Distinguishing Obligations and Violations in Goal-based Data Usage Policies

Sebastian Speiser
Karlsruhe Service Research Institute (KSRI)
Institute of Applied Informatics and Formal Description Methods (AIFB)
Karlsruhe Institute of Technology (KIT), Germany
Email: speiser@kit.edu

***Abstract*—Laws, regulations and contracts often allow actions, such as usages of data artefacts, under the condition that a set of obligations is fulfilled. Formalising such allowances as policies enables the automated testing whether actions are compliant. Previous approaches to formalise obligations treat them as special objects in the underlying logic. We propose to represent obligations and other compliance conditions in a uniform way, in order to increase understandability by non-expert users. The challenge of such an approach is to differentiate between policy violations and not yet fulfilled obligations. We present a solution based on abductive reasoning, which is described in general terms for policy languages based on first-order logic (FOL). Furthermore, we discuss the use of decidable fragments of FOL as a base for practical policy languages.**

***Keywords*-policy; obligations; usage policies**

I. Introduction

Processes that have to comply with laws, regulations, norms, licenses, and contracts are ubiquitous. Data owners can restrict processes that use their data based for example on privacy law or copyright law. One important aspect of such restrictions are obligations. Obligations are duties that have to be fulfilled, when a right is exercised. Consider for example that it is allowed for a blogger to reuse an image in a non-commercial blog post, however the blogger is obliged to add an attribution of the original creator of the image to his post. Here, the obligations are clearly separated from other restriction, e.g., that the post must be non-commercial. This separation is also found in existing approaches to formalise such restrictions as computer-understandable policies, where obligations are modelled with special operators. Formalisations are useful to automate compliance checks in order to enable systems that adhere to restrictions or assist human users to do so. Special obligation operators that specify, which actions have to be performed to reach compliance, have two drawbacks:

- The policy remains at a lower conceptual level than goal-based policies, which only describe compliant states, and leave the computation of required actions to the policy engine.
- Special operators in a language or its underlying logic mean additional effort for non-expert users to understand them and use them correctly.

In this paper, we propose a novel approach to represent policy conditions and obligations in a uniform way as goal-based policies. Definitions of obligations are given specific for each application in the corresponding domain vocabulary, defined and understood by the system users. Our approach is defined and described in abstract terms using first-order logic and abductive reasoning. We also discuss concrete policy languages that can be used to apply the theoretical results to practical problems.

The rest of the paper is structured as follows: in Section II, we introduce a motivating use cases for formalising policies with obligations. Goal-based data usage policies are introduced in Section III. Our core approach is explained in Section IV. Throughout the technical parts, we go through one continuous example from the use case to illustrate the introduced concepts. In Section V, we discuss practical policy languages and the realisation of the use case. Finally, we discuss related work in Section VI, and conclude in Section VII.

II. Use Case: Restricted Data Usage

Usage of data artefacts can be restricted on the foundation of copyright and privacy laws, company internal guidelines or social norms. We consider usage policies of data artefacts as the formal specification, which usages are allowed and which conditions apply. For formalizing policies, we need a vocabulary for describing data usages, which is visualised in Figure 1. The vocabulary describes Artefacts that can be used in Processes. An artefact has a Policy, to which processes using the artefact must comply. Processes are divided in (i) Usages, which consume an artefact for a specific Purpose, and (ii) Derivations, which generate new artefacts on the base of the used artefacts. A process a can trigger another process b, meaning that the execution of a will lead to the execution of b. We present the following examples of conditions on allowed data usages:

- A derivation of an artefact with usage policy p is allowed, if the generated artefact will also be assigned the same policy p. Such conditions are used, e.g., in Creative Commons ShareAlike licenses.

 ISBN: 978-1-61208-200-4

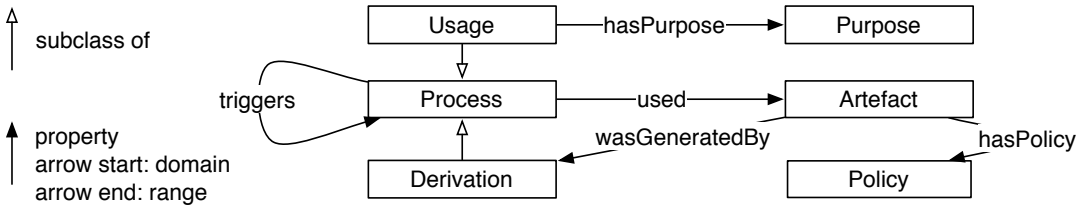

Figure 1. Vocabulary for Data Usage Restrictions

- A usage of an artefact is allowed, if the usage is for non-commercial purposes and triggers an attribution of the artefact's creator. Such conditions are used, e.g., in Creative Commons NonCommercial (NC), Attribution (BY) licenses (abbreviated as BY-NC).
- A usage of an artefact (e.g., a electronic health record) is allowed by doctors, who can also store the artefact under the condition that it is deleted after one year.

