Semantic Inferences Towards Smart IoT-Based Systems Actuation Conflicts Management

Gérald Rocher(✉), Jean-Yves Tigli, and Stéphane Lavirotte

I3S Laboratory, Université Côte d'Azur, CNRS, Sophia-Antipolis, France
{gerald.rocher,Jean-yves.tigli,
Stephane.lavirotte}@univ-cotedazur.fr

Abstract. IoT-based systems have long been limited to collecting field information via sensors distributed at the edge of their infrastructure. However, in many areas such as smart home, smart factory, etc. these systems include devices that interact with the physical environment via common actuators. Throughout the lifecycle of these systems, from design, to deployment to operation, the ability to avoid actuation conflicts, both in terms of the commands that actuators receive (direct conflicts) and the effects that they produce (indirect conflicts), is a new challenge in the realm of trustworthy Smart IoT-based Systems (SIS). As part of the European project ENACT, which aims to provide full DevOps support for trustworthy SIS, we present a lightweight ontology that provides SIS designers with (1) a semantic metamodel to formally describe SIS subsystems and the actuators they interact with, and (2) a set of SWRL (Semantic Web Rule Language) inference rules to automatically identify and semi-automatically resolve actuation conflicts. Consistent with the best practices of the DevOps approach, a particular emphasis is placed on facilitating the use and interpretation of inference results. To provide insight into the appropriateness of the proposed approach in the context of SIS, rule processing times for different actuation conflict configurations are provided.

Keywords: Actuation · Conflict · Identification · Resolution · Ontology · Internet of Things · DevOps

1 Introduction

DevOps is one of the best practices in software engineering today [1]. The method aims to harmonize software development (Dev) and software operations (Ops) in a collaborative framework. It facilitates all phases of a system lifecycle, from design, development, integration, testing and deployment, to runtime monitoring and behavioural analysis, with the latter phases introducing a new design phase into a perpetual, incremental and agile development cycle. If DevOps is obvious today, it is thanks to a number of technical enablers, such as infrastructures that enable and facilitate the deployment of systems designed on the basis of *convergence* and *virtualization* hypotheses. A backend is deployed in a cloud while the frontend relies on web interfaces supported by similar

L. M. Camarinha-Matos et al. (Eds.): IFIPIoT 2022, IFIP AICT 665, pp. 255–273, 2022.
https://doi.org/10.1007/978-3-031-18872-5_15

target devices such as smartphones, PCs, tablets, SmartTVs, interactive terminals etc. at the price of some web responsive design configurations.

However, in the context of the Internet of Things (IoT), these hypotheses are undermined by the strong heterogeneity and dynamics of the infrastructure. This heterogeneity can be divided into two types. The first type of heterogeneity is that of computational targets, embodied in a three-tier infrastructure vision: cloud but also edge and IoT devices. Consideration of computational and storage constraints along with the locality of edge and IoT device targets is critical here. The second type of heterogeneity concerns the wide variety of IoT devices, which by their nature cannot benefit from the well-known convergence phenomenon in the evolution of IT media. This new characteristic is intrinsically linked to their vocation and results from the natural evolution of digital systems, which has been observed from the 90s onwards: "*Silicon-based information technology [...] is far from having become part of the environment*" (Mark Weiser). While Pervasive Computing and Ambient Intelligence (AmI) have reinforced the idea that modern computing is not only confronted with the distribution, the availability (*everytime*) and the mobility of their supports (everywhere), it must now be recognized that modern computing is also confronted with the great diversity of IoT devices and their variability in terms of *sensors* and *actuators* (everything).

While many IoT-based systems focus on massively collecting field data from sensors, their scope becomes increasingly complex once they are able to act on the physical environment using actuators shared by multiple independent subsystems. The management of such devices becomes critical, as their sharing or simultaneous use potentially leads to the occurrence of conflicts that can result in user dissatisfaction at best and dramatic consequences in the field at worst. From a design standpoint, managing actuation conflicts is made difficult by the complexity of so-called Smart IoT-based Systems (SIS) and the large number of shared actuators they may rely on at the edge of the infrastructure. As we move towards trustworthy SIS, it is imperative to provide DevOps stakeholders with tools that can support both the identification of actuation conflicts and their resolution through the instantiation of Actuation Conflicts Managers (ACM) *at relevant conflict points in the design.* The local nature of ACM here suggests the possibility of their reuse which is relevant in the context of DevOps best practices because it enables continuous and rapid deployment.

Given this context, this paper makes a threefold contribution:

1. We present a lightweight ontology that provides DevOps stakeholders with a *semantic metamodel* for formally describing Smart IoT-based Systems (SIS) and the actuators with which they interact. The formal description is automatically obtained from the deployment and implementation models provided as part of the DevOps framework,
2. The identification and resolution of actuation conflicts are *automatically* and *systematically* derived from SWRL rules (Semantic Web Rule Language) used in conjunction with a Description Logic (DL) reasoner. Special attention is paid to facilitating the use and interpretation of inference results:

    a. Detected conflicts are clarified by special instances, and querying the knowledge base after inference is not required,

   b. ACM components are automatically instantiated at relevant conflict points in the design. They make their inputs and outputs explicit for further use. As such, ACM components are black boxes whose associated resolution logic must be selected by designers from off-the-shelf reusable solutions or designed as needed,
   c. Specific object properties are derived to help designers understand the reasons for identified conflicts.

3. Performance metrics for different actuation conflict configurations are provided. They provide insight into the relevance of the proposed approach to the targeted SIS, which can range from a few dozen (smart-home) to tens of thousands of actuators (smart-city).

The paper is organized as follows. In Sect. 2, we discuss direct and indirect actuation conflicts and review some relevant work that uses semantic web languages to identify and resolve them. In Sect. 3, we describe a lightweight ontology and 6 SWRL rules for automatic identification and semi-automatic resolution of actuation conflicts. In Sect. 4, we use the Stanford Protégé tool to illustrate the proposed ontology and associated SWRL rules with a small example. In Sect. 5, we present performance results for different actuation conflict configurations. In Sect. 6, we discuss future work.

2 Related Works

Potential actuation conflicts are likely to occur whenever independent subsystems compete for access to common actuators (*direct actuation conflicts*) or common physical properties through different actuators (*indirect actuation conflicts*). There is a rich literature and culture on *feature interaction* in telecommunication systems, in software systems and, more recently, in IoT-based systems [2].

However, as highlighted in [2], the proposed methods mainly consider the identification and resolution of direct conflicts. Indirect actuation conflicts can be subtle, making them difficult to detect. For example, a ventilation system *indirectly* affects physical properties such as temperature and humidity by influencing airflow. A TV, understood primarily as an entertainment device, can also be understood as an actuator that affects sound, brightness, and, to a lesser extent, temperature. Indirect actuation conflicts involve non-trivial semantic and subjective considerations and are therefore difficult to resolve automatically while still satisfying all SIS end users. For example, if one user wants to increase the temperature in a room while another wants to increase the airflow in the same room by opening a window and then possibly lower the temperature, what must be the resolution strategy that satisfies both? In this context, the use of semantic web formal description languages [4] and their reasoning capabilities seems to be a relevant approach for describing SIS, identifying and resolving their actuation conflicts.

