Combining Semantic Web and IoT to Reason with Health and Safety Policies

Emre Goynugur, Murat Sensoy
Computer Science Department
Ozyegin University
Istanbul, Turkey
{*emre.goynugur, murat.sensoy*}*@ozyegin.edu.tr*

Geeth de Mel
Daresbury Lab
IBM Research
Warrington, UK
geeth.demel@uk.ibm.com

***Abstract*—Monitoring and following health and safety regulations are especially important—but made difficult—in hazardous work environments such as underground mines to prevent work place accidents and illnesses. Even though there are IoT solutions for health and safety, every work place has different characteristics and monitoring is typically done by humans in control rooms. During emergencies, conflicts may arise among prohibitions and obligations, and humans may not be better placed to make decision without any assistance as they do not have a bird's-eye-view of the environment. Motivated by this observations, in this paper, we discuss how health and safety regulations can be implemented using a semantic policy framework.We then show how this framework can be integrated into an in-use smart underground mine solution. We also evaluate the performance of our framework to show that it can cope with the complexity and the amount of data generated by the system.**

***Keywords*-Policy framework, OBDA, Planning, Conflict detection, Conflict resolution, Health and safety, Semantic web**

I. Introduction

As smart objects become an increasingly important part of our lives, it is crucial that they behave according to our social norms and preferences. Policies (or norms) are soft constraints that regulate the actions of autonomous entities—such as humans—through permissions, prohibitions, and obligations [1]. Therefore, policies stand as an intuitive and principled way for regulating actions and interactions among smart objects in Internet of Things (IoT) paradigm.

Policies can be applied in every IoT setting where actions of intelligent entities should be regulated by one or more authorities. A policy could be a simple statement of intent—e.g., *sound actions are prohibited, if the baby is sleeping* for a smart home environment or policies could be used to modify the behaviors of other entities in the environment—e.g., policies could be used to program an intelligent farming application with respect to the changing environmental conditions and the status of the crops (e.g. *start watering, if the humidity level is low*).

Monitoring and following regulations are especially important in health and safety applications in a workplace—especially in hazardous working environments. The International Labour Organization (ILO) estimate that millions of workers die every year because of occupational accidents and work-related illnesses[1], and more than 96% of these fatalities occur in low- and middle- income countries. The great majority of these accidents are preventable through adherence with existing International Occupational Health and Safety standards[2], which are not always implemented by workplaces. For example, Turkey has experienced a significant number of fatalities and serious injuries due to workplace accidents—1308 lives lost since 2000 and 13000 miners involved in accidents in 2013[3]. For these reasons, in this paper, we focus on the health and safety regulations in underground mines and how these rules can be enforced in mining facilities that use a smart mine solution[4,5]. We particularly chose this domain due to its complexity and the significance of these policies for workers' health.

There is a multitude of policy frameworks; some with rich policy representations [2], [3], [4], and some targeting pervasive environments [5], [6], [7]. However, they are either not scalable (to create and refine policy instances to large number of devices) or expressive enough to use high level concepts to describe devices and situations. To address aforementioned issues, in our previous work [8], we presented a semantic policy framework that can *(a)* define high-level policies and refine them to device- and service-level policies in context; *(b)* automatically detect policy conflicts and propose resolutions using an AI planner. .

In this paper, we briefly introduce the preliminaries of our policy framework. We then discuss how to model health and safety regulations with our policy language, how to integrate our framework into an in-use smart IoT mining solution, and show how planning could be useful in this new domain. Finally, we evaluate the performance of our implementation using official mining safety regulations and a database that we obtained from a mining company.

[1] http://ibm.biz/ilo-work-related-illnesses
[2] https://www.iso.org/iso-45001-occupational-health-and-safety.html
[3] http://www.bbc.co.uk/news/world-europe-27414972
[4] http://litumiot.com/mine-rtls-worker-tracking/
[5] www.cat.com/en_US/by-industry/mining/underground-mining

II. Preliminaries

We use OWL-QL [9], a language based on DL-Lite [10] 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 [10] for a detailed description on syntax and semantics of DL-Lite.

A DL-Lite knowledge base $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ consists of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$. Axioms of the following forms compose $\mathcal{K}$: (a) *class inclusion axioms*: $B \sqsubseteq C \in \mathcal{T}$ where B is a basic class $B := \mathsf{A} \mid \exists R \mid \exists R^-$, C is a general class $C := B \mid \neg B \mid C_1 \sqcap C_2$, A is a named class, R is a named property, and R^- is the inverse of R; (b) *role inclusion axioms*: $R_i \sqsubseteq P \in \mathcal{T}$ where $P := R_j \mid \neg R_j$; and (c) *individual axioms*: $B(\mathtt{a}), R(\mathtt{a}, \mathtt{b}) \in \mathcal{A}$ where $\mathtt{a}$ and $\mathtt{b}$ are named individuals. Table I illustrates a part of our domain ontology (TBox).

