Orange research program

Smart Object Management

Orange research domain

Augmented Operations

Inferring Threatening IoT Dependencies using Semantic Digital Twins

Toward Collaborative IoT Device Management

Presented by

Amal GUITTOUM – PhD student

Contributors

François AISSAOUI (Orange)
Sébastien BOLLE (Orange)
Noel DE PALMA (LIG)
Fabienne BOYER (LIG)

Eclipse IoT days – Grenoble, January 19^{th}, 2023

Plan

1 Context, Problem statement, Goal

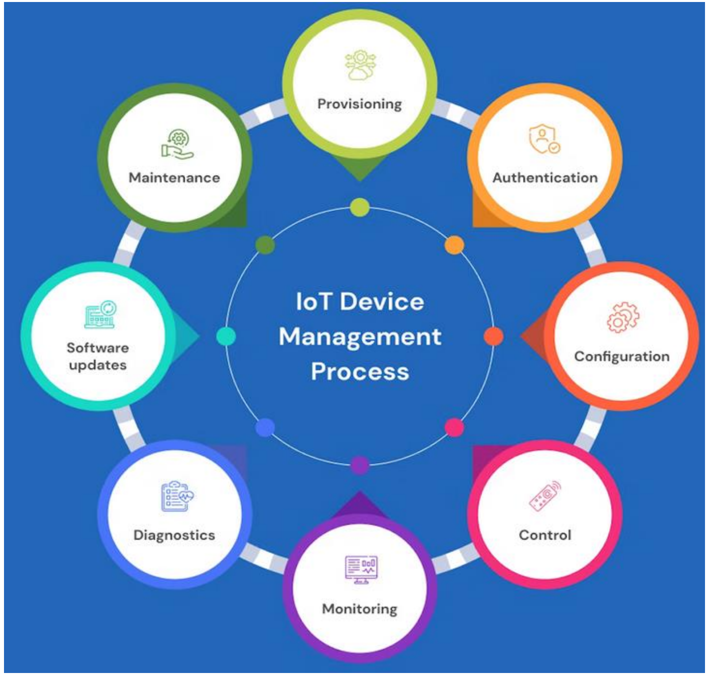
2 Framework

4 Business Use Case: Cascading Failure management in a Smart Home

5 Conclusion and future work

Context: IoT Device Management (DM)

IoT Device Management (DM) is
“An essential set of management capabilities in the Internet of things (IoT), providing support for, but not being limited to device remote activation and deactivation, diagnostics, firmware/software updating and sensor node working status management” [ITU-T,Y.4000Y.4702].

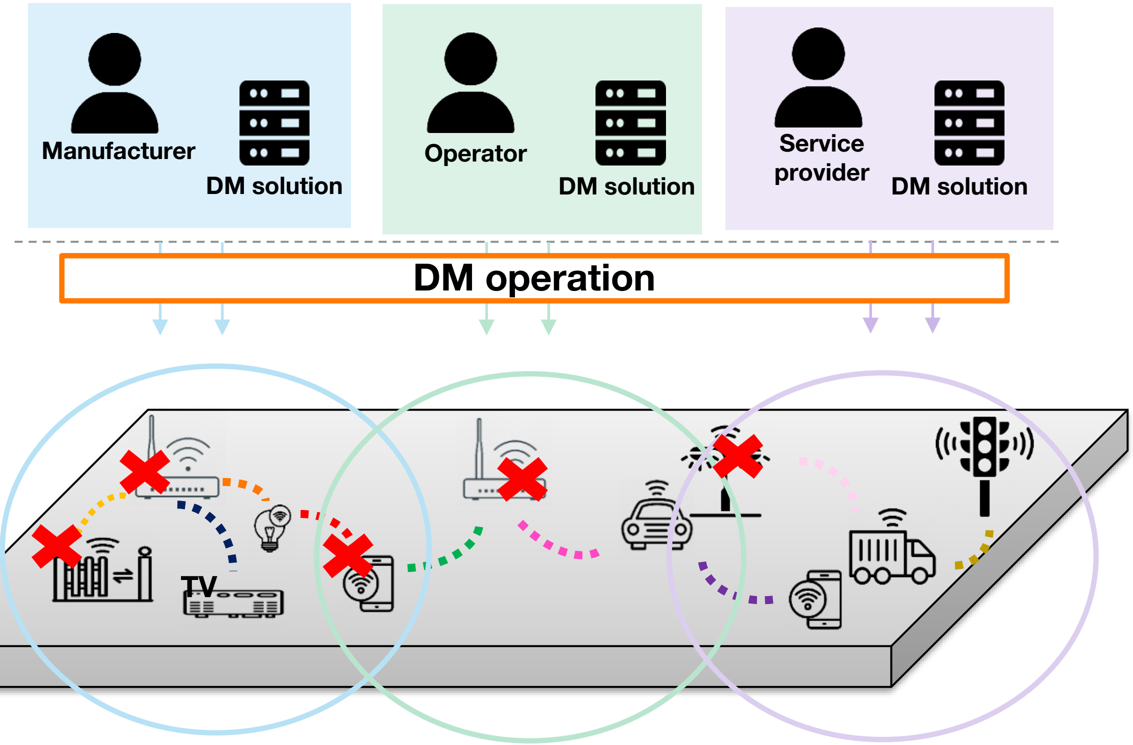


https://www.intuz.com/guide-on-iot-device-management

Context: multi-actor, siloed DM

In practice, DM is ensured by multiple actors, each actor proposes its own DM solution.

- ❑ **Siloed and chaotic management of IoT devices by different actors.**
- ❑ **Cascading failures are hard to manage by these siloed DM solution.**



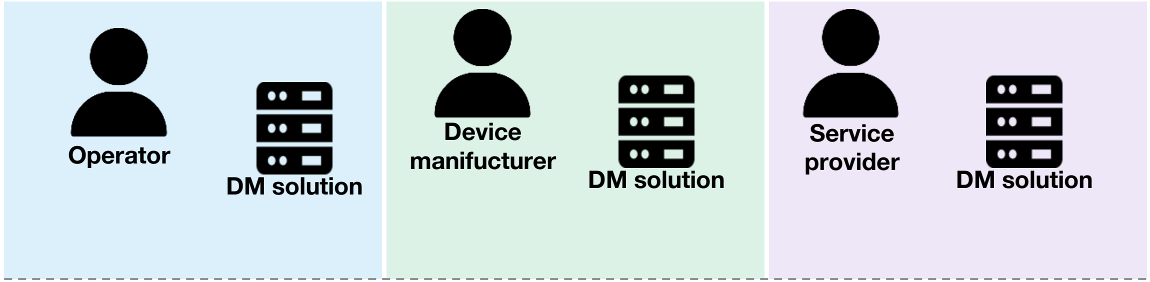
Problem statement

The first step to help these siloed DM solutions in managing cascading failure is the identification of IoT dependency topology.