Some research has been done in this direction recently. In [5], the authors propose a generic knowledge graph to represent the relations between IoT services and environment entities. The indirect actuation conflicts are then identified based on Event-Condition-Action (ECA) automation rules defined by end-users. No resolution is proposed in this work. In [6] the authors present A3ID, an automatic indirect actuation conflicts detection method based on IF-This-Then-That (IFTTT) rules and knowledge graphs that capture the functionality, effect and scope of the devices involved in the design. No resolution method is proposed in this work. In [7], the authors consider the case where different end-users interact with a Building Automation System (BAS). End-user requirements are encoded by an ontology model that provides semantic information about the physical environment. Identification of indirect conflicts is achieved by SPARQL queries [3] to this model, while resolution operations are performed using constraint solving.

Most of these approaches are based on knowledge of the functional logic of the systems under consideration, end-user requirements, rules and policies. SIS, as defined in this paper, may be large-scale systems (e.g., smart city) built on highly dynamic subsystems (e.g., cloud services, containerized microservices, embedded software, heterogenous edge devices, etc.) whose functional logic (hardware and software) is not necessarily under the control of the DevOps stakeholders. The knowledge is therefore limited to the structural interactions between subsystems and the actuators they act upon at the edge of the infrastructure provided by deployment and implementation models as part of the DevOps approach (e.g., [8, 9]). By focusing on the structural interactions, the identification of potential actuation conflicts can be done systematically, and their resolution applied locally through reusable ACM that implement different resolution strategies, in line with DevOps best practices.

Finally, none of the above approaches provides processing time data for managing actuation conflicts. For example, in [6], although the authors conducted experiments with 11,859 IFTTT-like rules with up to 99 actuators, no performance data is provided.

3 A Lightweight Ontology for Identifying and Resolving Actuation Conflicts

The Semantic Web can be defined as "*a vision for the future of the Web in which information is given explicit meaning, making it easier for machines to automatically process and integrate information available on the Web*" [10]. In this context, explicit meaning is provided by semantically rich metadata that relies on *ontologies*. An ontology (a.k.a. vocabulary) is a meta-model that defines concepts and relationships used to describe and represent a particular domain[1]. It is based on logic-based knowledge representation languages such as RDF (Resource Description Framework), RDFS (RDF-Schema) [11] and OWL-* family of languages [12] (OWL-LITE, OWL-DL and OWL-FULL), each providing different levels of expressiveness for asserting facts or axioms.

What makes ontologies interesting is their capacity, from their underlying logic-based knowledge representation languages, to derive logical consequences (i.e., implicit assertions) from a set of asserted facts or axioms (i.e., explicit assertions). However, the derivational capacity is limited by the expressive capacity of the language in question. The greater the expressive capacity of the language, the lower the inference and the computability [13]. For example, OWL-LITE and OWL-DL are decidable and correspond to $\mathcal{SHIF}$ and $\mathcal{SHOIN}$ Description Logics (DL) respectively, with $\mathcal{SHOIN}$ DL providing higher expressivity than $\mathcal{SHIF}$ DL. OWL-FULL, provides the highest expressivity, but is not decidable.

[1] https://www.w3.org/standards/semanticweb/ontology.

In the context of SIS and DevOps, the use of semantic web technologies seems to be relevant assuming:

1. Explicit statements describing (1) the structural relationships between SIS subsystems and the actuators they interact with, (2) the actuators (at least, their localization and the physical properties they act upon), can be extracted from DevOps deployment and implementation models,
2. The knowledge representation language is expressive enough to identify direct and indirect actuation conflicts and guide DevOps stakeholders towards their resolution from inference under constraint of decidability.

Based on these assumptions, a lightweight OWL-DL ontology is presented below for automatically identifying and semi-automatically resolving actuation conflicts.

It includes the following concepts and relationships whose individuals are taken from DevOps deployment and implementation models, shown in green in Fig. 1:

`Entity` - An entity is an abstract element,

`Subsystem` - A subsystem is an entity that sends commands to an entity,

`Physical Property` - A physical property is any observable and measurable property whose value characterizes a state of a physical system [14] (e.g. temperature, brightness, humidity, pressure, sound, etc.),

`Context` - A context can be any abstract, spatially bounded physical system (e.g. kitchen, living room, etc.) whose state can be characterized by physical properties,

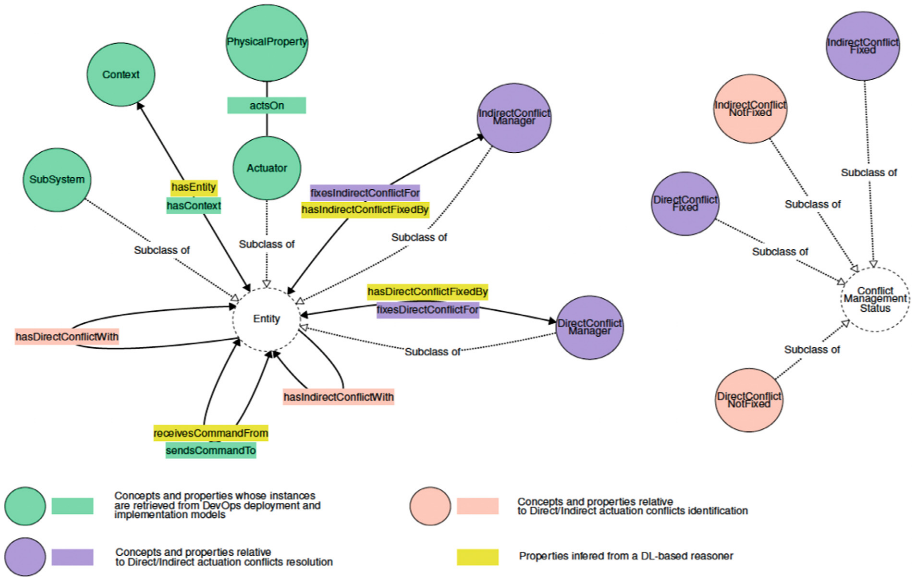


Fig. 1. Concepts and object properties of the proposed lightweight ontology.

`Actuator` - An actuator is an entity that has exactly one context, acts on (i.e. changes) at least one physical property and receives commands from at least one subsystem.

On this basis, an example of a knowledge description is given below, the graphical representation of which is shown in Fig. 2 is given below:

```
<!-- An actuator -->
<rdf:Description rdf:about="#TV">
  <hasContext rdf:resource="#Livingroom"/>
  <actsOn rdf:resource="#Luminosity"/>
  <actsOn rdf:resource="#Sound"/>
</rdf:Description>
<!-- A SubSystem -->
<rdf:Description rdf:about="#RemoteControl">
  <sendsCommandTo rdf:resource="#TV"/>
</rdf:Description>
<owl:AllDifferent>
<owl:distinctMembers rdf:parseType="Collection">
  <rdf:Description rdf:about="#TV"/>
  <rdf:Description rdf:about="#RemoteControl"/>
</owl:distinctMembers>
</owl:AllDifferent>
```

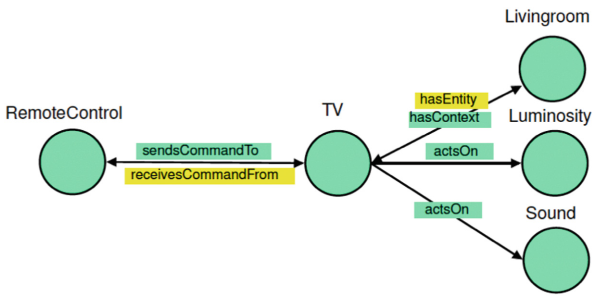


Fig. 2. Example of semantic description that can be expressed from DevOps deployment and implementation models (properties depicted in yellow are inferred from a DL-based reasoner).