$SafeZone \sqsubseteq Region$
$ExplosionZone \sqsubseteq Region \sqcap \exists hasExplosionRisk$
$PanickedPersonnel \sqsubseteq \exists Personnel \sqcap \exists panic$
$Blaster \sqsubseteq Personnel$
$Operator \sqsubseteq Personnel$
$Driller \sqsubseteq Personnel$
$Explosive \sqsubseteq Asset$

Table I
A part of the domain ontology

Table II depicts an instantiation of this TBox—i.e., an ABox.

1	$Blaster(John)$
2	$Region(sector-2)$
3	$Operator(Bob)$
4	$inPanic(Bob, true)$
5	$inRegion(Bob, sector-2)$
6	$inRegion(John, sector-2)$
7	$methane(sector-2, 1.5)$

Table II
Example ABox.

Motivated by the work of Sensoy et al. [2], we formalize a policy as a seven-tuple (α, N, $\chi : \rho$, $a : \varphi$, e, c) where *(1)* α is the activation condition; *(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; *(6)* d is the deadline; and *(7)* 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 example, the conjunction of atoms $Personnel(?p) \wedge inRegion(?p,' Area51')$ describes the sentence "*all personnel in Area 51*" in our policy language. We extended the policy definition in our previous work [8] with a deadline field, since timeliness of actions are essential for health and safety domain.

Table III illustrates example policies for the health and safety domain; we especially focus on two types of policies—obligations and prohibitions—as they are the most typical form of policies we find in the domain. The first policy prohibits personnel from entering into the regions where the methane level is above 1.0%, while the latter obliges personnel to help a person in panic.

When multiple policies act upon an agent, 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 III satisfy these conditions, thus they are in conflict.

Early detection of conflicts is particularly important in hazardous environments like underground mines, since unnoticed conflicts may cost lives. In the scenario depicted in Table II, Bob pressed his panic button and he is in a dangerous region to which, $John$ is not allowed to enter. Furthermore, Bob could be in a restricted zone or rescuing Bob might require $John$ to operate an unauthorized device. There is no room for hesitation under time constraints and evaluating these options can easily confuse $John$ due to environmental conditions, physical fatigue, and distress.

In the next section, we discuss how an intelligent policy framework could be utilized to minimize confusions by reducing the cognitive burden on end users of mining solutions.

III. Enhancing Heath and Safety in Underground Mines

The safety of the workers in an underground mine can be greatly enhanced by assessing their health status, measuring gas levels in the mine, tracking assets and preventing vehicle collisions. In this section, we present our use case domain, discuss the significance of regulations, and explain how our framework could be applied to in-use solution.

A. Smart Mine Solutions

In general, IoT solutions for underground mines have two main components; one for tracking RFID tags (assets, vehicles, personnel), and another one for measuring gas levels. The gas levels are measured approximately every 5 seconds and tags are sensed arbitrarily as assets or personnel come close to indoor readers. The sensor data is stored

	Prohibition	Obligation
$\chi : \rho$	$?p : Personnel(?p)$	$?p : Personnel(?p)$
N	P	O
α	$methane(?r, ?level) \wedge\ ?level > 1.0$	$panic(?p, ?urgent) \wedge\ inRegion(?p, ?r)\ inRegion(?p, ?r)$
$a : \varphi$	$?a : EnterRegion(?a) \wedge\ actor(?a, ?p) \wedge target(?a, ?r)$	$?a : Rescue(?a) \wedge\ actor(?a, ?p) \wedge target(?a, ?p)$
e	$SafeZone(?r)$	$inRegion(?p, ?safe) \wedge SafeZone(?safe)$
d		$15 : minutes$
c	10.0	30.0

Table III
EXAMPLE POLICY REPRESENTATIONS

1	Personnel should not use unauthorized tools.
2	Assets should not function outside their assigned regions.
3	Personnel should not enter unauthorized regions.
4	All personnel must go to a safe zone before blasting.
5	Personnel are obliged to help, if a person is in need.
6	Personnel must exit a zone if CO level is $\geq$ 1.0%
7	Firemen are obliged to respond to fire dangers in the mine.
8	Employer must take her personnel to safety.