However,

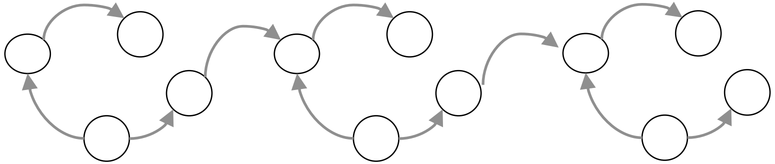
- ⚠ **IoT dependencies are dynamic.**
- ⚠ **IoT dependencies are governed by different actors.**
- ⚠ **IoT dependencies are represented using heterogenous data models.**

Goal



Dependencies as a service

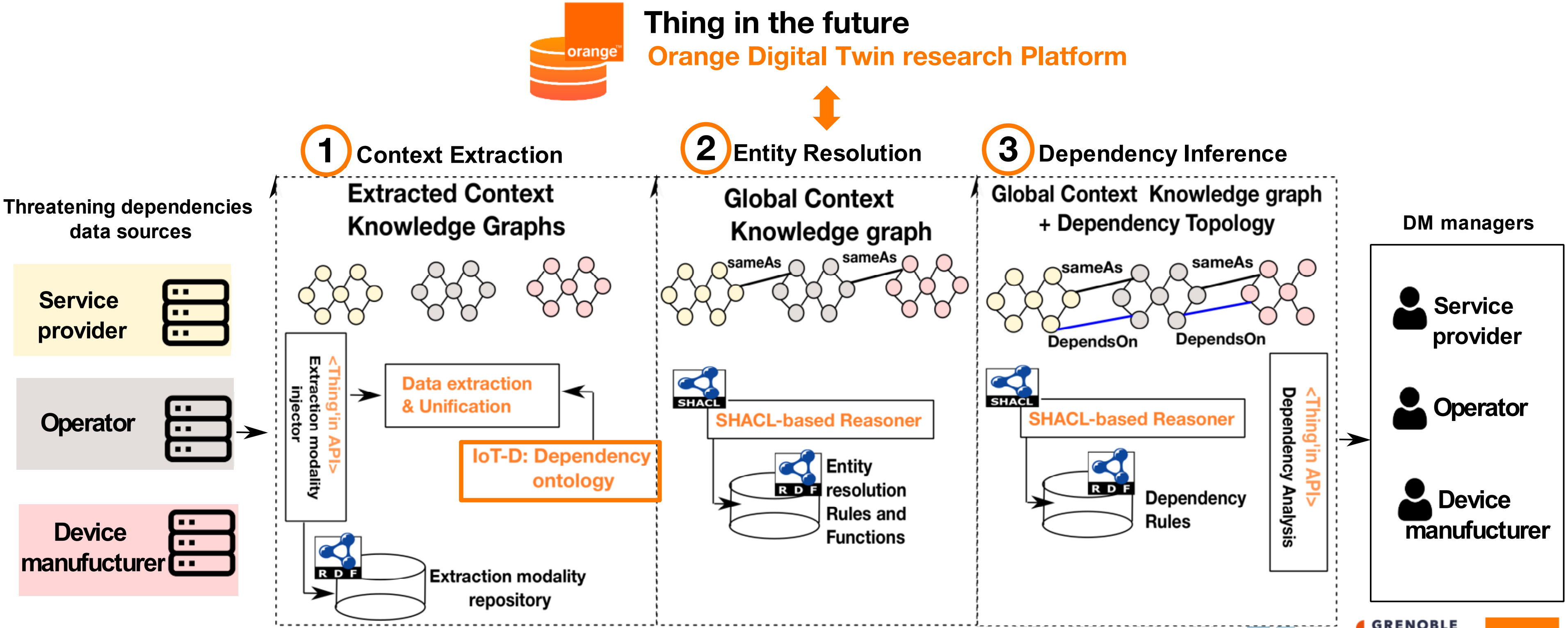
A unified, global view of IoT dependencies



Federative Entity

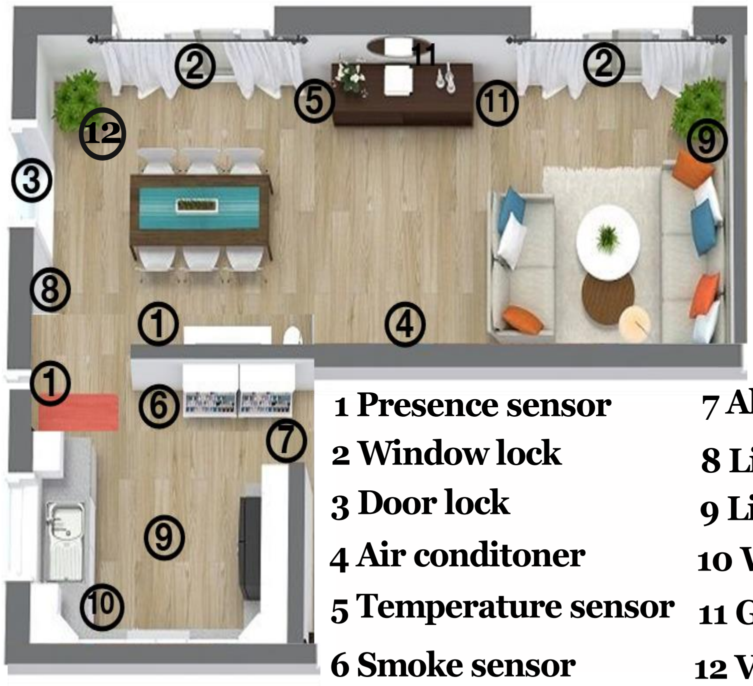
Framework

- Collaborative and automatic identification of dependencies between IoT devices.
- Based on Digital Twin technology and Semantic Web standards.