This structured representation of knowledge provides a formal description of SIS and a basis for identifying and resolving direct and indirect actuation conflicts from inferences.

3.1 Automatic Actuation Conflicts Identification

The proposed ontology is equipped with the following concepts and properties related to the identification of direct and indirect conflicts (shown in pink in Fig. 1):

`DirectConflictNotFixed` - The individuals of this concept correspond to all actuators that are potentially subject to a direct actuation conflict.

→An actuator is potentially subject to a direct conflict if it receives its commands from at least two different entities,

`IndirectConflictNotFixed` - The individuals of this concept correspond to all actuators that are potentially subject to an indirect actuation conflict.

→An actuator is potentially subject to an indirect conflict if it shares its context with at least one other actuator acting on the same physical property.

While OWL-DL ontologies provide simple, reusable, and easy-to-understand models of domain knowledge, they lack the declarative expressiveness that rules provide, especially when it comes to designing complex assertions of facts that go beyond the simple declaration of domain concepts, as is the case with the above concepts [15]. The Semantic Web Rule Language (SWRL) [16] enables declarative assertions using OWL concepts. By combining first-order Horn logic (HL) and DL-based reasoners such as Pellet, Fact++, etc., it achieves higher expressive power and reasoning capacity. In this paper, the proposed ontology is SWRL-enabled, i.e. it contains a set of Horn clause rule axioms (Table 1 and Table 2) that conform to DL-*Safety* (i.e. rule axioms contain only known concepts, which makes them decidable [17]).

Table 1. Horn-clause axioms for Direct/Indirect actuation conflicts identification

1	An actuator **`?act`** is subject to a direct conflict if it receives its commands from at least two different entities: `Actuator(?act) ^ sendsCommandTo(?ent1,?act) ^ receivesCommandFrom(?act,?ent2) ^ differentFrom(?ent1,?ent2) → hasDirectConflictWith(?act,?ent1)`
2	An actuator **`?act1`** is subject to an indirect conflict if it shares its context with at least one other actuator **`?act2`** acting on the same physical property: `hasContext(?act1,?ctx) ^ receivesCommandFrom(?act1,?ent1) ^ hasEntity(?ctx,?act2) ^ receivesCommandFrom(?act2,?ent2) ^ actsOn(?act1,?eff) ^ actsOn(?act2,?eff) ^ differentFrom(?act1,?act2) ^ differentFrom(?ent1,?ent2) → hasIndirectConflictWith(?act1,?act2)`

The identification of actuation conflicts is then done in two steps:

1. The first step consists in asserting the object properties **`hasDirectConflictWith`** and **`hasIndirectConflictWith`** to each actuator that is potentially subject to direct and/or indirect conflicts. This step is achieved thanks to the axioms of the Horn-clause rule defined in Table 1,
2. The second step relies on a DL-based reasoner, i.e. **`DirectConflictNotFixed`** individuals are derived from actuator individuals that have the **`hasDirectConflictWith`** property. The same is true for **`IndirectConflictNotFixed`** individuals.

An example is shown in Fig. 3 where the TV instance has direct conflict with **`RemoteControl#1`** and **`RemoteControl#2`** instances. So far, no ACM has been instantiated to fix this conflict (**`DirectConflictNotFixed`**).

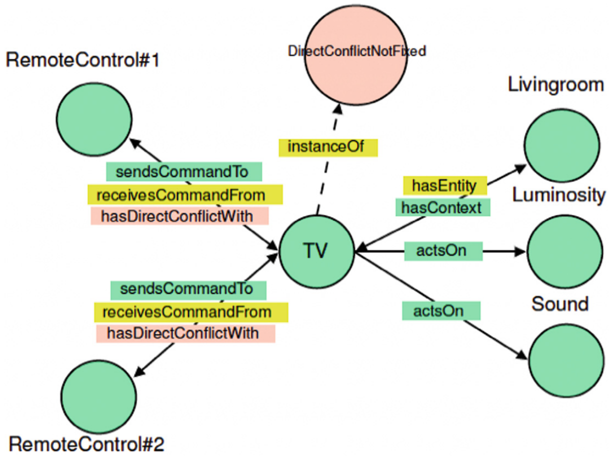


Fig. 3. Example of direct actuation conflicts identification

3.2 Semi-automatic Actuation Conflicts Resolution

Based on the assertions derived during the actuation conflicts identification phase, a means should be proposed to resolve the conflicts identified semi-automatically. For this purpose, the proposed ontology is equipped with additional concepts and properties, shown in purple in Fig. 1. Besides these concepts and properties, four additional Horn-clause rule axioms are defined in the ontology (cf. Table 2). In particular, rule 3 and rule 4 are used to instantiate individuals of the concepts **`DirectConflict-Manager`** and **`IndirectConflictManager`** at relevant points in the design. The instantiation of these individuals is done automatically thanks to the built-in SWR-LAPI extension `swrlx:makeOWLThing`, which can be used to create new individuals directly from a rule, where a **`DirectConflictManager`** is instantiated for each actuator that have the property **`hasDirectConflictWith`** asserted (rule 3); an **`IndirectConflictManager`** is instantiated for each actuator that have the property **`hasIndirectConflictWith`** asserted (rule 4).

Table 2. Horn-clause axioms for Direct/Indirect actuation conflicts resolution

3 Whenever an actuator **?act** has the property **hasDirectConflictWith(?act,?ent)** asserted, a direct ACM ?acm is instantiated with the property **fixesDirectConflictFor(?acm,?act)** asserted. The entity **?ent** sending commands to actuator **?act**, whose direct actuation conflict is fixed by the direct conflict manager **?acm**, also send commands to **?acm**:

```
Actuator(?act) ^ hasDirectConflictWith(?act,?ent) ^
swrlx:makeOWLThing(?acm,?act) → DirectConflictManager(?acm) ^
fixesDirectConflictFor(?acm,?act) ^ sendsCommandTo(?ent,?acm)
```

4 Whenever an actuator ?act1 has the property **hasIndirectConflictWith(?act1,?act2)** asserted, an indirect ACM **?acm**, associated to the pair PhysicalEffect/Context, is instantiated with the property **fixesIndirectConflictFor(?acm,?act1)** asserted:

```
PhysicalProperty(?eff1) ^ Context(?ctx) ^
hasEntity(?ctx,?act1) ^ Actuator(?act1) ^ actsOn(?act1,?eff1)
^ hasIndirectConflictWith(?act1,?act2) ^ actsOn(?act2,?eff2) ^
sameAs(?eff1,?eff2) ^ swrlx:makeOWLThing(?acm,?eff1,?ctx) →
IndirectConflictManager(?acm) ^ fixesIndirectConflict-
For(?acm,?act1)
```