III. Goal-based Data Usage Policies

In computer science, the notion of a *policy* refers to a formal description of the actions and behaviors that are allowed or required in a protected context. The context can be characterised for example by the data artefacts that are used, properties or identities of agents involved in performing the action, or temporal constraints. Formal specifications enable the automated detection of policy violations of systems or agents that are formally described. Additionally, in many applications, required adoptions to transit from violation to compliance, can be automatically computed and realised by the corresponding system or agent. In this sense, policies can be used to formalise laws, norms and regulations that apply to a computer system, or a process realised or supported by such a system.

In our approach, we consider goal-based policies as defined by Kephart and Walsh [1]. Goal-based policies are on a high conceptual level, as they only describe the desired states of (the modelled) world, instead of specifying how such a state can be reached.

In the following, we give a general formal definition of policies based on first-order logic (FOL). We assume that the state of the world is described by the FOL theory T of the signature consisting of constants C and predicates P. A policy p is applicable to a set $S_p \subseteq C$ of policy subjects. Policy subjects can either be compliant or non-compliant with p, all other constants $c \in C \setminus S_p$ are called inapplicable. A policy p is defined by a formula $\phi_p[x]$ with one free variable. The compliant subjects are given by the set of constants that when replacing x in ϕ establish $T \models \phi$.

In the following, we restrict all given theories, formulae, and policies to stay in the Datalog fragment of first-order logic. Datalog is the FOL language of function-free Horn clauses [2] and is used as a base for many policy languages, e.g., [3], [4], [5]. Policies in our definition as formulae with one free variable can be expressed as monoid Datalog queries. Compliance checks can be solved via query evaluation, which is decidable. As we will discuss in Section V-B, also all other required operations are decidable for Datalog.

As an example, we formalise the policy BY-NC restricting data usages to trigger an attribution of the original creator (see Section II):

$$\text{BY-NC}(x) \leftarrow \mathsf{Usage}(x) \wedge \mathsf{triggers}(x,a) \wedge \mathsf{Attribution}(a) \wedge \mathsf{hasPurpose}(x,r) \wedge \mathsf{NonCommercial}(r).$$

In Datalog, the variables a and r are existentially quantified, which means that the right-hand side of the rule is a FOL formula with one free variable (x) and thus defining $\phi_{\text{BY-NC}}$.

IV. Distinguishing Obligations and Violations

In situations, where a data usage is classified non-compliant to the used artefact's policy, we have to distinguish between policy violations and not yet fulfilled obligations. Obligations are temporary violations of a policy, which will be fixed after a certain amount of time to reach compliance. Consider for example the obligation to attribute the original creator of an artefact when it is used: using the artefact is classified as non-compliant but only temporarily until the attribution is given and thus compliance reached. If usage is restricted to non-commercial purposes and a usage is classified as non-compliant because it has a commercial purpose, the violation is not temporary and thus there is not an obligation required, but the usage should be prevented. In the following, we present an approach to distinguish violations and obligations for usages classified as non-compliant to a policy.

Consider a data usage described by the theory T, where a policy subject s is found non-compliant to a policy p defined by $\phi_p[x]$. The solution is structured along the following steps:

1) finding out why s is non-compliant;
2) if s can be made compliant by adding new facts, identify the required facts;
3) identify obligations in the facts;
4) checking whether obligation handling makes the usage compliant;
5) if compliance is given, schedule the obligations with the corresponding handlers.