Table IV
EXAMPLES FROM UNDERGROUND MINE REGULATIONS OF TURKEY.

in two relational database tables and the business logic (policies) is embedded in the software. These systems are able to perform geofencing, locate assets in the dark, detect dangerous gas levels and so forth. However, they do not offer adaptive solutions and they are not capable of making intelligent decisions in case of multiple conflicts. Although IoT devices carry a huge potential for improving mining operations, existing *smart* mine solutions are limited since they mostly operate on raw data with dangerous events and outcomes being hard-coded along with *predefined one-size* fits all remediations and the collected data is monitored by human in control rooms without any intelligent assistance.

B. Domain Policies

Since underground mines are extremely dangerous environments, worker safety is highly regulated. We use policies taken from the health and safety regulations of underground mines in Turkey as our motivating examples. These policies are depicted in Table IV. The language used in these regulations is too vague for people without expert knowledge to understand. Furthermore, some of these regulations are more important than others and, if critical circumstances arise that makes it impossible to satisfy all the constraints and it might become necessary to violate the less important regulations; for instance, a personnel can take initiative and cut the electricity in a dangerous location, if there is not enough time for an engineer to arrive.

C. Ontology Based Access

The proposed policy framework requires an OWL-QL database to function; this is due to two reasons: *(1)* we recognize the costs and difficulties in modifying a companies' existing systems, thus any semantic solution to the problem should have the least overhead, thus encoraging the adaptation; and *(2)* the policies are already stored in a database, thus any semantic language that provide easy interface to database based reasoning is preferred. For these reasons, we developed a software agent that uses an Ontology Based Access and Integration (OBDA/OBDI)[11] framework (i.e., Ontop [12]) to work with the target database. In short, OBDI allows us to define an ontology over a relational database by mapping [13] the concepts and properties of the ontology to the columns of tables in the target database using SQL queries—e.g., to retrieve individuals of *Blaster* class, we can map it to the following SQL query: *select id from tbl_personnel where role = 2*.

Ontop[6] is one of the most efficient OBDI implementations available and it is actively developed. In this paper, we do not evaluate the performance of Ontop, however we show that an OBDI approach could be used to implement a policy framework that can meet the performance demands of a large scale existing IoT system without performing any modifications to the target databases.

D. Automatic Conflict Resolution via Planning

In our recent work [14], we showed that it is possible to minimize the violation costs of policies or to completely avoid conflicts in a smart home environment using an automated planner. Moreover, we were able to embed inference rules of OWL-QL into a PDDL [15] domain using derived predicates and model policies with cost functions.

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 explosion risk prohibition in PDDL as shown by the code snippet in Figure 1.

We can model safety regulations as a planning problem instead of hard-coded rules and, by exploiting the technology,

[6]https://github.com/ontop/ontop/wiki/ObdalibQuestBenchmarks

```
(:durative-action go-to-region
  :parameters (?person ?from ?to)
  :duration (= ?duration (distance ?from ?to))
  :condition (and
    (at start (Personnel ?person))
    (at start (inRegion ?person ?from))
    (at start (not-inPanic ?person))
    (at start (connectedTo ?from ?to)))
  :effect (and
    (at start (not(inRegion ?person ?from)))
    (at end (inRegion ?person ?to))
    (at end
            (increase (total-cost)
                        (+ (explosionRisk ?to) 2)))))
```

Figure 1. Explosion risk prohibition in PDDL

offer adaptive solutions. By providing information in real-time, all the sensors and tags available in the mine make it possible for us to model the mine environment in a detailed and up-to-date manner. Planning can then be used in two different ways: *(1)* in under normal conditions, to organise the activities of the workers in the mine with the goal of maximising extraction and minimising the use of resources; and *(2)* in abnormal conditions, to deal with emergencies or anomalies. However, in both cases, the main goal of planner is to minimize the violation cost of policies. In some cases, a planner might decide not to fulfill an obligation if it violates too many prohibitions.

IV. Evaluation

In this section, we describe how we evaluated the performance of our policy framework by integrating it with a real system architecture.

A. Setup

We ran our experiments on a server with 32 GB RAM and two 8-core Intel Xeon CPU (E5-2650 0 - 2.00GHz). However, we obtained similar results using a Late 2011 MacBook Pro with 8GB RAM and one 4-core Intel i7 (2.4GHz), since we did not parallelize any of the computations. We mainly conducted our experiments on the server due to its larger main memory capacity.