5 Each entity **?ent** sending commands to actuator **?act**, whose indirect actuation conflict is fixed by an indirect conflict manager **?acm**, also send commands to **?acm**:

```
fixesIndirectConflictFor(?acm, ?act) ^ receivesCom-
mandFrom(?act, ?ent) → sendsCommandTo(?ent,?acm)
```

6 Direct/Indirect conflict managers are merged whenever they fix actuation conflict for the same actuators:

```
fixesDirectConflictFor(?acm1, ?act) ^ fixesIndirectConflict-
For(?acm2, ?act) → sameAs(?acm1, ?acm2)
```

In rule 3, **DirectConflictManager** individuals are created once per actuator (`swrlx:makeOWLThing(?acm,?act)`) while in rule 4, **IndirectConflictManager** individuals are created once for each pair (physical effect, context) (`swrlx:makeOWLThing(?acm,?eff,?ctx)`). This prevents ACM from being duplicated. As shown in Fig. 4, each instance of a direct/indirect ACM is bound to subsystems that send commands to the faulty actuators and to the faulty actuators themselves by asserting the **fixesDirectConflictFor** or **fixesIndirectConflictFor** properties depending on whether the ACM in question targets a direct or an indirect actuation conflict (rules 3, 4 and 5). In line with DevOps best practices, this approach enables the *systematic* implementation of *local* and *reusable* ACM whose integration into the deployment and implementation models can be greatly facilitated thanks to their associated properties.

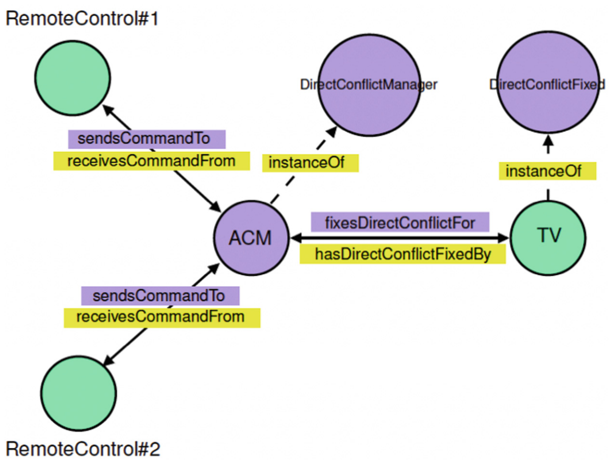


Fig. 4. Example of direct actuation conflicts resolution.

One special case must still be considered in order for the proposed actuation conflict resolution to be complete. This is the case when an actuator is subject to both direct and indirect conflicts. An example can be found in Fig. 5 where the TV receives its commands from two different entities and both TV and Lamp have the same context Livingroom and act on the same physical property (Luminosity).

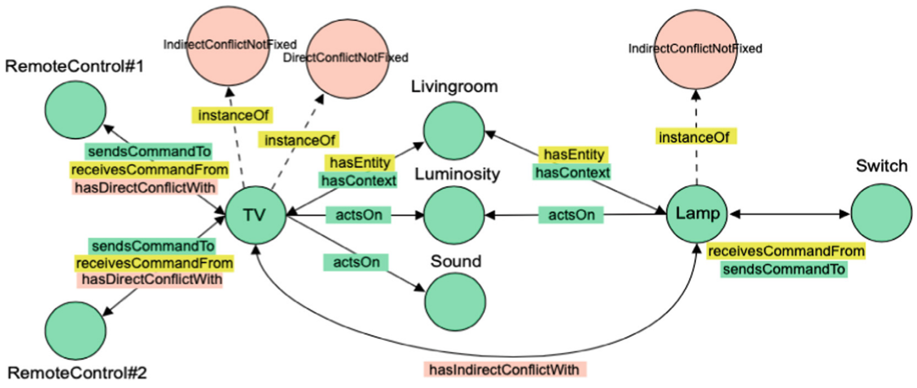


Fig. 5. Example of direct/indirect actuation conflicts identification.

In such a configuration, two ACM individuals must be created, a **`DirectConflictManager`** and an **`IndirectConflictManager`**, as shown in Fig. 6. Here, both individuals must be *merged* to prevent an indirect actuation conflicts from occurring between them. The solution to this is to consider both individuals as identical. This is achieved by the rule 6 defined in Table 2 and the use of the property `owl:sameAs`.

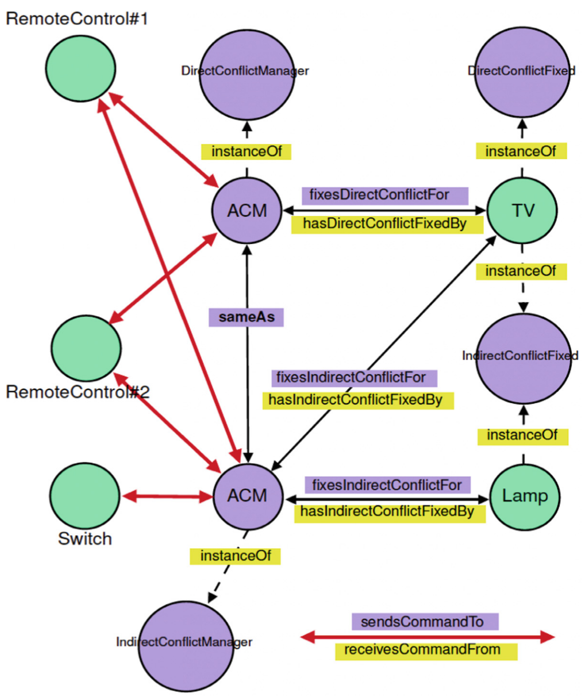


Fig. 6. Example of direct/indirect actuation conflicts resolution.

As such, actuation conflict managers are not yet associated with conflict resolution logic; they are *black boxes*. Concrete actuation conflict managers must be selected by designers from off-the-shelf reusable solutions, hence the semi-automatic qualification of the proposed resolution approach. We will have the opportunity to return to this point in the perspectives of this research. Finally, the counterparts of the concepts **`DirectConflictNotFixed`** and **`IndirectConflictNotFixed`**, are provided in the proposed ontology:

`DirectConflictFixed` - Individuals of this concept correspond to all actuators that are potentially subject to a direct actuation conflict and for which the property **`hasDirectConflictFixedBy`** is asserted, derived from `<owl:inverseOf rdf:resource="fixesDirectConflictFor"/>`,

`IndirectConflictFixed` - Individuals of this concept correspond to all actuators that are potentially subject to an indirect actuation conflict and for which the property **`hasIndirectConflictFixedBy`** is asserted, derived from `<owl:inverseOf rdf:resource= "fixesIndirectConflictFor"/>`.