 ISBN: 978-1-61208-200-4

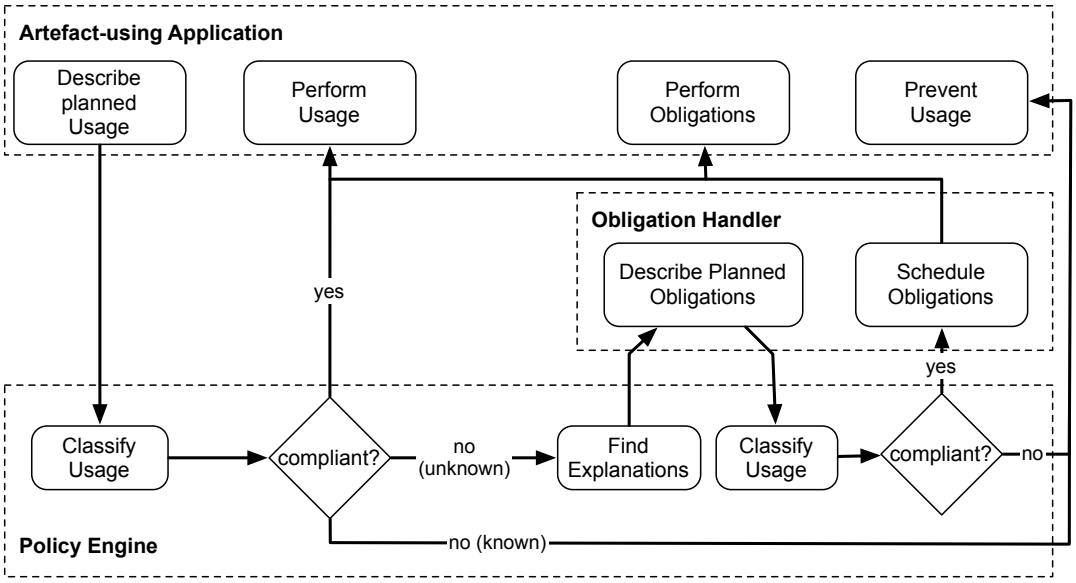

Figure 2. Architecture of a Policy-aware System for Data Usage and Automated Obligation Handling

Step 1: Finding out why s is non-compliant:
We consider s to be non-compliant with p, if we cannot infer that s makes ϕ_p true, i.e., $T \not\models \phi_p[s]$. One reason why we cannot infer $\phi_p[s]$ can be that it contradicts T, i.e., $T \models \neg\phi_p[s]$. In case of a contradiction, we cannot establish $\phi_p[s]$ by adding new facts (e.g., from describing the fulfilment of an obligation), because of the monotonicity of FOL. As such contradictions cannot be fixed by obligation handling, we only proceed if $T \not\models \phi_p[s]$ and $T \not\models \neg\phi_p[s]$.

As an example, consider the following theories describing data usages:

T_1: $\mathsf{Usage(u1)} \wedge \mathsf{hasPurpose(u1, r1)} \wedge \mathsf{NonCommercial(r1)}$.

T_2: $\mathsf{Usage(u2)} \wedge \mathsf{triggers(u2, a2)} \wedge \mathsf{Attribution(a2)}$.

T_3: $\mathsf{Usage(u3)} \wedge \mathsf{hasPurpose(u3, r3)} \wedge \mathsf{Commercial(r3)}$.

All three theories T_1, T_2, T_3 do *not* model a usage compliant to the policy BY-NC. However, $T_3 \models \neg\phi_{\text{BY-NC}}[\mathsf{u3}]$ and thus will be disregarded in further examples.

Step 2: Identify suitable theories to add:
Next, we search for a set $\mathcal{E}$ of theories that make s compliant to p. The search naturally translates into a problem that can be solved by abductive reasoning. The term of abductive reasoning goes back to Peirce [6] and refers to finding an explaining hypothesis for a circumstance. In other words, for a given observation b find an explanation a from which b can be logically inferred. In this sense, abduction is the reverse of of deduction, where b is found for a given a. Abductive reasoning was applied to formal logics and several algorithms were given for various logic formalisms (e.g., [7], [8], [9], [10]). In the following, we formally define our understanding of abductive reasoning for FOL. Given a theory T and a set F of atomic facts (formulae of the form $p(c_1, \ldots, c_n)$, where $p \in P$ is a predicate of arity n, and each $c_i \in C$ is a constant), find an explanation E, such that F can be inferred from T and E, or more formally: $T \cup E \models F$. Additionally, we require that there exists an interpretation for $T \cup E$, i.e., $T \cup E$ is consistent. For sake of simpler notation, we also apply abduction to find an explanation for a sentence $\phi[c]$, where $\phi[x]$ is a formula with the only free variable x. This can be realised by introducing a fresh unary predicate p' and the axiom $\forall x.p'(x) \leftrightarrow \phi[x]$; then abduction can be applied to finding an explanation for the atomic fact $p'(c)$. Applying abduction to our problem of finding suitable theories for making s compliant to p, we search a set $\mathcal{E}$, such that: $\forall E \in \mathcal{E}. T \cup E \models \phi_p[s]$. We require that every explanation E is minimal in the sense that there is no other explanation E' which entails E and there is no subtheory of E, which is also an explanation:

$$\forall E \in \mathcal{E}.\ \nexists E' \in \mathcal{E}. E \neq E' \wedge T \cup E' \models T \cup E.$$
$$\forall E \in \mathcal{E}.\ \nexists E'. E' \subseteq E \wedge T \cup E' \models \phi_p[s].$$

The set of explanations can still be of infinite size, e.g., because of transitive predicates, and the minimality conditions might not always be desired [7]. We leave the exact definition of the explanations selected for $\mathcal{E}$ open to be specified for concrete applications. Similarly, there maybe a system-specific preference order on the explanations, therefore we describe the following steps for a single explanation $E \in \mathcal{E}$.

Continuing the previous examples, we choose the following explanation E_1, E_2 such that $T_1 \cup E_1 \models \phi_{\text{BY-NC}}[\mathsf{u1}]$ and

 ISBN: 978-1-61208-200-4

$T_2 \cup E_2 \models \phi_{\text{BY-NC}}[\mathsf{u2}]$:

E_1: $\mathsf{triggers}(\mathsf{u1}, \mathsf{a1}) \wedge \mathsf{Attribution}(\mathsf{a1})$.
E_2: $\mathsf{hasPurpose}(\mathsf{u2}, \mathsf{r2}) \wedge \mathsf{NonCommercial}(\mathsf{r2})$.

Step 3: Identification of obligations:
An explanation E contains facts that would make s compliant to p. Not all of the facts in E however can be fulfilled by adding the description of an obligation, but could only be the result of complying to an unfulfilled condition (see example below). Depending on the specific application, we thus define a set $\mathcal{O}$ of obligations, and for every obligation $o \in \mathcal{O}$ a query $q_o(p_1, \ldots, p_n)$ and an obligation handler h_o. The query q_o defines, which kind of required facts can be handled by the corresponding obligation handler h_o. In our example, we define one obligation $o1$ with a handler h_{o1} that can automatically add attributions to data usages. The corresponding query q_{o1} is defined as:

$$q_{o1}(x, a) \leftarrow \mathsf{triggers}(x, a) \wedge \mathsf{Attribution}(a).$$

The bindings for the query are passed to the obligation handler h_o, which will return a FOL theory T' that describes the planned fulfilment of the obligations identified by the bindings. In our example, for E_1 the query q_{o1} gives the binding $\{x \mapsto \mathsf{u1}, a \mapsto \mathsf{a1}\}$, for which the handler h_{o1} plans to create an attribution action, described by the returned theory:

$$T_1'\text{: } \mathsf{triggers}(\mathsf{u1}, \mathsf{a1}') \wedge \mathsf{Attribution}(\mathsf{a1}').$$

For the explanation E_2, the query q_{o1} gives no bindings, and thus the obligation handler only returns the empty theory T_2'.

Step 4: Checking if obligation handling leads to compliance:
After getting the descriptions of the planned obligation fulfilments, we want to ensure that fulfilling them is sufficient to make s compliant. For this we check whether $T \cup T' \models \phi_p[s]$. If this is the case, we can proceed to the next step and schedule the planned obligation fulfilments. Otherwise, we found out that s is not only a temporary violation, but should be prevented completely. In our example, we see that $T_1 \cup T_1' \models \phi_{\text{BY-NC}}[\mathsf{u1}]$, but $T_2 \cup T_2' \not\models \phi_{\text{BY-NC}}[\mathsf{u2}]$. Thus, we prevent u2 from execution, but allow u1 and tell the obligation handler h_{o1} to schedule the attribution $\mathsf{a1}'$ (**Step 5: Obligation handling**).

A system architecture realizing the complete process of Steps 1 to 5 is visualised in Figure 2.

In order, to ensure that every obligation can be unambiguously assigned to an obligation handler, one can require that the q_o queries define pairwise disjoint sets for different obligations. Another, weaker, requirement would be that no obligation definition is subsumed by an other definition. In some systems, however, it may also be practical to pose no such requirements and have an obligation subsuming all other obligations, which has an handler that logs all obligation instances.