We developed an underground mine ontology with the help of the faculty members of Istanbul Technical University's (ITU) mining engineering department. The ontology in total has 45 classes, 7 object properties, and 20 data properties. It is still in the development process, however it is sufficient to capture all the concepts and relationships in the mining company's database.

We were told that there are 1600 personnel, 185 gas sensors, and approximately 10 vehicles working at the same time. This specific company does not track its assets, however we assume that they are being tracked and there are 6400 assets (4 per personnel) in the underground facility to make the problem more challenging. Measurements of sensors are made every 5 seconds and tag readings are done arbitrarily. Based on this information, we filled in the database with randomly generated data. The above numbers belong to a fairly large mining facility. Thus, the starting point of the data set for our experiments, which represents smaller mines, constitutes of 200 personnel, 30 gas sensors, 30 vehicles and 800 assets. Then, at each iteration we increased the size of the data set by 100 personnel, 15 gas sensors, and 400 assets. Hence, we started with 1060 rows in total and increased it up to 83460 rows (160 iterations). We did not increase the number of vehicles, as they do not generate much data and the number of existing vehicles in the starting point is already larger than real numbers. We used the values in IJCRS 15 Data Challenge: Mining Data from Coal Mines [16] to generate values for gas levels.

We used an in-memory database H2 to be able to respond the changes in the real time. In-memory databases might be expensive and sometimes may not be suitable for storing all of the data of the mining company, however the important factor here is that we do not need the entire information inside the database—e.g. we only need the most recent sensor readings to check which policies are activated or expired or we do not need to store personnel addresses. The ontology is mapped to a database schema, which is simpler than the company's database and more suitable for Ontop mappings—i.e., it requires less SQL joins to represent concepts. Figure 2 depicts the implemented framework.

Finally, we developed a software agent in Java, which uses Ontop API and connects to the database of the company to update our in-memory database. It is important to note that only frequently changing data is the locations of personnel and assets, and sensor readings. Small changes in sensor readings might be useful for predicting the future value of gas levels or locations, however we do not need to update the sensor data so often to check policy activations. For efficiency reasons and to put less workload on the tables in the disk, we created two materialized views that capture the

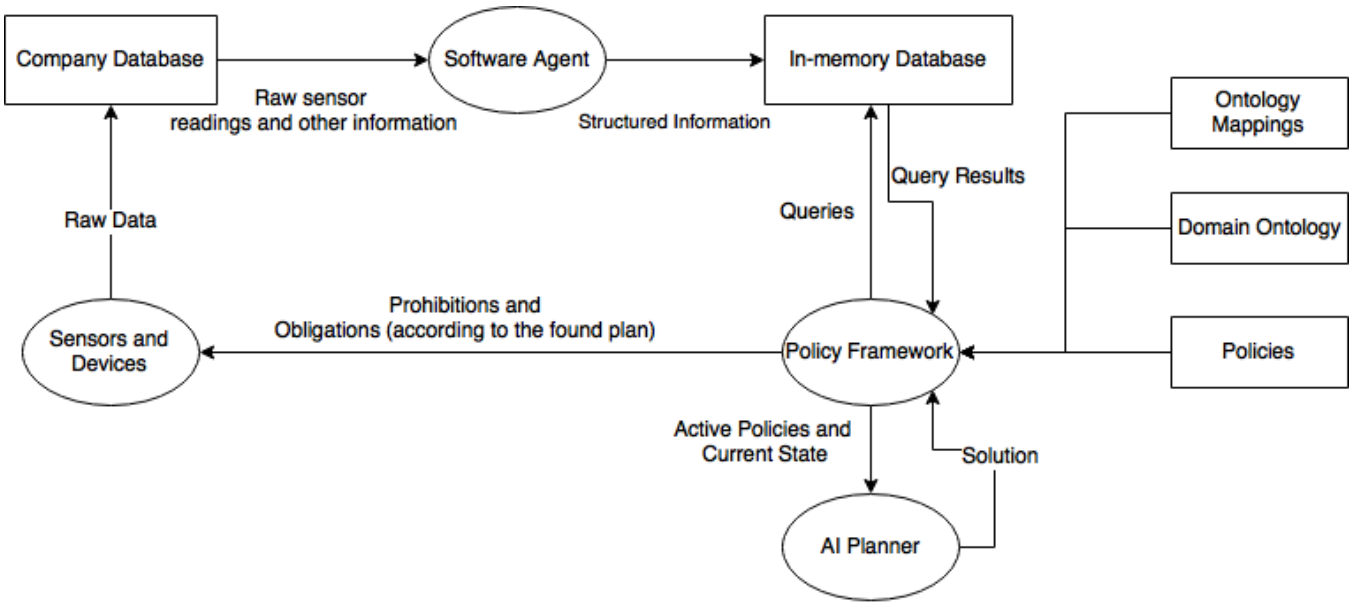

Figure 2. Overview of the integration of our framework

last locations and sensor readings. The agent simply queries these tables, when it needs to refresh the data in the memory.