4 Identifying and Resolving Actuation Conflicts with the Stanford Protégé Tool

This work is part of the DevOps approach, which aims, among other things, to enable continuous and fast software deployment thanks to a set of tools and models shared by all actors involved in the process. In this context, based on the semantic model described previously, we propose the use of the Stanford Protégé [18] tool to reason about the knowledge and analyze the results. Protégé is a free, open-source platform that provides a set of tools for building domain models and knowledge-based applications with ontologies[2]. The Protégé SWRLTab supports the execution of SWRL Horn-clause rules using the Drools rule engine [19] in conjunction with Fact++ DL reasoner [20]. As an example, consider a SIS with five indirect actuation conflicts, as shown in Fig. 7 below.

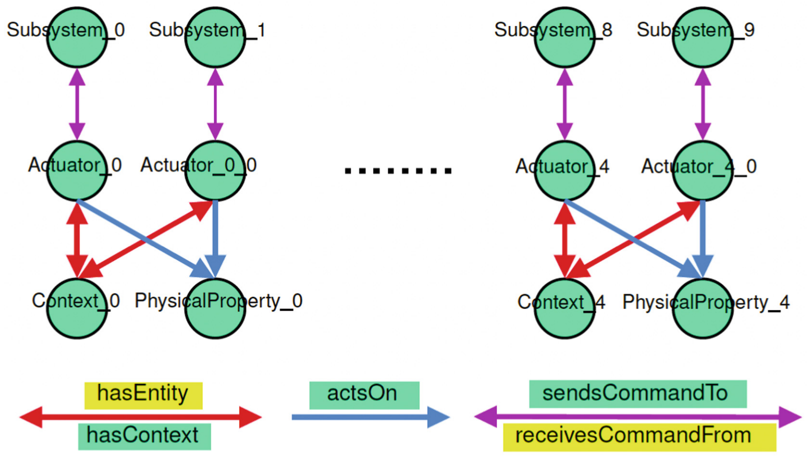
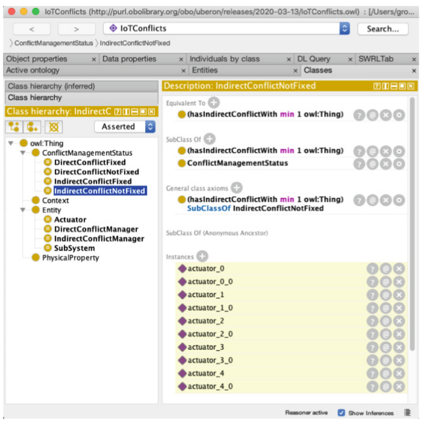


Fig. 7. SIS with five indirect actuation conflicts.

Initially, only the rules for identifying actuation conflicts identification are enabled (rule 1 and rule 2 in Table 1). Executing these rules in conjunction with the Fact++ reasoner produces the results shown in Fig. 8 and Fig. 9. Actuators that are subject to indirect actuation conflicts are identified as individuals of the concept **`Indirect-ConflictNotFixed`** (Fig. 8). Object properties associated with each actuator give designers the opportunity to better understand the cause of the conflicts (Fig. 9).

Now, the rules for resolving actuation conflicts are activated (rule 3, rule 4, rule 5 and rule 6 defined in Table 2). Executing these rules in conjunction with the Fact++ reasoner produces the results shown in Fig. 10. Actuator individuals that are subject to an indirect actuation conflict are now individuals of the **`IndirectConflictFixed`** concept. Individuals of **`IndirectConflictMager`** have been created automatically, as shown in Fig. 11.

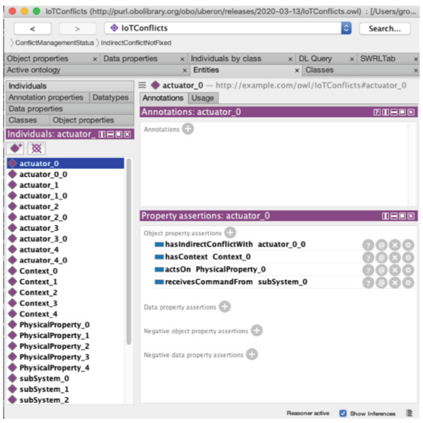
[2] https://protege.stanford.edu.


IoTConflicts (http://purl.obolibrary.org/obo/uberon/releases/2020-03-13/IoTConflicts.owl) : [/Users/gro...
IoTConflicts
Search...
ConflictManagementStatus 〉 IndirectConflictNotFixed
Object properties
Data properties
Individuals by class
DL Query
SWRLTab
Active ontology
Entities
Classes
Class hierarchy (inferred)
Class hierarchy
Class hierarchy: IndirectC
Asserted
Reasoner active
Show Inferences

```
owl:Thing
  ConflictManagementStatus
    DirectConflictFixed
    DirectConflictNotFixed
    IndirectConflictFixed
    IndirectConflictNotFixed
  Context
  Entity
    Actuator
    DirectConflictManager
    IndirectConflictManager
    SubSystem
  PhysicalProperty
```

```
Description: IndirectConflictNotFixed
Equivalent To
  (hasIndirectConflictWith min 1 owl:Thing)
SubClass Of
  (hasIndirectConflictWith min 1 owl:Thing)
  ConflictManagementStatus
General class axioms
  (hasIndirectConflictWith min 1 owl:Thing)
  SubClassOf IndirectConflictNotFixed
SubClass Of (Anonymous Ancestor)
Instances
  actuator_0
  actuator_0_0
  actuator_1
  actuator_1_0
  actuator_2
  actuator_2_0
  actuator_3
  actuator_3_0
  actuator_4
  actuator_4_0
```

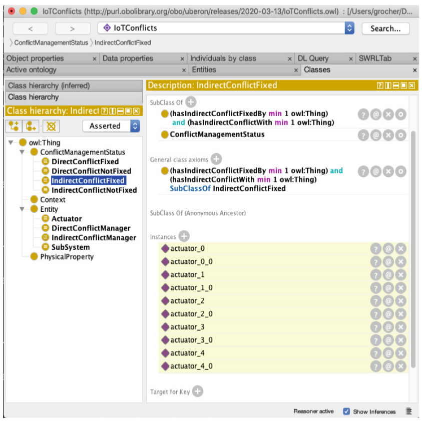
Fig. 8. Actuator individuals potentially subject to indirect conflicts are made directly available under the concept **`IndirectConflictNotFixed`**.



```
actuator_0
actuator_0_0
actuator_1
actuator_1_0
actuator_2
actuator_2_0
actuator_3
actuator_3_0
actuator_4
actuator_4_0
Context_0
Context_1
Context_2
Context_3
Context_4
PhysicalProperty_0
PhysicalProperty_1
PhysicalProperty_2
PhysicalProperty_3
PhysicalProperty_4
subSystem_0
subSystem_1
subSystem_2
```

```
actuator_0 — http://example.com/owl/IoTConflicts#actuator_0
Annotations: actuator_0
Annotations
Property assertions: actuator_0
Object property assertions
  hasIndirectConflictWith  actuator_0_0
  hasContext  Context_0
  actsOn  PhysicalProperty_0
  receivesCommandFrom  subSystem_0
Data property assertions
Negative object property assertions
Negative data property assertions
```

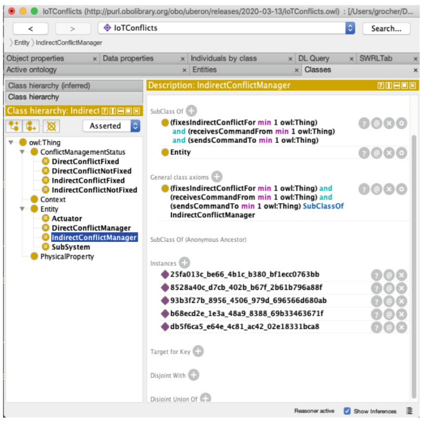
Fig. 9. Object properties associated with each actuator make the cause of conflict clear.