V. Implementation and Application

In this section, we describe how the proposed concepts can be used for realising the use case of data usage restrictions. We then argue, how two popular policy formalisms (Datalog and OWL) can be used with our approach, by describing how the required operations can be realised with standard reasoner methods. Finally, we briefly describe how we implemented the approach for Datalog-based policies.

A. Realisation of Use Case

We already discussed the Creative Commons NonCommercial, Attribution policy and its application to three different usages as a running example in the explanation of our approach. The other two policies are given in the following:

- Creative Commons ShareAlike (abbreviated SA): derived artefact should have the same policy:

$$\text{SA}(x) \leftarrow \mathsf{Derivation}(x) \wedge \mathsf{wasGenBy}(a, x) \wedge \mathsf{hasPolicy}(a, \text{SA}).$$

  Assigning an allowed target policy for a generated artefact, can be done automatically by an obligation handler h_{o2} taking the bindings of the following obligation query: $q_{o2}(a, p) \leftarrow \mathsf{hasPolicy}(a, p)$. If the obligation handler receives bindings that would assign incompatible policies to an artefact, the obligation handler returns an empty theory back, meaning that it cannot fulfil the obligations. Otherwise, it returns a theory describing that a compatible policy is assigned to the artefact, which is scheduled in case that the obligation descriptions make the usage compliant.
- Electronic health record policy (abbreviated EHR): doctors can use the artefact and store it for one year:

$$\begin{aligned}\text{EHR}(x) \leftarrow &\Big(\mathsf{Usage}(x) \wedge \mathsf{performedBy}(x, a) \wedge \mathsf{Doctor}(a)\Big) \vee \\ &\Big(\mathsf{Storage}(x) \wedge \mathsf{performedBy}(x, a) \wedge \mathsf{Doctor}(a) \wedge \\ &\quad \mathsf{triggers}(x, d) \wedge \mathsf{Deletion}(d) \wedge \\ &\quad \mathsf{performedAt}(d, t) \wedge t \leq \text{now() + 1y}\Big).\end{aligned}$$

  To a storage action, which is classified as non-compliant, at least one of the following applies: (i) it is not performed by a doctor, or (ii) there is no deletion scheduled. The former cannot be handled as an obligation: allowing only doctors access to the health record is a hard constraint, which cannot even temporarily be violated. In contrast, an automated deletion can be scheduled by an obligation handler in the future, making the storage action compliant. The corresponding obligation query is given as $q_{o3}(d, t) \leftarrow \mathsf{Deletion}(d) \wedge \mathsf{performedAt}(d, t)$.

 ISBN: 978-1-61208-200-4

B. Applicability to Concrete Policy Formalisms

We used a Datalog-based policy formalism in this work, as it is a popular choice for policy languages and has desirable computational properties: compliance checks and obligation identification can be solved via query evaluation, which is decidable. Checking whether an obligation query is subsumed by another query is a query containment problem, which can be reduced to query evaluation [11]. Testing whether obligation queries are disjoint is equivalent to testing disjointness of database views and queries, for which algorithms exist [12]. Finally, there exist numerous approaches to abductive reasoning that can be applied to Datalog, e.g., [8], [10].

Other fragments of FOL, for which practical tools exist, are represented by Description Logics [13], namely the Web Ontology Language (OWL) and its profiles [14]. OWL is also used for policy languages (e.g., [15], [16]) and defines concepts, which correspond to formulae with one free variable, and thus is compatible to our approach. Standard inference tasks for OWL reasoners cover almost all required tasks for our procedure presented in this paper: instance classification (for compliance checks and obligation identification), class subsumption (for checking subsumption of obligation queries) and class disjointness checks (for testing disjointness of obligation queries). Missing is only abduction, which is not regarded as a standard task, but solved by several approaches, e.g., [9].

C. Implementation

We developed a prototypical implementation of our approach for Datalog-based policy languages, as described in this paper. The prototype uses the DLV System [17] for compliance classification and a custom obligation handling system based on the abductive reasoning engine HYPROLOG [18] running on SWI-Prolog [19]. The prototype is not optimised, but is able to classify simple examples based on our use case in less than 0.2 seconds and find and handle obligations in less than 1 second on a 2.4 GHz standard laptop computer. The conducted measurements show that an integration into a fast *design, compliance check, modification life cycle* is possible. More extensive performance measurements will be conducted in future work, when the policy and obligation engine is integrated into a concrete policy-aware system for exposing compositions of data artefacts on the Web.

