Policy Conflict Resolution in IoT via Planning

Emre Göynügür[1(✉)], Sara Bernardini[2], Geeth de Mel[3], Kartik Talamadupula[3], and Murat Şensoy[1]

[1] Ozyegin University, Istanbul, Turkey
emre.goynugur@ozu.edu.tr, murat.sensoy@ozyegin.edu.tr
[2] Royal Holloway University of London, London, UK
sara.bernardini@rhul.ac.uk
[3] IBM Research, Hampshire, UK
geeth.demel@uk.ibm.com, krtalamad@us.ibm.com

Abstract. With the explosion of connected devices to automate tasks, manually governing interactions among such devices—and associated services—has become an impossible task. This is because devices have their own obligations and prohibitions in context, and humans are not equipped to maintain a bird's-eye-view of the environment. Motivated by this observation, in this paper, we present an ontology-based policy framework which can efficiently detect policy conflicts and automatically resolve such using an AI planner.

Keywords: IoT · Semantic web · Policy · Conflict resolution · Planning

1 Introduction

Internet connected and interconnected devices—collectively referred to as *Internet of Things (IoT)*—are fast becoming a reliable and cost effective means to automate daily activities for people and organizations. This interconnection among devices—and services—not only yields to the need for representing such interactions, but also to the problem of efficiently managing them.

In traditional systems, *policies* are typically used to govern these interactions. However, most of these systems are static in nature when compared with IoT-enabled systems. In IoT, resources supporting capabilities could become available w.r.t. time, location, context, and so forth. Thus, much efficient tooling is required to handle the governance. There are a multitude of frameworks—some with rich policy representations [10], others targeting pervasive environments [5,7]. However, with respect to IoT, these frameworks are either computationally intensive or are not expressive enough to be effective.

Inspired by this observation, we present a semantically-aware policy framework based on OWL-QL [2] to effectively represent interactions in IoT as policies and an efficient mechanism to automatically detect and resolve conflicts. In the context of IoT, we predominantly observe two types of policies—*obligations* which mandates actions, and *prohibitions* which restrict actions [6]. Conflicts among such policies occur when prohibitions and obligations get applied to the

M. Mouhoub and P. Langlais (Eds.): Canadian AI 2017, LNAI 10233, pp. 169–175, 2017.
DOI: 10.1007/978-3-319-57351-9_22

same action of a device or a service at the same time. In order to provide a uniform solution to this problem, we propose and implement a mechanism which minimizes the policy violations by automatically reformulating the conflict resolution as an AI planning problem.

2 Policy Representation and Reasoning

We use OWL-QL [2], a language based on DL-Lite [1] family, to represent and reason about policies. DL-Lite has low reasoning overhead with expressivity similar to UML class diagrams. A DL-lite knowledge base $\mathcal{K}$ consists of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$, and the reasoning is performed by means of query rewriting. Due to the page limitations, we refer the reader to [1] for a detailed description on syntax and semantics of DL-Lite.

In order to motivate and to provide a consistent example throughout the document, we base our scenario in a smart home environment. Table 1 shows snippets of the TBox and the ABox of our smart home.

Table 1. Example TBox and ABox for an OWL-QL ontology.

TBox		ABox
$MobilePhone \sqsubseteq PortableDevice$	$Awake \sqsubseteq \neg Sleeping$	$Baby(John)$
$SomeoneAtDoor \sqsubseteq Event$	$Baby \sqsubseteq Person$	$Adult(Bob)$
$PortableDevice \sqsubseteq Device$	$Adult \sqsubseteq Person$	$Doorbell(dbell)$
$SoundNotification \sqsubseteq Sound \sqcap Notification$	$Speaker \sqsubseteq Device$	$Flat(flt)$
$TextNotification \sqsubseteq Notification$	$Doorbell \sqsubseteq Device$	$inFlat(Bob, flt)$
$TV \sqsubseteq \exists hasSpeaker \sqcap \exists hasDisplay$	$\exists playSound \sqsubseteq Device$	$Sleeping(John)$
$MakeSound \sqsubseteq Action \sqcap \exists playSound$	$Notify \sqsubseteq Action$	$SomeoneAtDoor(e1)$
$NotifyWithSound \sqsubseteq MakeSound \sqcap Notify$	$Awake \sqsubseteq State$	$producedBy(e1, dbell)$
$\exists hasSpeaker \sqsubseteq \exists playSound$	$Sleeping \sqsubseteq State$	$hasResident(flt, John)$
$MediaPlayer \sqsubseteq \exists playSound$		$inFlat(dbell, flt)$