B. Experiments

The main goals of our experiments are as follows: 1) to see how fast we can run activation/expiration queries and how many policy instances we can create before a new update in the database; and 2) and how these number changes with respect to the number of working personnel, assets, and sensors.

We executed the activation conditions as SPARQL[7] queries with Ontop reasoner over an H2 in-memory database. We measured two different times: the query execution time and policy instance creation time. The latter measures how fast we can create Java objects which represent active (with bindings) instances of a policy. In other words, we execute a query, and create an active instance for each returned row. It would not be useful just to show the query performance of Ontop as we still need to refine these activated policies to their addressees.

We evaluated the performance of our framework with the 8 policies presented in Table IV; each of the policies are given 3 warm-up runs and then 10 consecutive runs for evaluation. We discard the slowest run and take the average of the remaining 9 runs. For each policy, we timed how fast we created policy instances and computed the average. We repeated this experiment 10 times and calculated their mean and standard deviation.

1) Policy Activation Performance: Figure 4 illustrates the execution times of the activation conditions of our 8 different policies. The slowest run is about 1.3 miliseconds and the time does not increase as the data increases; hence the returned number of rows increases. Aside from the fact that Ontop being a very efficient framework, the main reason behind this is the fact that even the largest data set is manageable. The data stored on the memory is greatly reduced compared to the data stored on target system's disk, however the in-memory database still contains more information than the real system, since we increased the number of entities to test the system. The history of readings could be useful for doing prediction or anomaly detection, yet we only require the most recent readings and only a part of the database to see which policies are activated or expired. The isolation of the ontology data is a simple method to handle large volumes of arbitrary sensor data.

2) Policy Instance Creation Performance: The policy instance creation times for each policy are shown in Figure 5. It can be seen that deviation is higher and instance creation time varies for each policy. The deviations are caused by overheads in Java code, however instance creation time mainly depends on the expiration conditions of policies and how fast we can bind them. These run times could be improved by doing lazy binding. The slowest average creation time for an active instance belongs to the first policy and the required time is $\sim$3 milliseconds. The fastest time is less than a millisecond, since that policy does not have an expiration condition—i.e., always active.

C. Planning Performance

It is crucial to model this high-risk domain in PDDL by making as few assumptions as possible and evaluate the performance of planners on real-world problem instances.

[7]https://www.w3.org/TR/rdf-sparql-query/

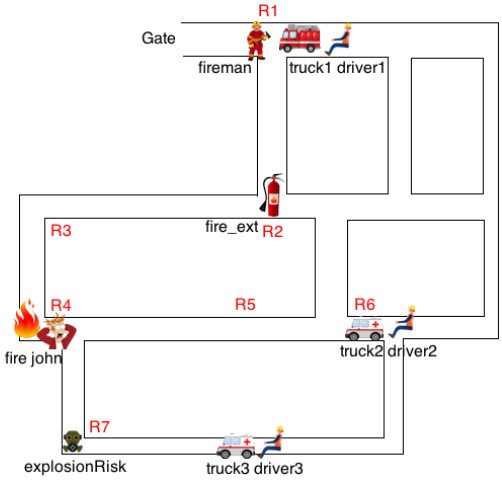

Figure 3. Initial state of the PDDL problem

This would allow us to test if planners can meet the performance demand posed by mining problems.

We modeled a simple domain and a problem in which a person is in panic at a fire zone as depicted by Figure 3. The initial state consists of firemen, transporter vehicles, drivers, gas levels, the map of the mine, and so on. We also put explosion risks to some of the regions to create policy conflicts such as some people are obliged to help the person in panic, but they are also prohibited to enter the regions with explosion risk. We used LAMA [17] to use derived predicates and OPTIC [18] to perform temporal planning. Although, both planners were able to provide a solution for this scenario, they fail to solve problems, which are only slightly more complex.

Step	Action
1)	*move-asset* truck1 driver1 fireman r1 r2
2)	*go-to-region* truck2 driver2 r6 r5
3)	*move-asset* truck driver1 fireman r2 r3
4)	*go-to-region* truck2 driver2 r5 r4
5)	*move-asset* truck2 driver2 john r4 r3
6)	*go-to-region* truck1 driver1 r3 r2
7)	*move-asset* truck1 driver1 fire_ext r2 r3
8)	*move-asset* truck2 driver2 john r3 r2
9)	*extinguish-fire* fireman fire_ext r3 r4 fire
10)	*move-asset* truck2 driver2 john r2 r1

Table V
RESCUE PLAN.