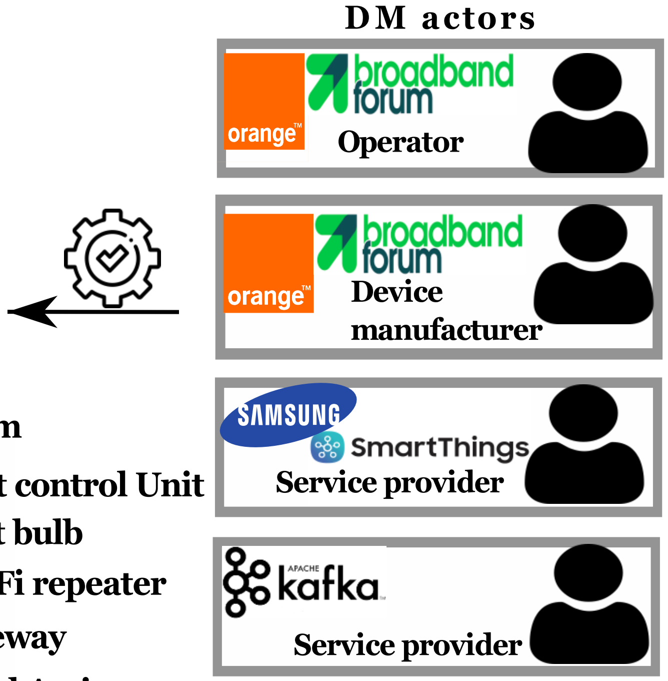


Business Use Case: Cascading Failure management in a Smart Home

- Cascading Failure management in a Smart Home managed by multiple DM actors such as Orange and Samsung.

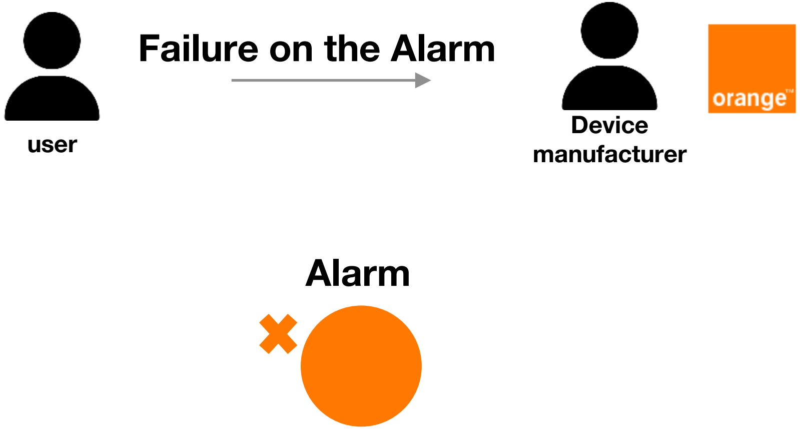


1 Presence sensor
2 Window lock
3 Door lock
4 Air conditoner
5 Temperature sensor
6 Smoke sensor
7 Alarm
8 Light control Unit
9 Light bulb
10 Wi-Fi repeater
11 Gateway
12 Vocal Assistant



Business Use Case: Cascading Failure management in a Smart Home

Consider the following scenario:



user

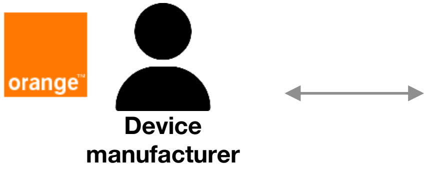
Failure on the Alarm

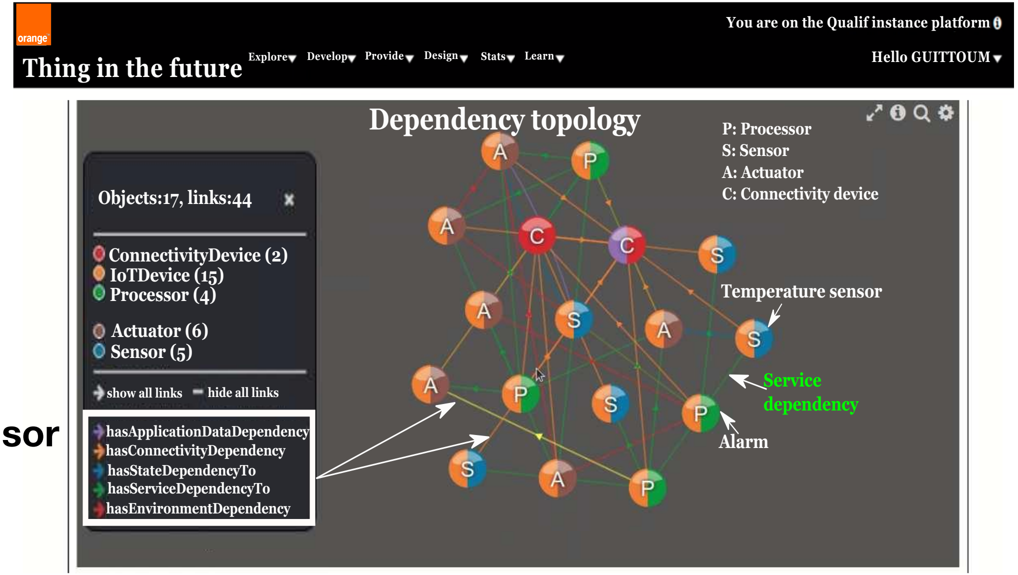
Device manufacturer

orange

Alarm

Business Use Case: Cascading Failure management in a Smart Home




You are on the Qualif instance platform

Thing in the future

Hello GUITTOUM

Dependency topology

P: Processor
S: Sensor
A: Actuator
C: Connectivity device

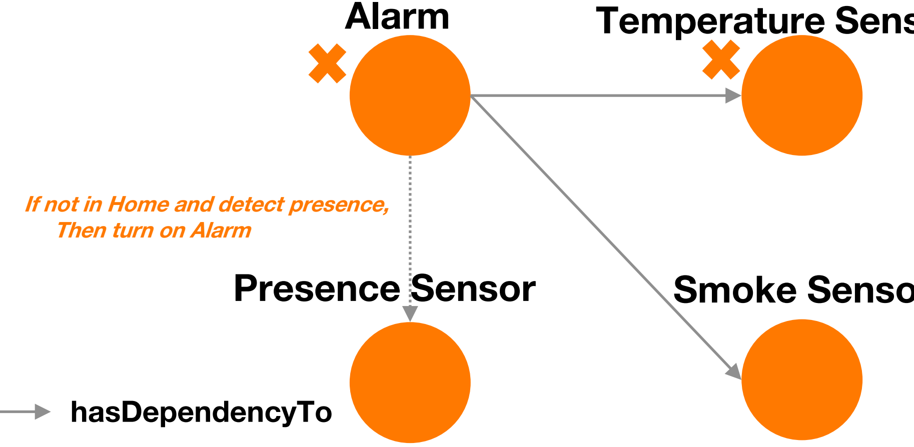
Objects:17, links:44

- ConnectivityDevice (2)
- IoTDevice (15)
- Processor (4)
- Actuator (6)
- Sensor (5)

show all links — hide all links

- hasApplicationDataDependency
- hasConnectivityDependency
- hasStateDependencyTo
- hasServiceDependencyTo
- hasEnvironmentDependency

Analysis results:



If not in Home and detect presence, Then turn on Alarm

Conclusion and future work

Conclusion

A standard-based framework for dependencies inference in heterogenous, multi-actor scenario.

Combine Semantic Web and Digital Twin technologies.

Integrated to Orange digital twin research platform Thing in the future.

Future Work

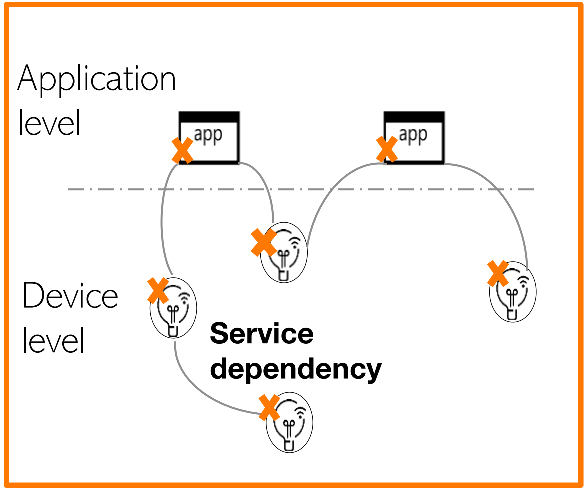
A multi-agent system to enable autonomous and collaborative cascading failure management across heterogenous DM solutions.

Thank you

Appendix

Context: Dependencies-related threats

The isolated DM solutions are limited when dealing with dependencies-related threats.



Application level

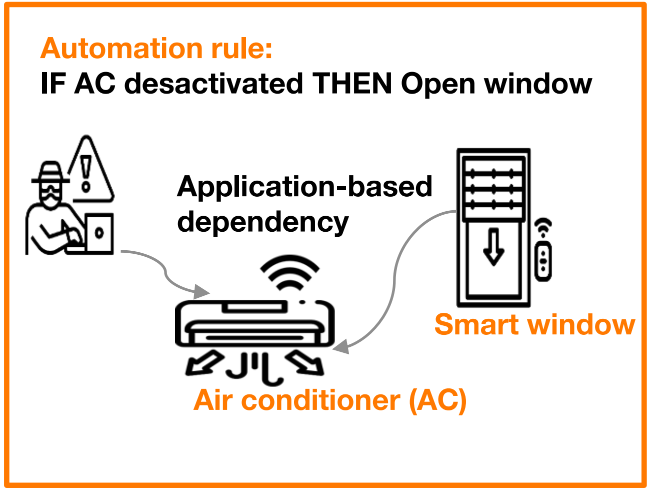
app

app

Device level

Service dependency

1 Cascading failure



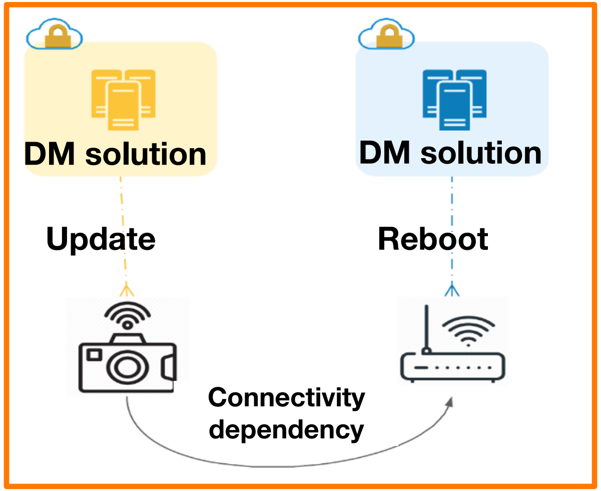
Automation rule:
IF AC desactivated THEN Open window

Application-based dependency

Smart window

Air conditioner (AC)