VI. Related Work

As noted before, the term of abductive reasoning goes back to Peirce [6] and many technical solutions for different logic formalisms were developed, e.g., [7], [8], [10], [9]. Related to our task to find out the reasons for a policy non-compliance are so-called *why not*, respectively *how to* questions [20]. Becker and Nanz explicitly mention the use of abductive reasoning in policy systems to determine what is missing to reach compliance [8]. Not targeted at policies but at formal knowledge systems in general is the work of Chalupsky and Ross for answering *why not* queries, i.e., giving reasons why some desired inference does not hold [21]. The applications of explanations and abduction to policies have in common that they aim at helping the user to reach compliance. Our goal is to automatically identify obligations and pass them to an obligation handler. Not all missing pieces described by an explanation can just be regarded as obligations, but could also be violations of the policy. Finding out, which pieces are obligations and whether they cover the full explanation is a non-trivial task for a policy-based system, for which we presented to the best of our knowledge, the first solution.

Xu and Fong present a policy language with obligations [22], for which they list a set of requirements taken from surveying obligation policy languages in the literature, including [23], [24], [25], [26], [27]. In contrast to the languages analysed by Xu and Fong and the language they propose, there are no special logic operators for representing obligations in our approach. Instead, domain- and application-specific types of obligations can be defined. We model only desired goal states, i.e., the states compliant to a policy, and leave computation of what has to be done to reach compliance (including the fulfilment of obligations) to the policy system. This is in contrast to the other approaches, which specify the actions that have to be performed directly using the obligation operators. In the following, we describe how the requirements identified by Xu and Fong [22] are handled by our approach:

- *Trigger* and *obligation*: define under which conditions the obligation is applicable, and what the obligation is. In our approach, both are described in one logical formula specifying the desired and compliant goal states.
- *Temporal constraint*: specifies the time span in which an obligation should be fulfilled. In our approach, this can be modelled, if needed, as part of the domain ontology. Depending on the application, different models of time spans can be employed, e.g., (i) attribution must be given at the same time as usage, or (ii) deletion of artefact must take place latest one year after it was stored.
- *Penalty* or *reward*: what happens if the obligation is violated (*penalty*), respectively fulfilled (*reward*). A penalty can just be modelled as another possibility to reach compliance, namely by executing the protected actions and fulfilling the penalty instead of the obligation. A reward is simply a more relaxed policy, i.e., allowing more actions if the obligation is also fulfilled.

VII. Conclusions

We presented a novel approach to formalise obligations and other compliance conditions in a uniform way. The

 ISBN: 978-1-61208-200-4

approach enables goal-based policies on a high conceptual level without the need for users to learn special operators in the policy language. Instead definitions of obligations can be specified using domain- and application-specific vocabularies that are defined and understood by the users. Explanations about what a policy subject lacks to compliance are found by abductive reasoning. We presented a novel method to check whether an explanation is fully covered by obligations and to identify the obligations.

For future work, we plan to integrate the implemented method in a concrete application, realising the automated handling of obligations when using data with restricted usages to create new services and data sources.

Acknowledgements: This work was supported by the EU project PlanetData (FP7 - 257641).

REFERENCES

[1] J. O. Kephart and W. E. Walsh, "An Artificial Intelligence Perspective on Autonomic Computing Policies," in *Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks*, 2004, pp. 3–12.

[2] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*. Addison Wesley, 1994.

[3] P. A. Bonatti, J. L. De Coi, D. Olmedilla, and L. Sauro, "A Rule-based Trust Negotiation System," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, pp. 1507–1520, 2010.

[4] C. Ringelstein and S. Staab, "PAPEL: A language and model for provenance-aware policy definition and execution," in *Proceedings of 8th International Conference on Business Process Management (BPM)*, ser. LNCS, R. Hull, J. Mendling, and S. Tai, Eds., vol. 6336. Springer, 2010, pp. 195–210.

[5] M. Y. Becker, C. Fournet, and A. D. Gordon, "SecPAL: Design and Semantics of a Decentralized Authorization Language," *Journal of Computer Security (JCS)*, vol. 18, pp. 597–643, 2010.

[6] C. S. Peirce, "Abduction and Induction," in *Philosophical writings of Peirce*. Dover Publications, 1955.

[7] D. Poole, "Explanation and prediction: an architecture for default and abductive reasoning," *Computational Intelligence*, vol. 5, pp. 97–110, May 1989.