We provide a sample initial state and its plan in Table V. The plan is generated by Optic. In this scenario, we assume that it is an early stage of a fire and it is possible for a driver to pick up the person in panic without the help of firemen. However, without much effort we could also model the scenario, in which fire has to be extinguished before saving the person in panic.

The planner is intelligent enough to perform a division of tasks between two available drivers. The planner assigns *driver1* to locate *fireman* along with the fire extinguisher, and take them both close to the fire. Meanwhile, *driver2*'s task is to bring *john* to safety, while the ambulance, which is closer to *john* delegates its task to avoid the explosion zone.

V. Discussion and Conclusions

In order to increase the efficiencies, we can separate the querying and object creation processes; with two different threads running simultaneously, the system could execute activation and expiration queries while the activation that returns rows could create a new thread to create policy instances. We note that it takes between 0.2ms to 0.3ms to refresh the information in memory when there are more than 470k entries in sensor tables, so the process is efficient and applicable in highly dynamic situations. Since the mining company generates approximately 200k entries in a day from 185 gas sensors, this is applicable in real-time for the mining problem we have highlighted in this document.

Based on our experimentation, we observed that we can execute $\sim$770 queries in a second. Considering the gas levels are measured every 5 seconds and the database is only updated when their value change, at the worst case, we could run 3850 queries which may be expiration conditions of active instances or activation conditions of policies before the next update in the database. Furthermore, we can create $\sim$300 instances of a policy in a second, which allows us to refine a policy to 1500 assets and personnel in 5 seconds. This number approximately corresponds to all personnel in the mining facility. It is also important to note that the most of the policies will only be refined to a small part of the assets and personnel.

The results show that our framework can efficiently respond in real-time to the requirements of a mining system, which tracks about 1600 active personnel, 64800 assets and 2430 different sensors. In terms of sensor readings, tracked assets and personnel, we ran our experiments on much larger data sets than the once generated by the existing systems; our framework can support hundreds of policies when compared to existing systems and increases in performance directly proportional to the increase in data. The numbers can fluctuate or differ according to the host environment, however our framework fulfills its efficient and scalable policy reasoning promise required by an IoT systems. On the other hand, there is room for improvement in the automatic conflict resolution strategy before being used in real-world applications.

Further performance evaluations could be conducted to investigate how efficiently the system can determine the affected actions and addressees be notified. The planner performance should also be evaluated to determine if the system can resolve conflicts in real-time or find optimum solutions in generating plans to fulfill obligations.