2 Attacks



DM solution

DM solution

Update

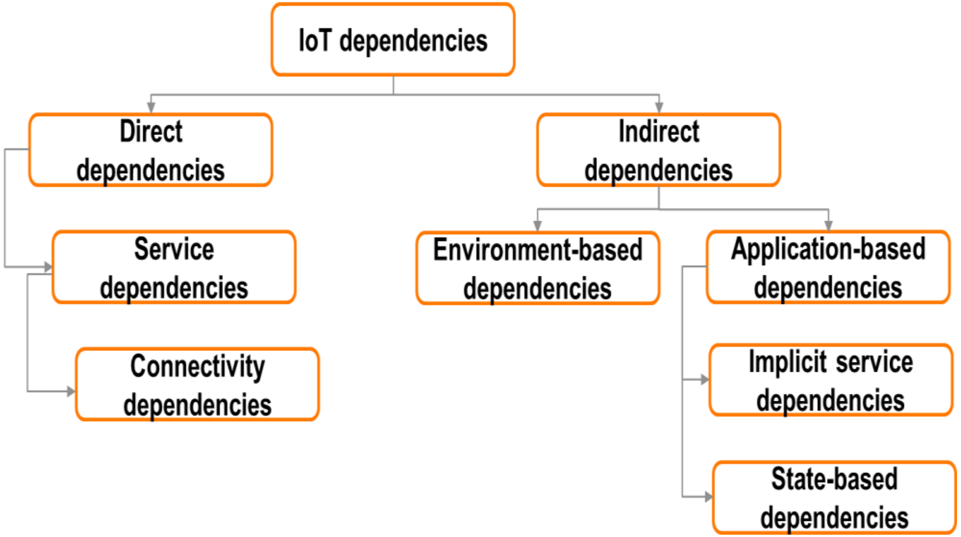
Reboot

Connectivity dependency

3 DM failures

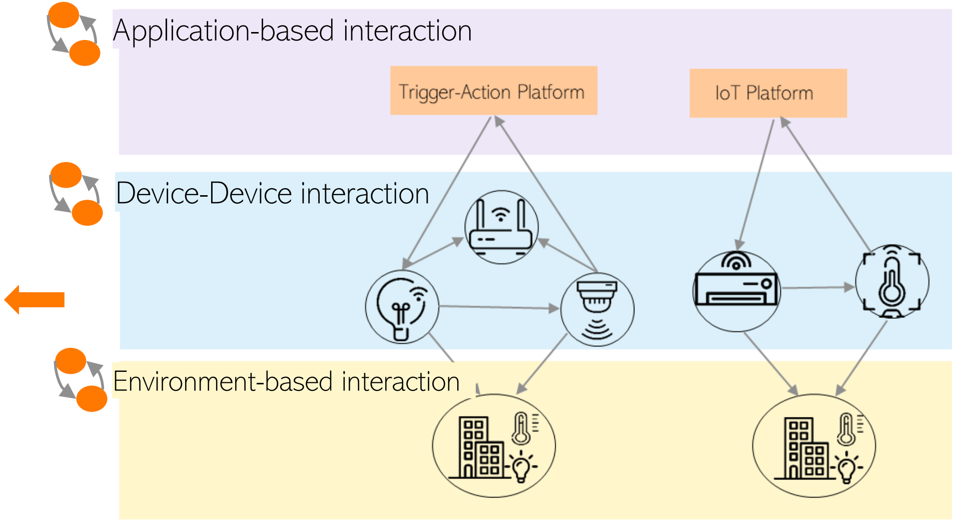
Framework: IoT Dependency Characterization

- ✓ Interactions among IoT devices generate direct and indirect dependencies between them.
- ✓ Dependencies are unidirectional relations between two IoT devices.
- ✓ An IoT device is directly dependent on another if it explicitly uses it to accomplish its function.
- ✓ An IoT device is indirectly dependent on another if its behavior is dependent on it.



IoT dependencies
Direct dependencies
Indirect dependencies
Service dependencies
Connectivity dependencies
Environment-based dependencies
Application-based dependencies
Implicit service dependencies
State-based dependencies

IoT dependency Taxonomy

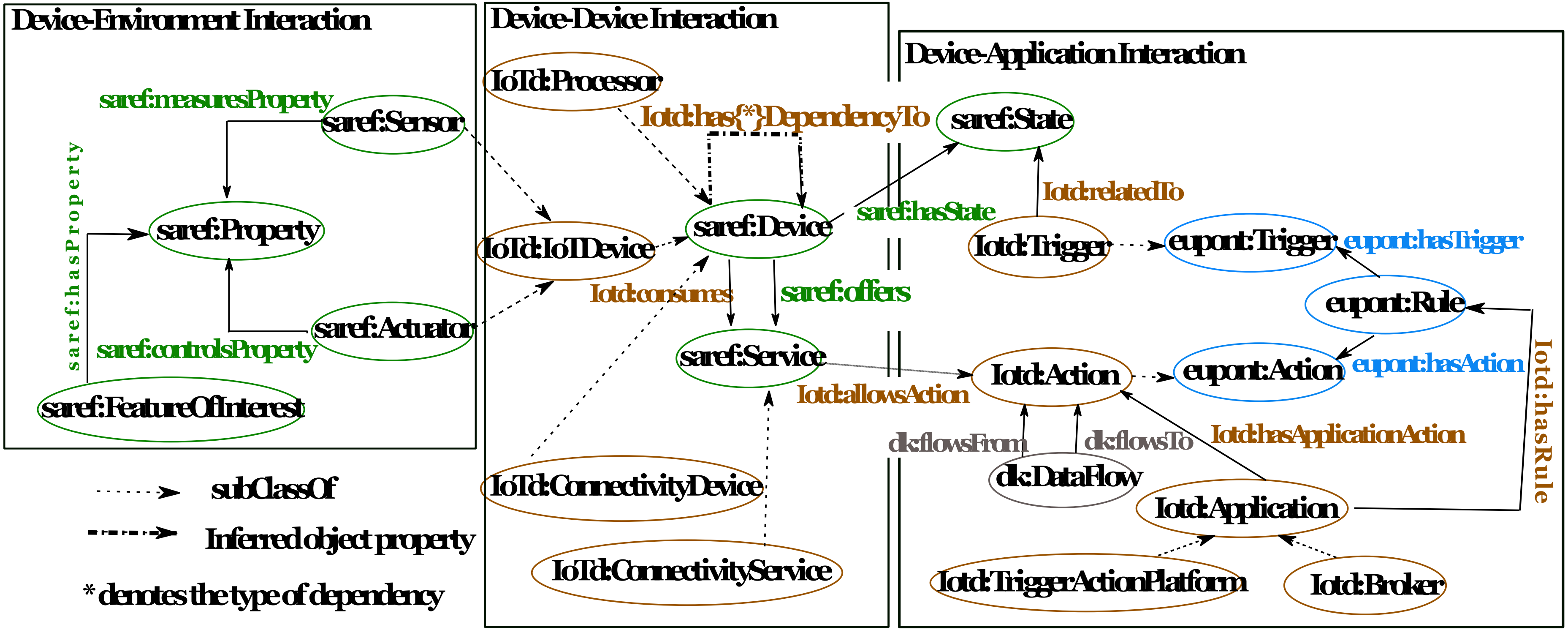


Application-based interaction
Trigger-Action Platform
IoT Platform
Device-Device interaction
Environment-based interaction

IoT behavior: A multi-level interaction-based behavior

Framework: IoT Dependency Modeling

IoT-D is a semantic ontology that allows a unified representation of context data describing dependencies between IoT devices.



