructs anonymous type assertions with the exact known cardinality per each property of importance, and adds those to the object to be checked.

5. Related work

The term "policy" appears to be rather overloaded in computer science literature [12]. It has been used in connection with several different notions, such as security, trust management, action languages, business rules, and quality of service. Despite the amount of existing work in the general area, there seems to be no previous research on ontology-based approaches for lifecycle and artefact governance policy.

Closely related work has been carried out on theory and applications for OWL and RDF data validation. Motik, Horrocks and Sattler [24] have proposed an extension of OWL with Integrity Constraints (IC) similar to those found in relational databases. Their approach allows a subset of TBox axioms to be designated as ICs, which are interpreted in the spirit of relational database constraints during ABox reasoning. Tao, Sirin, Bao, and McGuinness [19] also describe an alternative IC semantics for OWL, based on CWA and weak UNA. Their approach allows developers to augment OWL ontologies with IC axioms and combine open world reasoning with closed world constraint validation. They also show that, under certain conditions, IC validation can be reduced to query answering through SPARQL queries which are automatically generated from OWL DL class axioms. SPARQL Inferencing Notation (SPIN) [26] has similar objectives. It allows ontology class definitions to be linked to SPARQL queries in order to capture constraints and rules that formalize the expected behaviour of objects belonging to those classes.

A related tool implementation is presented by Rieckhof, Dibowski and Kabitzsch [27], who are interested in formal validation techniques for ontology-based electronic device descriptions. They describe the implementation of a validator that checks for consistency, completeness and correctness in device descriptions using SPARQL queries. Miksa, Sabina and Kasztelnik [20] present a prototype system for ontology-based modelling of network devices. Their motivation is to detect configuration errors and to propose combinations of compatible devices by means of instance checking and other DL reasoning services. They achieve this by implementing a method similar to our own in order to "close" the world and thus be able to detect configuration errors, while keeping the rest of the KB "open" in order to properly reason about combinations of compatible network devices [28].

Another stream of research which is relevant to this work has been looking into applications of Linked Data and semantic technologies in general for improving systems management in cloud environments. Haase et al [29] describe the challenges related to intelligent information management in enterprise clouds and discuss how semantic technologies have been leveraged to address those challenges in the commercial eCloudManager system developed by fluidOps. In [30], Feridun and Tanner from IBM describe an approach and architecture for the transformation of diverse network and server management data into Linked Data, which allows data centre operators to easily browse, search and query data across multiple sources. Lastly, Joshi [31] describes some initial work towards a policy-based framework facilitating the automation of the lifecycle of virtualized services, using ontologies and Semantic Web technologies like OWL, RDF and SPARQL.

6. Conclusions

Through our work on the cloud application platform developed by the CAST project we gained some valuable insights into the limitations of state of the art governance tools in relation to policy engineering. We particularly note the lack of separation of concerns between policy definition and policy enforcement, the lack of domain-level abstraction in the encoding of policies, and the lack of formal representation of policy constraints and relationships.

This paper reports on our ongoing work towards a new approach to policy-driven governance in cloud application platforms, which aims to improve lifecycle governance and artefact governance through ontology-based policy definition and enforcement. We presented an overview of our approach and focused on the policy modelling methodology which allows us to reduce the problem of policy conformance checking to instance checking with an OWL2 DL reasoner. We also discussed ontology language expressivity requirements and explained the interplay of OWL's Open World Assumption and lack of Unique Name Assumption with respect to the goal of policy conformance checking.

Based on our results to date, the proposed approach appears to be a viable methodology for the definition and enforcement of policies, and promises to provide significant improvements with respect to governance in open service delivery platforms. As future work we plan to elaborate on our method for automated closure axiom generation, and to extend the policy checking engine so that instance data can also be checked by way of closed world query answering.

References

[1] D. Kourtesis, V. Kuttruff and I. Paraskakis, "Optimising development and deployment of enterprise software applications on PaaS: the CAST project," in *ServiceWave 2010 Workshops – Towards a Service-Based Internet*, LNCS vol. 6569, Springer, 2011, pp. 14–25.

[2] D. Kourtesis and I. Paraskakis, "Governance in cloud platforms for the development and deployment of enterprise applications," presented at the IEEE CloudCom 2011 – 3rd IEEE International Conference on Cloud Computing Technology and Science, Athens, Greece, 2011.

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

[4] MuleSoft Galaxy. [Online]. Available: http://www.mulesoft.org/galaxy

[5] WSO2 Governance Registry. [Online]. Available: http://wso2.com/products/governance-registry/

[6] IBM WebSphere Service Registry and Repository. [Online]. Available: http://www.ibm.com/software/integration/wsrr/

[7] Oracle Enterprise Repository and Service Registry. [Online]. Available:

http://www.oracle.com/technetwork/topics/soa/index-085709.html

[8] L. Kagal, T. Finin and A. Johshi, “A Policy Language for a Pervasive Computing Environment,” in *Proc. of the 2003 IEEE Workshop on Policies for Distributed Systems and Networks*, IEEE Computer Society, 2003, pp. 63–74.