Motivated by the work of Sensoy et al. [10], we formalize a policy as a six-tuple $(\alpha,\ N,\ \chi : \rho,\ a : \varphi,\ e,\ c)$ where *(1)* α is the activation condition of the policy; *(2)* N is either obligation (O) or prohibition (P); *(3)* χ is the policy addressee and ρ represents its roles; *(4)* $a : \varphi$ is the description of the regulated action; a is the variable of the action instance and φ describes a; *(5)* e is the expiration condition; and *(6)* c is the policy's violation cost.

In a policy, ρ, α, φ, and e are expressed using a conjunction of query atoms. A query atom is in the form of either $C(x)$ or $P(x, y)$, where C is a concept, P is either a object or datatype property from the QL ontology, x is either a variable or individual, and y a variable, an individual, or a data value. For instance, using variables b and f, the conjunction of atoms $Baby(?b) \wedge Sleeping(?b) \wedge inFlat(?b, ?f)$ describes a setting where there is a sleeping baby in a flat.

Table 2. Example prohibition and obligation policies

Prohibition		Obligation
$\chi : \rho$	$?d : Device(?d)$	$?d : Doorbell(?d)$
N	P	O
α	$Baby(?b) \wedge Sleeping(?b) \wedge$ $inFlat(?b, ?f) \wedge inFlat(?d, ?f)$	$SomeoneAtDoor(?e) \wedge producedBy(?e, ?d) \wedge$ $belongsToFlat(?d, ?f) \wedge hasResident(?f, ?p)$
$a : \varphi$	$?a : MakeSound(?a)$	$?a : NotifyWithSound(?a) \wedge hasTarget(?a, ?p)$
e	$Awake(?b)$	$notifiedFor(?p?e)$
c	10.0	4.0

When multiple policies act upon a device, conflicts could occur. In our context, three conditions have to hold for two policies to be in conflict: *(a)* policies should be applied to the same addressee; *(b)* one policy must oblige an action, while the other prohibits it; and *(c)* policies should be active at the same time in a consistent state according to the underlying ontology. Our example policies represented in Table 2 satisfy these conditions, thus they are in conflict.

3 Resolving Conflicts via Planning

In order to resolve conflicts, we have utilized planning techniques. We represent our policies in PDDL2.1 [3]. PDDL is considered to be the standard language for modeling planning problems which commonly consist of a domain and a problem files. The domain defines the actions and predicates while the problem defines the initial and the goal states. Below we illustrate how planning can be useful in resolving conflicts and then we outline a way to pose this conflict resolution problem as a planning problem.

3.1 Illustrative Scenario

Let us envision a situation in which an obligation to notify someone within the house is created, however, there is a prohibition on making sound. This forces the doorbell to pick between one of the two policies to violate. However, if there was another way to fulfill both of them without violating one another, the conflict and ensuing violation could be avoided. Given that we are dealing with complex domains with multiple devices and services, notification action could be achieved in multiple possible ways–e.g., instead of making a sound, a visual message could be used. In more complicated scenarios, the planner would make a decision based on violation costs; a planer can then use costs of actions and violations to create a globally optimum plan that minimizes or avoids conflicts.

3.2 PDDL Domain

We exploit the TBox which contains the concepts and their relationships to construct the planning domain. Concepts and properties are represented using

PDDL predicates. Though we are unable to perform an automatic translation of action descriptions to PDDL domain, it is possible to do so via an infrastructure by exposing device capabilities as services. A discussion on how to do so is beyond the scope of this paper.