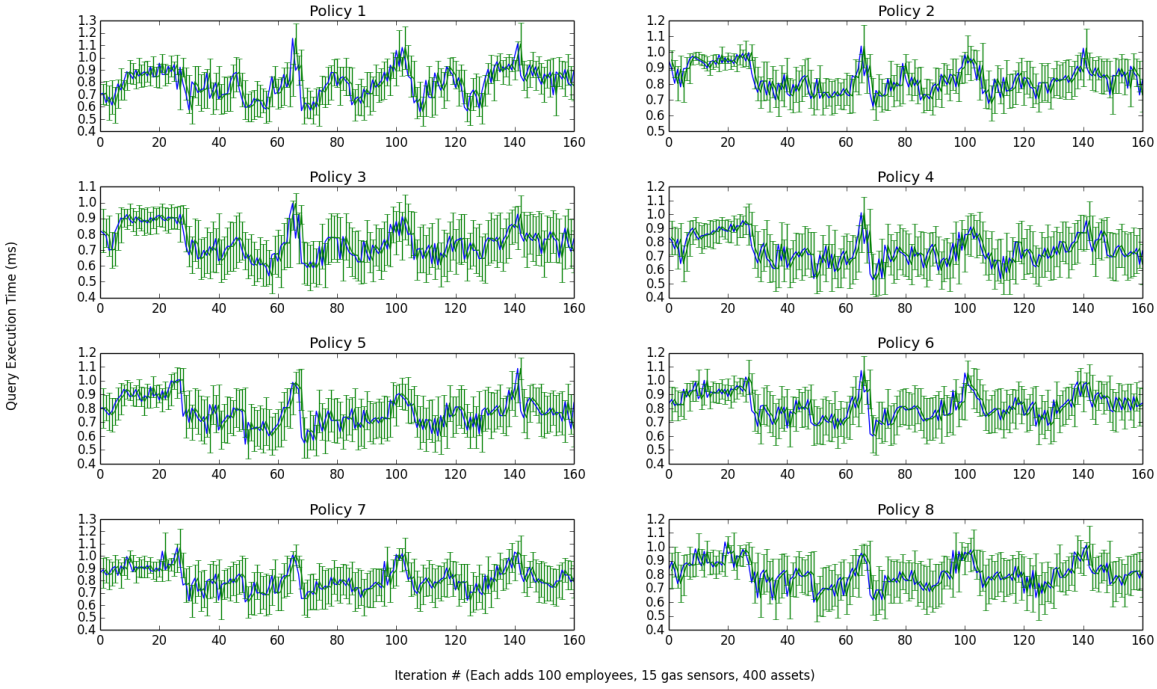

Figure 4. Query execution times of policies w.r.t. increasing dataset

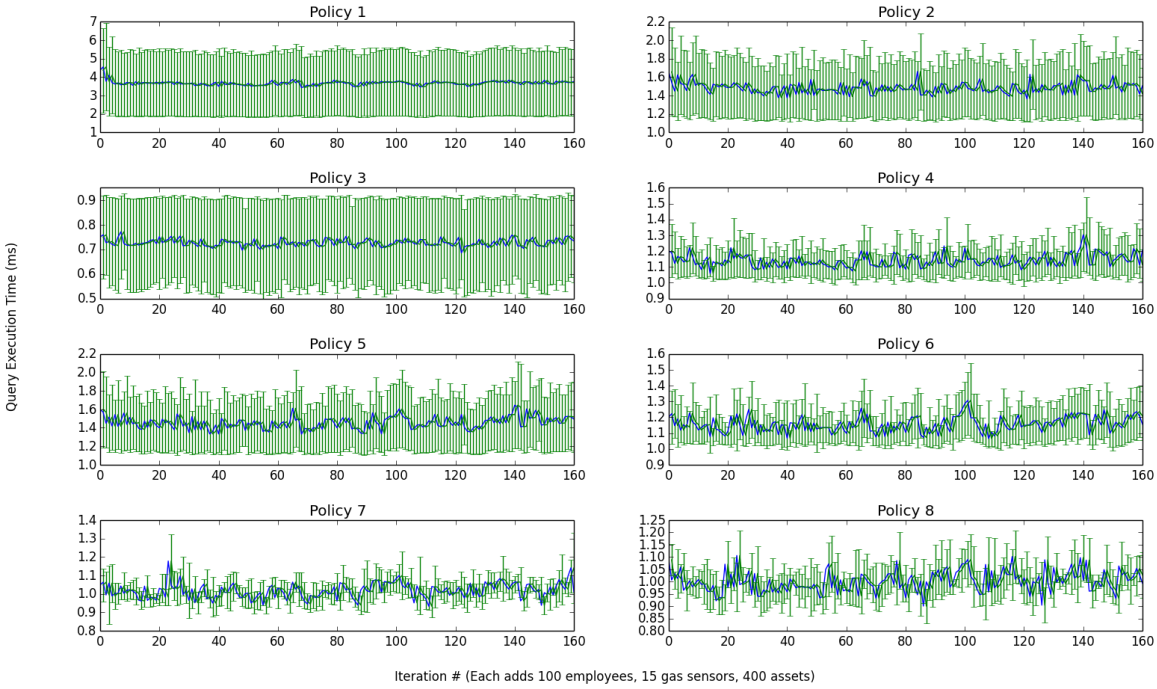

Figure 5. Policy creation times w.r.t. increasing dataset

ACKNOWLEDGMENT

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

Dr. Sensoy also thanks the Scientific and Technological Research Council of Turkey (TUBITAK) for its support under grant 113E238.

REFERENCES

[1] W. W. Vasconcelos, M. J. Kollingbaum, and T. J. Norman, "Normative conflict resolution in multi-agent systems." *Autonomous Agents and Multi-Agent Systems*, vol. 19, no. 2, pp. 124–152, 2009. [Online]. Available: http://dblp.uni-trier.de/db/journals/aamas/aamas19.html#VasconcelosKN09

[2] M. Sensoy, T. Norman, W. Vasconcelos, and K. Sycara, "Owl-polar: A framework for semantic policy representation and reasoning," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 12, no. 0, 2012. [Online]. Available: http://www.websemanticsjournal.org/index.php/ps/article/view/232

[3] A. Uszok, J. M. Bradshaw, J. Lott, M. R. Breedy, L. Bunch, P. J. Feltovich, M. Johnson, and H. Jung, "New developments in ontology-based policy management: Increasing the practicality and comprehensiveness of kaos," in *9th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2008), 2-4 June 2008, Palisades, New York, USA*, 2008, pp. 145–152. [Online]. Available: http://dx.doi.org/10.1109/POLICY.2008.47

[4] L. Kagal, T. Finin, and A. Joshi, "A Policy Language for A Pervasive Computing Environment," in *IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, June 2003.