[9] G. Tonti, J. Bradshaw, R. Jeffers, R. Montanari, N. Suri and A. Uszok, “Semantic Web Languages for Policy Representation and Reasoning: A Comparison of KAoS, Rei and Ponder,” in *International Semantic Web Conference 2003*, LNCS vol. 2870, Springer, 2003, pp. 419–437.

[10] A. Uszok, J. Bradshaw, J. Lott, M. Breedy, L. Bunch, P. Feltovich, M. Johnson and H. Jung, “New Developments in Ontology-Based Policy Management,” in *Proc. of the 2008 IEEE Workshop on Policies for Distributed Systems and Networks*, IEEE Computer Society, 2008, pp. 145–152.

[11] A. Uszok, J. Bradshaw, R. Jeffers, M. Johnson, A. Tate, J. Dalton and S. Aitken, “KAoS Policy Management for Semantic Web Services,” *IEEE Intelligent Systems*, vol. 19, no. 4, 2004, pp. 32-41.

[12] G. Antoniou, M. Baldoni, P. A Bonatti, W. Nejdl and D. Olmedilla, “Rule-based Policy Specification,” in *Secure Data Management in Decentralized Systems*, vol. 33, part III, Springer, 2007, pp. 169–216.

[13] T. Heath and C. Bizer, “Linked Data: Evolving the Web into a Global Data Space,” in *Synthesis Lectures on the Semantic Web: Theory and Technology*, vol. 1, no. 1, Morgan and Claypool, 2011, pp. 1-136.

[14] OWL 2 Web Ontology Language Primer. W3C Recommendation. [Online]. Available: http://www.w3.org/TR/owl2-primer/

[15] C. Bizer, T. Heath and T. Berners-Lee, “Linked Data—The Story So Far,” *International Journal On Semantic Web and Information Systems*, vol. 5, no. 3, 2009, pp. 1–22.

[16] SPARQL Query Language for RDF. W3C Recommendation. [Online]. Available: http://www.w3.org/TR/rdf-sparql-query/

[17] M.J. O’Connor and A. Das, “SQWRL: a Query Language for OWL,” in *6th International Workshop on OWL: Experiences and Directions*, Chantilly, VA, 2009.

[18] D. Nardi and R.J. Brachman, “An introduction to description logics,” in *The Description Logic Handbook. Theory, Implementation and Application*, F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi & P. Patel-Schneider, Eds., Cambridge University Press, 2010, pp. 1–43.

[19] J. Tao, E. Sirin, J. Bao, and D.L. McGuinness, “Integrity constraints in OWL,” in *Proc. of the 24th AAAI Conference on Artificial Intelligence (AAAI-10)*, Atlanta, Georgia, USA, 2010.

[20] K. Miksa, P. Sabina, and M. Kasztelnik, “Combining ontologies with domain specific languages: a case study from network configuration software,” in *Reasoning Web 2010. Semantic Technologies for Software Engineering*, LNCS vol. 6325, Springer, 2010, pp. 99–118.

[21] SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission. [Online]. Available: http://www.w3.org/Submission/SWRL/

[22] A. Krisnadhi, K. Sengupta and P. Hitzler, “Local closed world semantics: Keep it simple, stupid,” in *Proc. of the 2011 Int. Workshop on Description Logics*, CEUR, 2011, vol. 745.

[23] J. Tao, L. Ding, and D.L. McGuinness, “Instance data evaluation for semantic web-based knowledge management systems,” in *Proc. of the 42nd Hawaii Int. Conf. on System Sciences (HICSS’09)*, IEEE Computer Society, 2009, pp. 1–10.

[24] B. Motik, I. Horrocks and U. Sattler, “Adding integrity constraints to OWL,” in *4th International Workshop on OWL: Experiences and Directions*, Washington, DC, 2007.

[25] E. Sirin. “Data validation in OWL integrity constraints,” in *RR 2010 - Web Reasoning and Rule Systems*, LNCS 6333, Springer, pp. 18–22, 2010.

[26] SPIN: SPARQL Inferencing Notation. W3C Member Submission. [Online]. Available: http://www.w3.org/Submission/spin-overview

[27] F. Rieckhof, H. Dibowski and K. Kabitzsch, “Formal validation techniques for Ontology-based Device Descriptions,” in *Proc. of the 16th IEEE Int.* Conference on *Emerging Technologies & Factory Automation*, 2011, pp. 1–8.

[28] S. Zivkovic, K. Miksa and H. Kuhn, “A Modelling method for consistent physical devices management: An ADOxx case study,” in *Advanced Information Systems Engineering Workshops*, LNBIP vol. 83, Springer, 2011, pp. 104–118.

[29] P. Haase, T. Mathass, M. Schmidt, A. Eberhart and U. Walther, “Semantic technologies for enterprise cloud management,” in *ISWC 2010*, LNCS vol. 6497, Springer, 2010, pp. 98–113.

[30] M. Feridun and A. Tanner, “Using linked data for systems management,” in *Proc. of 2010 IEEE/IFIP Network Operations and Management Symposium*, 2010, pp. 926 –929.

[31] K.P. Joshi, “DC proposal: automation of service lifecycle on the cloud by using semantic technologies,” in *ISWC 2011*, LNCS vol. 7031, Springer, 2011, pp. 285–292.