Type feature of PDDL is suitable to encode simple class hierarchies, yet it is not sufficient to express multiple inheritance and subclass expressions with object or data properties. Thus, we represent types with PDDL predicates and encode the rules for inferences using derived predicates and disjunctions. For instance, to infer that someone is a parent we can use: `(:derived (Parent ?p) (or (hasChild ?p ?unbound_1) (Mother ?p) (Father ?p)))`

The planning problem contains a total cost function that keeps track of the accumulated cost associated with executing the found plan. In addition to the total cost, a new cost function is introduced for each different active prohibition policy. These prohibition cost functions are associated with the effects of the actions that they regulate to increase the total cost, when the policy is violated. For instance, we can encode the sound prohibition in PDDL as follows:

```
(:action NotifyWithSound :parameters (?person ?event ?device ?soundAction)
  :precondition (and (MakeSound ?soundAction) (canPerform ?device ?soundAction))
  :effect (and (gotNotifiedFor ?person ?event) (increase (total-cost) (p1Cost ?device))))
```

3.3 PDDL Problem

The instances in ABox, which contains knowledge about individual objects are mapped to the initial state and to the goal of the planning instance. For example, the atom `(canPerform dbell PlaySoundDbell)` indicates that *dbell* can produce a sound to notify when needed. Moreover, total and violation cost functions are initialized in the initial state. e.g. `(= (total-cost) 0) (= (p1Cost dbell) 10)`

We note that whenever there is a change in the world, it is reflected on the initial state. For example, when the baby wakes up, the value of function `(= (p1Cost dbell) 10)` is updated to 0 in the initial state. Recall that the goal of the planning problem is to fulfill the obligations while minimizing the total cost. However, if the final plan cost exceeds the violation cost of obligations, the planner chooses to violate the obligations instead of executing the plan.

4 Automated Translation from OWL-QL to PDDL

In order to resolve policy conflicts found through planning, we first need to represent policies in the planning domain. Below we describe how policies were translated into PDDL automatically while preserving their semantics.

The translation process starts with encoding all concepts and properties from the ontology as predicates and their inference rules as derived predicates in the PDDL domain. This allows us to represent the information in the knowledge base (KB) in the planning problem. The inference rules are generated using Quetzal [8], which is a framework to query graph data efficiently.

Next, all individuals in the KB are defined as objects in the PDDL problem. Similarly, class and property axioms of these individuals are selected from the KB and written into the initial state using PDDL predicates. We note that each entry in the KB is either a concept or a property assertion. The total cost is initially set to zero and prohibition cost functions for each policy-addressee pair are set to the corresponding violation costs. Cost functions for unaffected objects are initialized to zero. Finally, the goal state is produced by using the expiration conditions of the active obligation's instances with disjunctions. Lets assume there is another resident, *Alice*, at home. Now, it could be sufficient to notify either *Bob* or *Alice*. The goal state is defined as `(:goal (or(gotNotifiedFor bob someoneAtFrontDoor) (gotNotifiedFor alice someoneAtFrontDoor)))`

All active prohibition policies along with their bindings are encoded in the planning problem to prevent unintentional violation of other active policies while avoiding the actual conflict. In our implementation, we used a central server to processes the policies of all the connected devices for convenience. We note that for each prohibition instance, a cost function predicate is created using its name, and added to the effects of actions that the policy prohibits. For example, if $p1$ is the name of the policy, then (increase (total-cost) (p1 ?x)) statement is added into the effects of $notifyWithSound$ action. Finally, each function is initialized in the problem file.

Encoding an obligation policy becomes relatively simple when the desired goal state is already defined in the expiry conditions—i.e., the variables in the condition get bounded when the activation query is executed over the knowledge base. For example, in the case of Bob and Alice, activation condition would return two rows with different bindings; $\{?d = dbell, ?p = Bob, ?f = flt, ?e = someoneAtDoor\}$ and $\{?d = dbell, ?p = Alice, ?f = flt, ?e = someoneAtDoor\}$.

5 Evaluation

In order to evaluate our approach, we augmented our IoT framework [4] with the LAMA [9] planner; we then tested our implementation w.r.t. our running scenario of home automation. We compared our approach in the following settings: always prohibition, always obligation, and higher violation cost. We generated 60 problem files in total—i.e., 15 problem files for 4 different number of devices (2, 3, 5, 20). Our intuition here was that each newly added device favors the planning method even if they did not add a new capability.