IoTConflicts (http://purl.obolibrary.org/obo/uberon/releases/2020-03-13/IoTConflicts.owl) : [/Users/grocher/D...
IoTConflicts
Search...
ConflictManagementStatus 〉 IndirectConflictFixed
Object properties
Data properties
Individuals by class
DL Query
SWRLTab
Active ontology
Entities
Classes
Class hierarchy (inferred)
Class hierarchy
Class hierarchy: Indirect
Asserted
Reasoner active
Show Inferences

```
owl:Thing
  ConflictManagementStatus
    DirectConflictFixed
    DirectConflictNotFixed
    IndirectConflictFixed
    IndirectConflictNotFixed
  Context
  Entity
    Actuator
    DirectConflictManager
    IndirectConflictManager
    SubSystem
  PhysicalProperty
```

```
Description: IndirectConflictFixed
SubClass Of
  (hasIndirectConflictFixedBy min 1 owl:Thing)
   and (hasIndirectConflictWith min 1 owl:Thing)
  ConflictManagementStatus
General class axioms
  (hasIndirectConflictFixedBy min 1 owl:Thing) and
  (hasIndirectConflictWith min 1 owl:Thing)
  SubClassOf IndirectConflictFixed
SubClass Of (Anonymous Ancestor)
Instances
  actuator_0
  actuator_0_0
  actuator_1
  actuator_1_0
  actuator_2
  actuator_2_0
  actuator_3
  actuator_3_0
  actuator_4
  actuator_4_0
Target for Key
```

Fig. 10. Actuator individuals whose indirect conflicts is fixed by an ACM are directly made available under the concept **`IndirectConflictFixed`**.

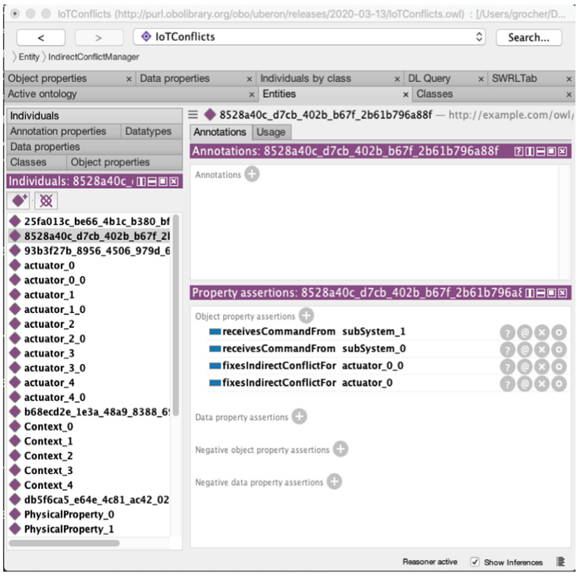

IoTConflicts (http://purl.obolibrary.org/obo/uberon/releases/2020-03-13/IoTConflicts.owl) : [/Users/grocher/D...
IoTConflicts
Search...
Entity 〉 IndirectConflictManager
Object properties
Data properties
Individuals by class
DL Query
SWRLTab
Active ontology
Entities
Classes
Class hierarchy (inferred)
Class hierarchy
Class hierarchy: Indirect
Asserted
Reasoner active
Show Inferences

```
owl:Thing
  ConflictManagementStatus
    DirectConflictFixed
    DirectConflictNotFixed
    IndirectConflictFixed
    IndirectConflictNotFixed
  Context
  Entity
    Actuator
    DirectConflictManager
    IndirectConflictManager
    SubSystem
  PhysicalProperty
```

```
Description: IndirectConflictManager
SubClass Of
  (fixesIndirectConflictFor min 1 owl:Thing)
   and (receivesCommandFrom min 1 owl:Thing)
   and (sendsCommandTo min 1 owl:Thing)
  Entity
General class axioms
  (fixesIndirectConflictFor min 1 owl:Thing) and
  (receivesCommandFrom min 1 owl:Thing) and
  (sendsCommandTo min 1 owl:Thing) SubClassOf
  IndirectConflictManager
SubClass Of (Anonymous Ancestor)
Instances
  25fa013c_be66_4b1c_b380_bf1ecc0763bb
  8528a40c_d7cb_402b_b67f_2b61b796a88f
  93b3f27b_8956_4506_979d_696566d680ab
  b68ecd2e_1e3a_48a9_8388_69b33463671f
  db5f6ca5_e64e_4c81_ac42_02e18331bca8
Target for Key
Disjoint With
Disjoint Union Of
```

Fig. 11. ACM Individuals are made available under the concept **`IndirectConflictManager`**.

The object properties associated with each ACM give designers a better understanding of the actuators involved (Fig. 12). Since ACM are bound to conflicting actuators (**`fixesIndirectConflictFor`**) and their associated subsystems (**`receivesCommandFrom`**), this facilitates their integration into DevOps deployment and implementation models.



IoTConflicts (http://purl.obolibrary.org/obo/uberon/releases/2020-03-13/IoTConflicts.owl) : [/Users/grocher/D...

IoTConflicts | Search...

Entity 〉 IndirectConflictManager

Object properties | Data properties | Individuals by class | DL Query | SWRLTab

Active ontology | Entities | Classes

Individuals | Annotation properties | Datatypes | Data properties | Classes | Object properties

Individuals: 8528a40c_

- 25fa013c_be66_4b1c_b380_bf
- 8528a40c_d7cb_402b_b67f_2l
- 93b3f27b_8956_4506_979d_6
- actuator_0
- actuator_0_0
- actuator_1
- actuator_1_0
- actuator_2
- actuator_2_0
- actuator_3
- actuator_3_0
- actuator_4
- actuator_4_0
- b68ecd2e_1e3a_48a9_8388_6
- Context_0
- Context_1
- Context_2
- Context_3
- Context_4
- db5f6ca5_e64e_4c81_ac42_02
- PhysicalProperty_0
- PhysicalProperty_1

8528a40c_d7cb_402b_b67f_2b61b796a88f — http://example.com/owl/

Annotations | Usage

Annotations: 8528a40c_d7cb_402b_b67f_2b61b796a88f

Annotations

Property assertions: 8528a40c_d7cb_402b_b67f_2b61b796a

Object property assertions

- receivesCommandFrom subSystem_1
- receivesCommandFrom subSystem_0
- fixesIndirectConflictFor actuator_0_0
- fixesIndirectConflictFor actuator_0

Data property assertions

Negative object property assertions

Negative data property assertions

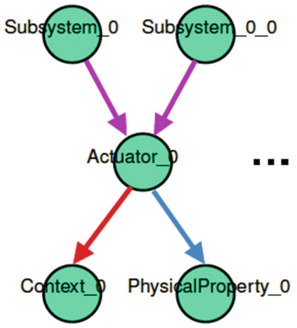
Reasoner active ✓ Show Inferences

Fig. 12. Object properties asserted to each ACM make clear further feedback in deployment and implementation models.