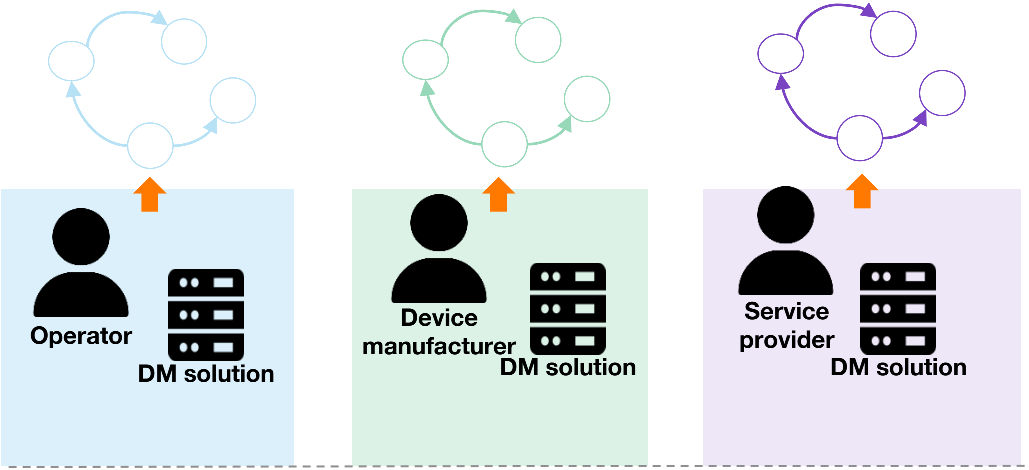
Framework: Context extraction

Extracts dependencies context data from DM solutions, then transforms the extracted data to a knowledge graph according to the IoT-D model.

Hypothesis: The DM actors provide the data extraction modalities from DM solutions via Thing'in, using the standard de W3C Thing Description

Output:

Extracted knowledge graphs



```
:Gateway rdf:type
                IoTd:ConnectivityDevice;
td:hasPropertyAffordance[
   td:hasForm [
       hctl:forContentType
                "application/json";
        hctl:hasOperationType
                td:readProperty ;
hctl:hasTarget   "{$USPLink}/dataModel
=Device.IEEE1905.NetTopology."]
].
```

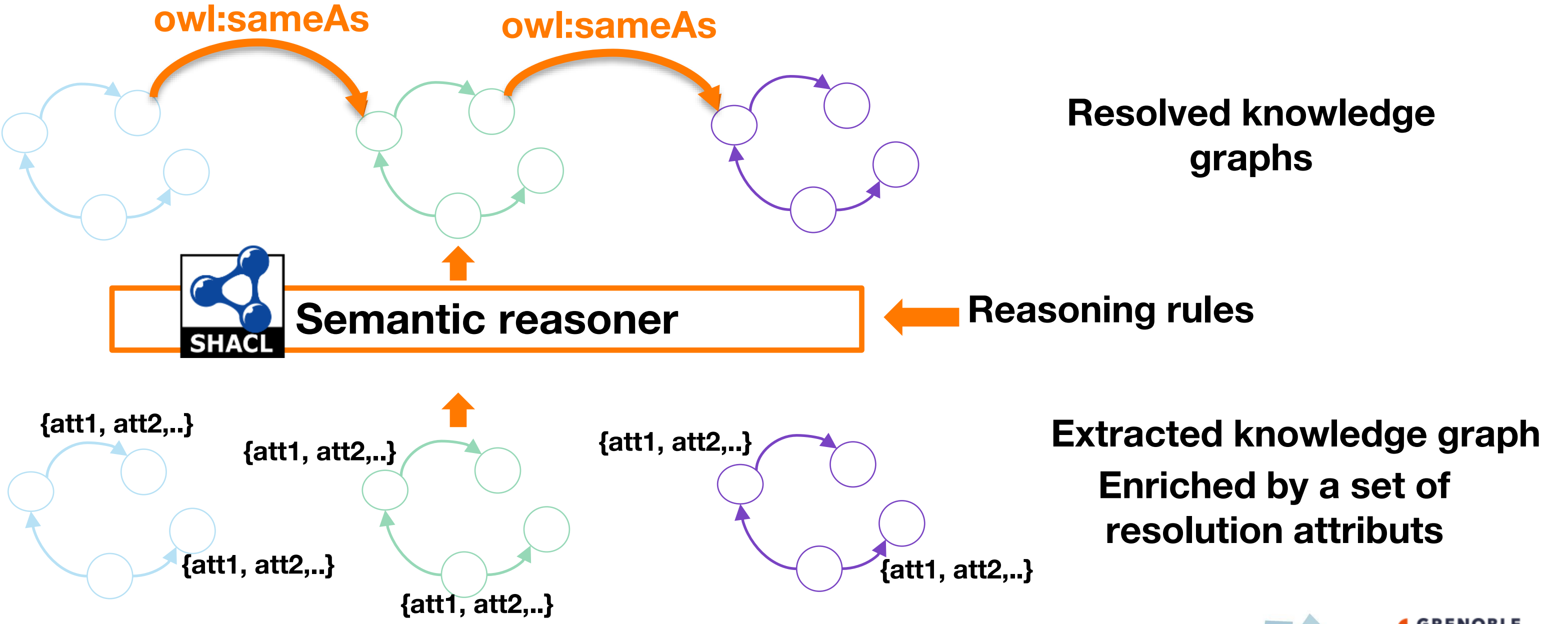
Example of a modality of data extraction

Framework: Entity resolution

Identifies and links similar entities in the extracted knowledge graphs, using SHACL semantic reasoning another W3C standard.

Hypothesis: The extracted knowledge graphs contain a set of resolution attributes for each entity.