Below we show and discuss the outcomes of our experiments—due to the page limitations, we only present results when the device numbers were 2 and 20. The abbreviations in the result tables are as follows: S_{CNT} = number of times the obligation is fulfilled, S_{AVG} = average violation cost to fulfill the obligation, S_{MAX} = max violation cost, and F stands for Failed.

Table 3 depicts the results obtained. As shown by the results, when 2 devices were used, the planner violated a policy to fulfill the obligation at least once; adding more devices (e.g., when the number of device were 20) reduced policy violations to zero. Thus, it supports our intuition—i.e., adding more devices with

Table 3. Obtained results for problems with 2 and 20 devices.

Method	S_{Cnt}	F_{Cnt}	S_{Avg}	S_{Max}	S_{Min}	F_{Avg}	F_{Max}	F_{Min}	S_{Cnt}	F_{Cnt}	S_{Avg}	S_{Max}	S_{Min}	F_{Avg}	F_{Max}	F_{Min}
Number of devices: 2									Number of devices: 20							
Prohi.	0	15	0	0	0	4	8	1	0	15	0	0	0	7	10	3
Obli.	15	0	6	10	1	0	0	0	15	0	6	10	2	0	0	0
Cost	4	11	3	6	1	4	8	1	9	6	4	8	2	6	10	3
Planning	15	0	0	3	0	0	0	0	15	0	0	0	0	0	0	0

different capabilities to the system spans the solution space for our planning problem. However, adding more devices do not necessarily affect the results of other strategies as they are not aiming to resolve conflicts from a system's perspective.

6 Conclusions

In conclusion, in this paper, we discussed how a planner could be used in a lightweight policy framework to automate policy conflict resolution. The policy framework is based on OWL-QL as it targets IoT applications, which generate large volumes of instance data, and efficient query answering w.r.t. policy representation and reasoning. We reformulated policy conflict resolution as a planning problem by encoding policies in PDDL by means of cost functions and goals. We then utilized a planner to avoid or mitigate conflicts found in plethora of policies. We then presented our initial results which scales well especially when the device numbers increase which is promising. We currently are investigating means to use user history to learn violation costs associated with policies.

References

1. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: the DL-Lite family. J. Autom. Reason. **39**(3), 385–429 (2007)
2. Fikes, R., Hayes, P., Horrocks, I.: OWL-QL: a language for deductive query answering on the semantic web. Web Semant.: Sci. Serv. Agents World Wide Web **2**(1), 19–29 (2004)
3. Fox, M., Long, D.: PDDL2.1: an extension to PDDL for expressing temporal planning domains. J. Artif. Intell. Res. **20**, 61–124 (2003)
4. Goynugur, E., de Mel, G., Sensoy, M., Talamadupula, K., Calo, S.: A knowledge driven policy framework for internet of things. In: Proceedings of the 9th International Conference on Agents and Artificial Intelligence (2017)
5. Kagal, L., Finin, T., Joshi, A.: A policy language for a pervasive computing environment. In: IEEE 4th International Workshop on Policies for Distributed Systems and Networks, June 2003
6. Kortuem, G., Kawsar, F., Sundramoorthy, V., Fitton, D.: Smart objects as building blocks for the internet of things. IEEE Internet Comput. **14**(1), 44–51 (2010)
7. Lupu, E.C., Sloman, M.: Conflicts in policy-based distributed systems management. IEEE Trans. Softw. Eng. **25**(6), 852–869 (1999)

8. Quetzal-RDF: Quetzal (2016). https://github.com/Quetzal-RDF/quetzal. Accessed 02 Oct 2016
9. Richter, S., Westphal, M.: The lama planner: guiding cost-based anytime planning with landmarks. J. Artif. Int. Res. **39**(1), 127–177 (2010). http://dl.acm.org/citation.cfm?id=1946417.1946420
10. Sensoy, M., Norman, T., Vasconcelos, W., Sycara, K.: OWL-POLAR: a framework for semantic policy representation and reasoning. Web Semant.: Sci. Serv. Agents World Wide Web **12**, 148–160 (2012)