5 Performance Analysis

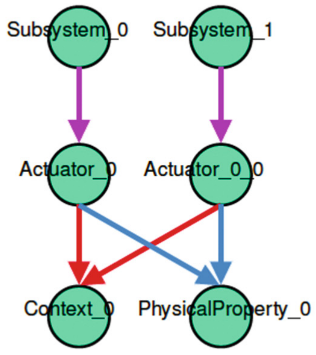
SIS can be implemented from a few dozen (smart homes) to thousands of actuators (smart cities). In this context, it is important to evaluate the performance of the proposed approach to gain insight into its relevance against the targeted SIS. To this end, we propose a set of synthetic actuation conflict configurations that serve as a reference for experiments and benchmarks, divided into four categories defined as follows:

1. The first category represents SIS that have only direct actuation conflicts and follow the pattern below, which is duplicated as many times as the number of direct conflicts requires:



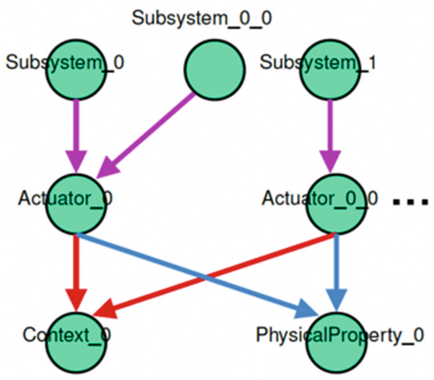
```
<rdf:Description rdf:about="#actuator_0">
<hasContext rdf:resource="#Context_0"/>
<actsOn rdf:resource="#PhysicalProperty_0"/>
</rdf:Description>
<rdf:Description rdf:about="#subSystem_0">
<sendsCommandTo rdf:resource="#actuator_0"/>
</rdf:Description>
<rdf:Description rdf:about="#subSystem_0_0">
<sendsCommandTo rdf:resource="#actuator_0"/>
</rdf:Description>
...
```

2. The second one represents SIS that have only indirect actuation conflicts, following the pattern below duplicated as often as the number of indirect conflicts requires:



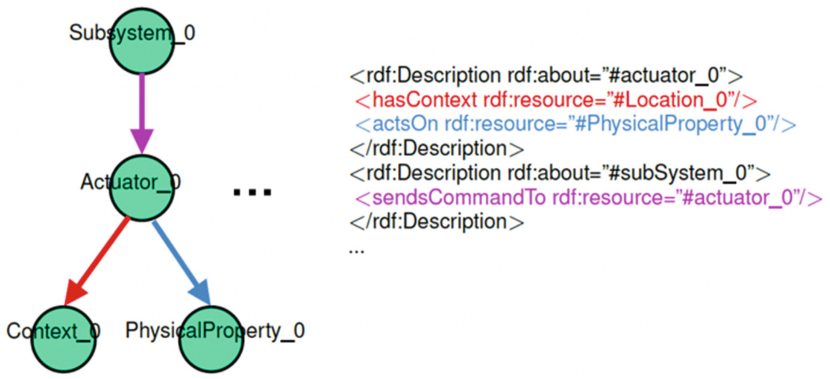
```
<rdf:Description rdf:about="#actuator_0">
 <hasContext rdf:resource="#Context_0"/>
 <actsOn rdf:resource="#PhysicalProperty_0"/>
</rdf:Description>
<rdf:Description rdf:about="#actuator_0_0">
 <hasContext rdf:resource="#Context_0"/>
 <actsOn rdf:resource="#PhysicalProperty_0"/>
</rdf:Description>
<rdf:Description rdf:about="#subSystem_0">
 <sendsCommandTo rdf:resource="#actuator_0"/>
</rdf:Description>
<rdf:Description rdf:about="#subSystem_1">
 <sendsCommandTo rdf:resource="#actuator_0_0"/>
</rdf:Description>
...
```

3. The third category corresponds to SIS that have both direct and indirect actuation conflicts, which are duplicated as many times as the number of indirect conflicts requires, according to the pattern below:



```
<rdf:Description rdf:about="#actuator_0">
 <hasContext rdf:resource="#Context_0"/>
 <actsOn rdf:resource="#PhysicalProperty_0"/>
</rdf:Description>
<rdf:Description rdf:about="#actuator_0_0">
 <hasContext rdf:resource="#Context_0"/>
 <actsOn rdf:resource="#PhysicalProperty_0"/>
</rdf:Description>
<rdf:Description rdf:about="#subSystem_0">
 <sendsCommandTo rdf:resource="#actuator_0"/>
</rdf:Description>
<rdf:Description rdf:about="#subSystem_0_0">
 <sendsCommandTo rdf:resource="#actuator_0"/>
</rdf:Description>
<rdf:Description rdf:about="#subSystem_1">
 <sendsCommandTo rdf:resource="#actuator_0_0"/>
</rdf:Description>
...
```

4. The fourth category corresponds to SIS that have no actuation conflict and are duplicated as many times as the number of actuators requires:



```
<rdf:Description rdf:about="#actuator_0">
 <hasContext rdf:resource="#Location_0"/>
 <actsOn rdf:resource="#PhysicalProperty_0"/>
</rdf:Description>
<rdf:Description rdf:about="#subSystem_0">
 <sendsCommandTo rdf:resource="#actuator_0"/>
</rdf:Description>
...
```

The experiments are conducted using the Protégé tool with either only the actuation conflicts identification rules (rule 1 and rule 2 in Table 1) or both the actuation conflict identification and resolution rules (rules 1, 2, 3, 4, 5 and 6 in Table 1 and Table 2) enabled. The complete experimental setup is defined as follows:

Macbook pro Quad-Core i7 2.8 GHz,
16 GB RAM 2133 Mghz LPDDR3,
Protégé 5.5.0,
(OWL API 4.5.9.2019-02-01T07:24:44Z),
(SWRLTab Protege 5.0+ Plugin (2.0.6)),
Fact++ 1.6.5.

The performance results are shown in Fig. 13 and indicate that processing time of the rules depends on the number of actuation conflicts to be identified and resolved. Without actuation conflicts, the performance results are mainly determined by the number of actuators involved in SIS. These results suggest that the proposed approach is suitable for SIS with no more than a few thousand actuators and a few hundred conflicts (e.g., smart homes, smart buildings, etc.). It should be noted that the DevOps approach is an agile and incremental approach. Due to the consecutive design loops, it is unlikely that the number of potential conflicts detected in the design will be more than a few hundred.