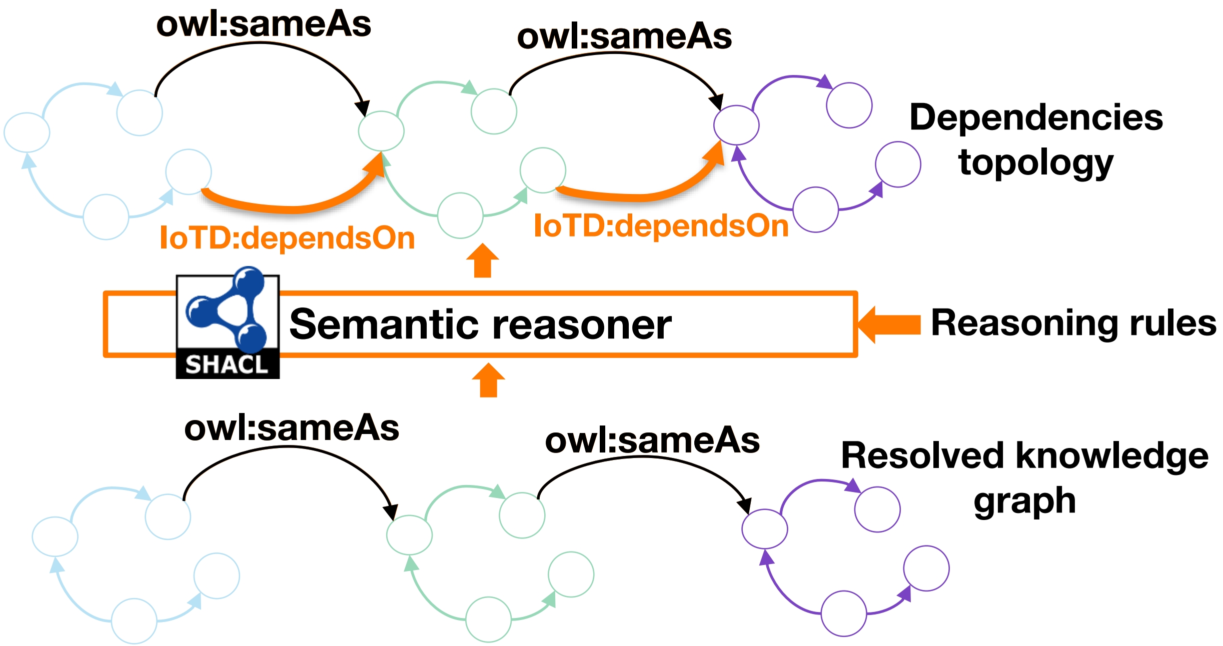
Output:

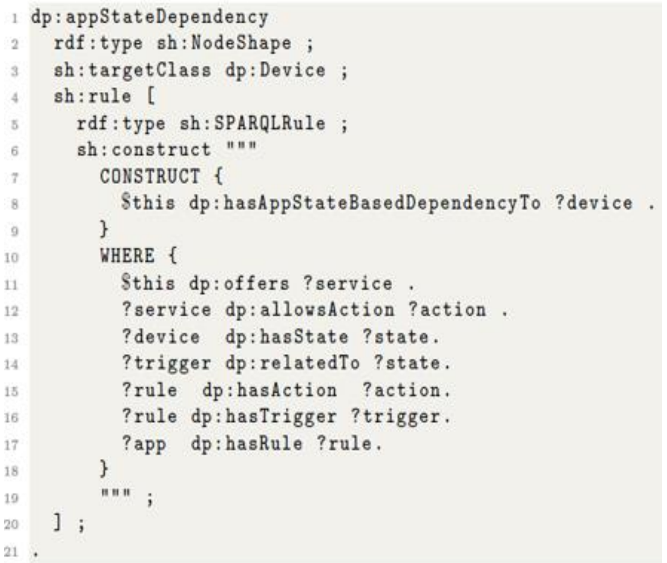


Framework: Dependency inference

Identifies the dependency topology from the resolved knowledge graph using SHACL semantic reasoning.

Output:

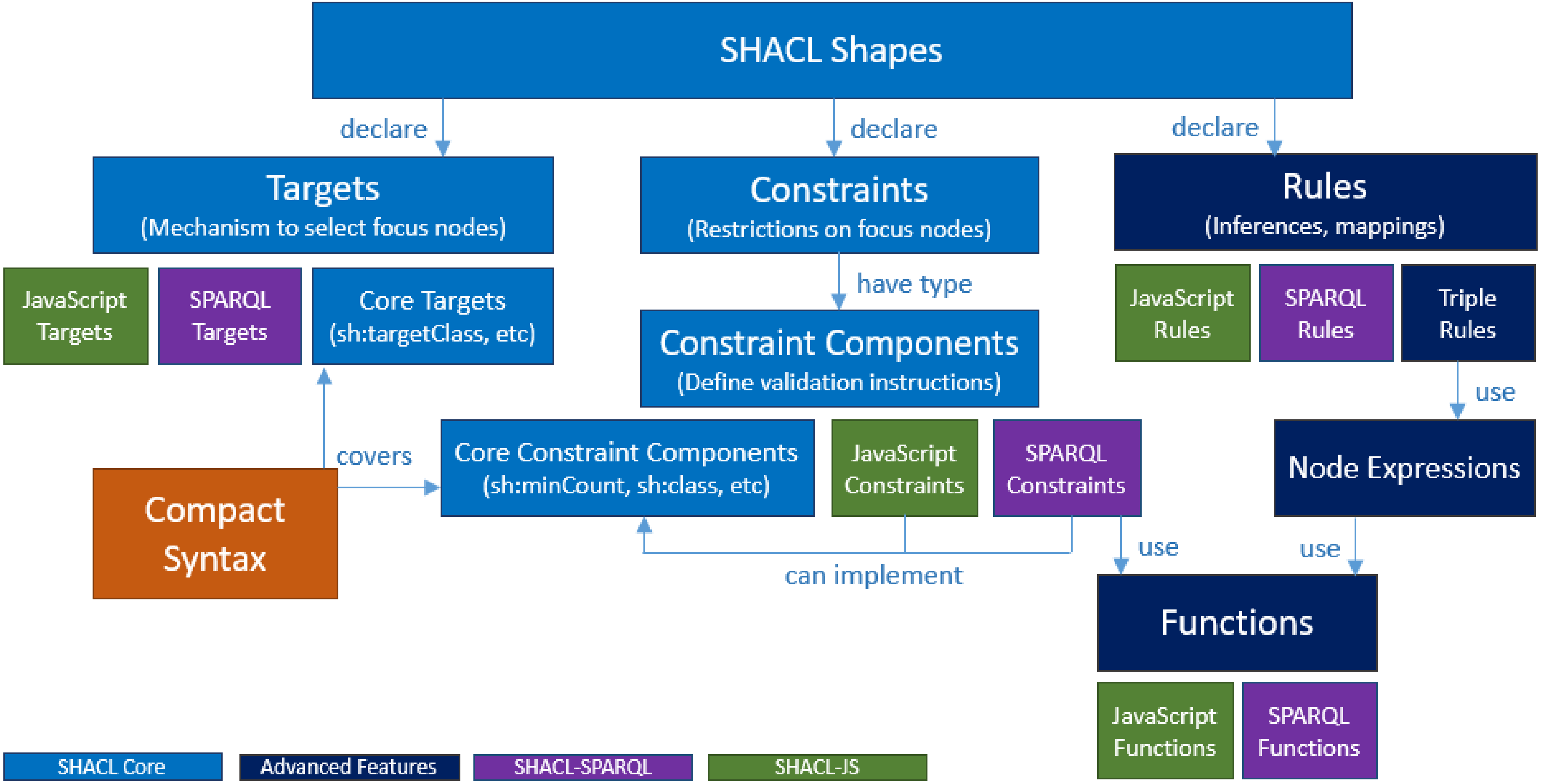




```
dp:appStateDependency
  rdf:type sh:NodeShape ;
  sh:targetClass dp:Device ;
  sh:rule [
    rdf:type sh:SPARQLRule ;
    sh:construct """
      CONSTRUCT {
        $this dp:hasAppStateBasedDependencyTo ?device .
      }
      WHERE {
        $this dp:offers ?service .
        ?service dp:allowsAction ?action .
        ?device  dp:hasState ?state.
        ?trigger dp:relatedTo ?state.
        ?rule  dp:hasAction  ?action.
        ?rule dp:hasTrigger ?trigger.
        ?app  dp:hasRule ?rule.
      }
      """ ;
  ] ;
.
```