[5] K. P. Twidle, N. Dulay, E. Lupu, and M. Sloman, "Ponder2: A policy system for autonomous pervasive environments." in *ICAS*, R. Calinescu, F. Liberal, M. Marín, L. P. Herrero, C. Turro, and M. Popescu, Eds. IEEE Computer Society, 2009, pp. 330–335. [Online]. Available: http://dblp.uni-trier.de/db/conf/icas/icas2009.html#TwidleDLS09

[6] Y. Zhu, S. L. Keoh, M. Sloman, and E. C. Lupu, "A lightweight policy system for body sensor networks," *IEEE Trans. on Netw. and Serv. Manag.*, vol. 6, no. 3, pp. 137–148, Sep. 2009. [Online]. Available: http://dx.doi.org/10.1109/TNSM.2009.03.090301

[7] N. Qwasmi and R. Liscano, "Distributed policy based management for wireless sensor networks to support the internet of things environment," in *Proceedings of 24th Annual International Conference on Computer Science and Software Engineering*, ser. CASCON '14. Riverton, NJ, USA: IBM Corp., 2014, pp. 335–338. [Online]. Available: http://dl.acm.org/citation.cfm?id=2735522.2735577

[8] E. Goynugur, G. D. Mel, M. Sensoy, K. Talamadupula, and S. Calo, "A knowledge driven policy framework for internet of things," in *Proceedings of the 9th International Conference on Agents and Artificial Intelligence - Volume 2: ICAART,*, 2017, pp. 207–216.

[9] R. Fikes, P. Hayes, and I. Horrocks, "Owl-ql?a language for deductive query answering on the semantic web," *Web semantics: Science, services and agents on the World Wide Web*, vol. 2, no. 1, pp. 19–29, 2004.

[10] D. Calvanese, G. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati, "Tractable reasoning and efficient query answering in description logics: The dl-lite family," *J. Autom. Reason.*, vol. 39, no. 3, pp. 385–429, 2007.

[11] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, and R. Rosati, "Ontologies and databases: The *DL-Lite* approach," in *Semantic Technologies for Informations Systems – 5th Int. Reasoning Web Summer School (RW 2009)*, ser. Lecture Notes in Computer Science, S. Tessaris and E. Franconi, Eds. Springer, 2009, vol. 5689, pp. 255–356.

[12] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro, and G. Xiao, "Ontop: Answering SPARQL queries over relational databases," *Semantic Web*, vol. 8, no. 3, pp. 471–487, 2017. [Online]. Available: http://dx.doi.org/10.3233/SW-160217

[13] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati, "Journal on data semantics x," S. Spaccapietra, Ed. Berlin, Heidelberg: Springer-Verlag, 2008, ch. Linking Data to Ontologies, pp. 133–173.

[14] E. Goynugur, S. Bernardini, K. Talamadupula, G. de Mel, and M. Sensoy, "Policy Conflict Resolution in IoT via Planning," in *Proceedings of the Twenty Thirtieth Canadian Conference on Artificial Intelligence*, 2017.

[15] M. Fox and D. Long, "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains," *Journal of Artificial Intelligence Research*, vol. 20, 2003.

[16] A. Janusz, M. Sikora, L. Wrobel, S. Stawicki, M. Grzegorowski, P. Wojtas, and D. Slkezak, "Mining Data from Coal Mines: IJCRS15 Data Challenge," in *Proceedings of RSFDGrC 2015*, ser. LNAI, vol. 9437. Springer, 2015, pp. 429–438.

[17] S. Richter and M. Westphal, "The lama planner: Guiding cost-based anytime planning with landmarks," *J. Artif. Int. Res.*, vol. 39, no. 1, pp. 127–177, Sep. 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1946417.1946420

[18] J. Benton, A. Coles, and A. Coles, "Temporal Planning with Preferences and Time-Dependent Continuous Costs," in *Proceedings of the Twenty Second International Conference on Automated Planning and Scheduling (ICAPS-12)*, 2012.