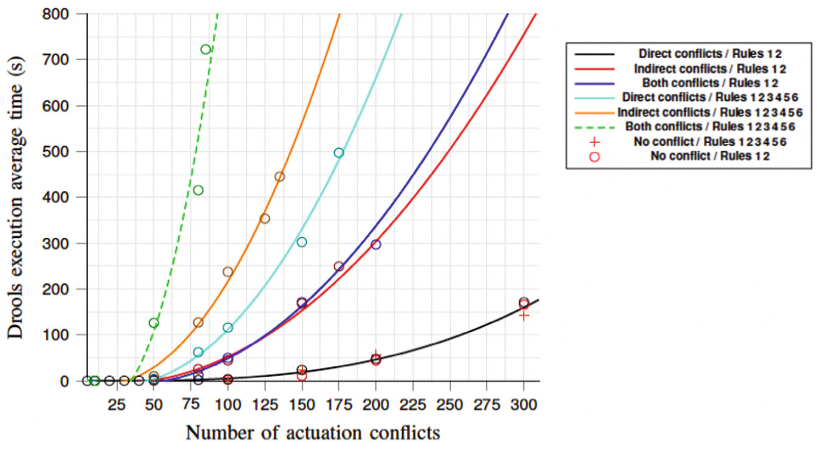


Fig. 13. SWRL rule performance results for different actuation conflict configurations.

6 Future Work

While the identification of actuation conflicts is automatic, their resolution is semi-automatic. Actuation Conflicts Managers (ACM) are automatically instantiated to resolve conflicts at relevant points in the design. However, these ACM are black boxes and require designers to manually select concrete ACM from a set of available off-the-shelf ACM. To better assist designers in this task, we plan to extend the proposed ontology with additional semantics that allow selection of relevant off-the-shelf ACM based on their configuration (input/output types, command types, etc.).

Performance wise, we plan to compare the SWRL-based approach with an approach based on Shapes Constraint Language (SHACL) [21]. SHACL is a standard validation language that allows to define rules whose violations are formalized into reports. It also can be used as a modelling language through SPARQL-based constructs.

Finally, the proposed ontology is simple enough to have its concepts and relationships aligned with those from existing ontologies. For example, we plan to align the proposed ontology with the Smart Applications REFerence ontology (SAREF) and its extensions [22].

7 Conclusion

In the realm of trustworthy Smart IoT-based Systems (SIS), the implementation of actuators at the edge of their infrastructure requires new development tools to help DevOps stakeholders detect and resolve actuation conflicts that can lead to unexpected and potentially harmful behavior as early as possible. While much work as gone into identifying and resolving direct actuation conflicts (concurrent access to a common actuator), little attention has been paid to the indirect conflicts (concurrent access to physical properties) introduced by SIS and the devices they implement at the edge of their infrastructure. These conflicts can be subtle, making them difficult to detect. Their resolution must take into account subjective knowledge related to the context of use of SIS and user expectations.

In this context, a lightweight ontology was proposed to provide DevOps stakeholders with (1) a semantic meta-modelling framework to formally describe SIS and the actuators with which they interact from deployment and integration models; (2) a set of 6 SWRL rules used in conjunction with a DL-based reasoner to automatically identify and semi-automatically resolve actuation conflicts. Performance analysis of the proposed approach for different configurations of actuation conflicts has shown that it is acceptable for SIS with up to a thousand actuators and a few hundred actuation conflicts, making it suitable for smart home, smart building, etc. use-cases.

Acknowledgment. The research leading to these results has received funding from the European Commission's H2020 Program under grant agreement numbers 780351 (ENACT). This work was conducted using the Protégé resource, which is supported by grant GM10331601 from the National Institute of General Medical Sciences of the United States National Institutes of Health.

References

1. Ebert, C., Gallardo, G., Hernantes, J., Serrano, N.: DevOps. IEEE Softw. **33**, 94–100 (2016)
2. Ibrhim, H., Hassan, H., Nabil, E.: A conflicts' classification for IoT-based services: a comparative survey. PeerJ Comput. Sci. **7**, 480 (2021)
3. Harris, S., Seaborne, A., Prud'hommeaux, E.: SPARQL 1.1 query language. W3C Recommendation **21**, 778 (2013)
4. Euzenat, J., Rousset, M.-C.: Semantic web. In: Marquis, P., Papini, O., Prade, H. (eds.) A Guided Tour of Artificial Intelligence Research, pp. 181–207. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-06170-8_6
5. Huang, B., Dong, H., Bouguettaya, A.: Conflict detection in IoT-based smart homes. arXiv:2107.13179 (2021)
6. Xiao, D., Wang, Q., Cai, M., Zhu, Z., Zhao, W.: A3ID: an automatic and interpretable implicit interference detection method for smart home via knowledge graph. IEEE Internet of Things J. **7**, 2197–2211 (2019)
7. Camacho, R.J.L.: Intelligent actuation in home and building automation systems. Master's thesis (2014)
8. Rocher, G., et al.: An actuation conflicts management flow for smart iot-based systems. In: 7th International Conference on Internet of Things: Systems, Management and Security (IOTSMS) (2020)
9. Ferry, N., et al.: Genesis: continuous orchestration and deployment of smart IoT systems. In: IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), pp. 870–875 (2019)
10. Heflin, J.: OWL web ontology language-use cases and requirements. W3C Recommendation (2004)
11. McBride, B.: The resource description framework (RDF) and its vocabulary description language RDFS. In: Staab, S., Studer, R. (eds.) Handbook on Ontologies, pp. 51–65. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24750-0_3
12. Van Harmelen, F., McGuinness, D.L.: OWL web ontology language overview. World Wide Web Consortium (W3C) Recommendation (2004)

13. Colomo-Palacios, R.: Semantic competence pull: a semantics-based architecture for filling competency gaps in organizations. In: Global, I. (ed.) Semantic Web for Business: Cases and Applications, pp. 321–335 (2009)
14. Mark, B.: Theory of Knowledge: Structures and Processes. World scientific (2016)
15. Lawan, A., Rakib, A.: The semantic web rule language expressiveness extensions-a survey. arXiv preprint arXiv:1903.11723 (2019)
16. Horrocks, I., et al.: SWRL: a semantic web rule language combining OWL and RuleML. W3C Member Submission **21**, 1–31 (2004)
17. Rosati, R.: Semantic and computational advantages of the safe integration of ontologies and rules. In: Fages, F., Soliman, S. (eds.) PPSWR 2005. LNCS, vol. 3703, pp. 50–64. Springer, Heidelberg (2005). https://doi.org/10.1007/11552222_6
18. Musen, M.A.: The protégé project: a look back and a look forward. AI Matters **1**, 4–12 (2015)
19. Browne, P.: JBoss Drools Business Rules. Packt Publishing Ltd. (2009)
20. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: system description. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 292–297. Springer, Heidelberg (2006). https://doi.org/10.1007/11814771_26
21. Knublauch, H., Kontokostas, D.: Shapes constraint language (SHACL). W3C Recommendation (2017)
22. Daniele, L., Hartog, F., Roes, J.: Created in close interaction with the industry: the smart appliances reference (saref) ontology. In: Cuel, R., Young, R. (eds.) FOMI 2015. LNBIP, vol. 225, pp. 100–112. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21545-7_9