Example of state-based dependency inference rule

SHACL standard features


SHACL Shapes

declare

Targets
(Mechanism to select focus nodes)

declare

Constraints
(Restrictions on focus nodes)

declare

Rules
(Inferences, mappings)

JavaScript Targets

SPARQL Targets

Core Targets (sh:targetClass, etc)

have type

Constraint Components
(Define validation instructions)

JavaScript Rules

SPARQL Rules

Triple Rules

use

covers

Compact Syntax

Core Constraint Components (sh:minCount, sh:class, etc)

JavaScript Constraints

SPARQL Constraints

Node Expressions

can implement

use

use

Functions

JavaScript Functions

SPARQL Functions

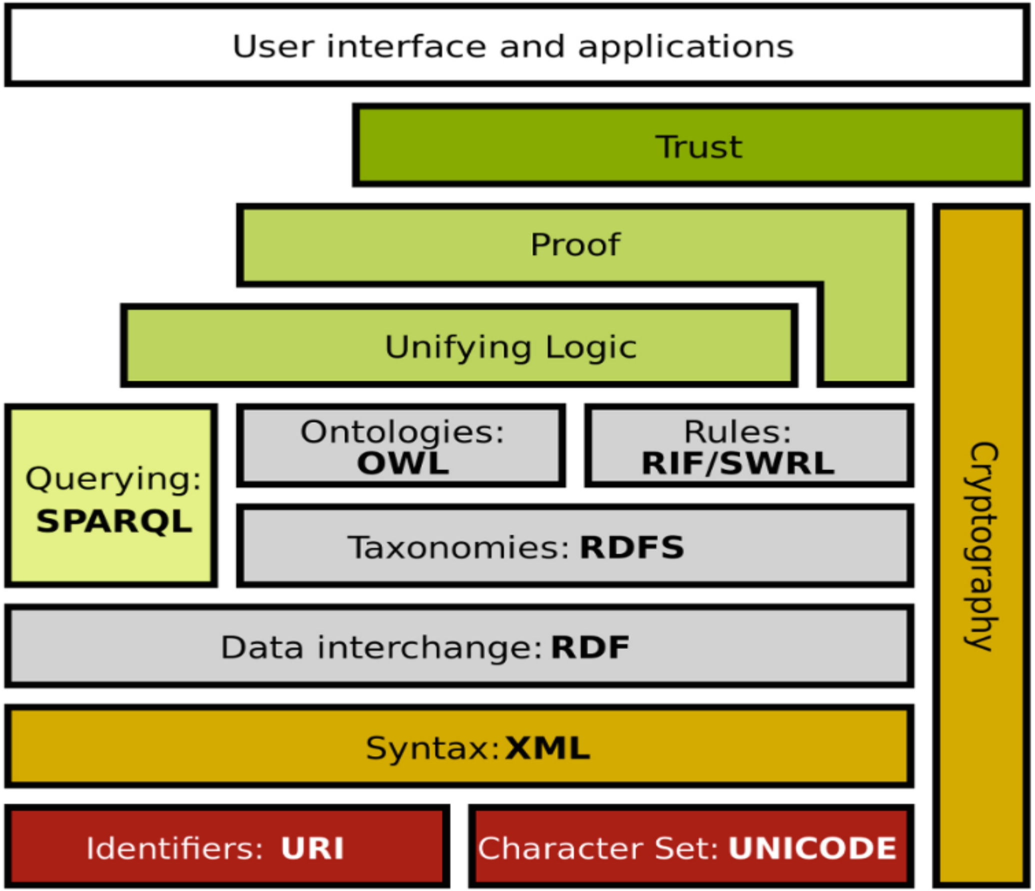
SHACL Core

Advanced Features

SHACL-SPARQL

SHACL-JS

Semantic Web stack


User interface and applications
Trust
Proof
Unifying Logic
Querying: SPARQL
Ontologies: OWL
Rules: RIF/SWRL
Taxonomies: RDFS
Data interchange: RDF
Syntax: XML
Identifiers: URI
Character Set: UNICODE
Cryptography