[8] M. Y. Becker and S. Nanz, "The Role of Abduction in Declarative Authorization Policies," in *Proceedings of the 10th International Symposium on Practical Aspects of Declarative Languages (PADL)*, ser. LNCS, P. Hudak and D. Warren, Eds., vol. 4902. Springer, 2008, pp. 84–99.

[9] S. Klarman, U. Endriss, and S. Schlobach, "Abox abduction in the description logic $\mathcal{ALC}$," *Journal of Automated Reasoning*, vol. 46, pp. 43–80, 2011.

[10] U. Endriss, P. Mancarella, F. Sadri, G. Terreni, and F. Toni, "The CIFF Proof Procedure for Abductive Logic Programming with Constraints," in *European Conference on Logics in Artificial Intelligence*, ser. LNCS, J. Alferes and J. Leite, Eds. Springer, 2004, vol. 3229, pp. 31–43.

[11] S. Abiteboul and O. M. Duschka, "Complexity of answering queries using materialized views," in *Proceedings of the 7th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*. ACM, 1998, pp. 254–263.

[12] M. W. Vincent, M. Mohania, and M. Iwaihara, "Detecting privacy violations in database publishing using disjoint queries," in *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology (EDBT)*. ACM, 2009, pp. 252–262.

[13] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, Eds., *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[14] W3C OWL Working Group, *OWL 2 Web Ontology Language: Document Overview*. W3C Recommendation, 27 October 2009, available at http://www.w3.org/TR/owl2-overview/.

[15] M. Krötzsch and S. Speiser, "ShareAlike your data: Self-referential usage policies for the Semantic Web," in *Proceedings of the 10th International Semantic Web Conference (ISWC)*, ser. LNCS, vol. 7032. Springer, 2011, pp. 354–369.

[16] A. Uszok, J. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lott, "KAoS Policy and Domain Services: Toward a Description-Logic Approach to Policy Representation, Deconfliction, and Enforcement," in *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY)*, 2003, pp. 93–96.

[17] "DLV," (Software) http://www.dlvsystem.com/dlvsystem/index.php/DLV, accessed March 14th 2012.

[18] "HYPROLOG," (Software) http://akira.ruc.dk/~henning/hyprolog/, accessed March 14th 2012.

[19] "SWI-Prolog," (Software) http://www.swi-prolog.org/, accessed March 14th 2012.

[20] P. A. Bonatti, D. Olmedilla, and J. Peer, "Advanced Policy Explanations on the Web," in *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI)*. IOS Press, 2006, pp. 200–204.

[21] H. Chalupsky and T. Russ, "Whynot: Debugging failed queries in large knowledge bases," in *Proceedings of the 14th conference on Innovative Applications of Artificial Intelligence (IAAI)*. AAAI Press, 2002, pp. 870–877.

[22] C. Xu and P. W. L. Fong, "The Specification and Compilation of Obligation Policies for Program Monitoring," Department of Computer Science, University of Calgary, Canada, Tech. Rep. 2011-996-08, April 2011, available at http://pages.cpsc.ucalgary.ca/~pwlfong/Pub/UC-CPSC-TR-2011-996-08.pdf.

 ISBN: 978-1-61208-200-4

[23] N. H. Minsky and A. D. Lockman, "Ensuring integrity by adding obligations to privileges," in *Proceedings of the 8th International Conference on Software Engineering (ICSE)*. IEEE Computer Society Press, 1985, pp. 92–102.

[24] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The Ponder Policy Specification Language," in *Proceedings of the Workshop on Policies for Distributed Systems and Networks (POLICY)*, ser. LNCS, vol. 1995. Springer, 2001, pp. 18–38.

[25] P. Gama and P. Ferreira, "Obligation Policies: An Enforcement Platform," in *IEEE Workshop on Policies for Distributed Systems and Networks (POLICY)*, 2005, pp. 203–212.

[26] K. Irwin, T. Yu, and W. H. Winsborough, "On the modeling and analysis of obligations," in *ACM Conference on Computer and Communications Security (CCS)*, 2006, pp. 134–143.

[27] M. Hilty, A. Pretschner, D. Basin, C. Schaefer, and T. Walter, "A policy language for distributed usage control," in *European Symposium on Research in Computer Security (ESORICS)*, 2007, pp. 531–546.

 ISBN: 978-1-61208-200-4