Semantics Driven Agent Programming

Candidato: Ing. Francesco Antoniazzi
Relatore: Prof. Tullio Salmon Cinotti
Coordinatore Dottorato: Prof. Davide Sangiorgi

Dottorato in Computer Science and Engineering - XXXII ciclo
Alma Mater Studiorum Università di Bologna

Lyon - Bologna, 2 Aprile 2020

francesco.antoniazzi@unibo.it

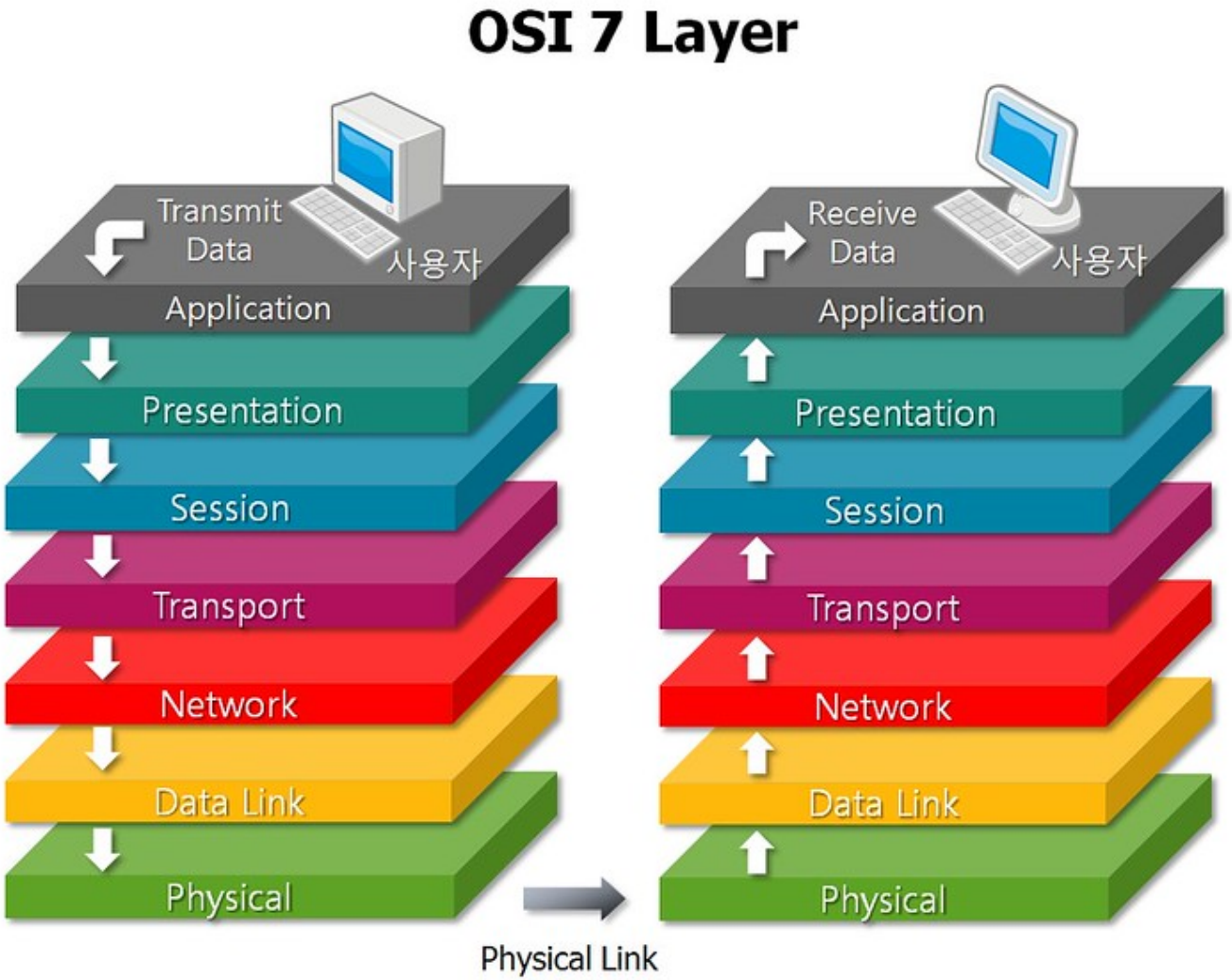
Outline

Internet of Things

State of the Art

- Providing internet connectivity to all sort of devices: everyday items, machines, clothing, buildings, ..., in order to collect information on entities and environments. And realize applications;
- Interoperability issues: vertical silos;
- Breaking fragmentation means creating a bridge at some level of ISO-OSI stack;

Breaking fragmentation



Breaking fragmentation

Let us just consider Application layer protocols, like MQTT and AMQP. The *topic* communication implies that there is an agreement on the semantics among application:

1. The topic URI is interpreted in the same way;
   this is the place in which temperature information should be stored

Breaking fragmentation

Let us just consider Application layer protocols, like MQTT and AMQP. The *topic* communication implies that there is an agreement on the semantics among application:

1. The topic URI is interpreted in the same way;
   this is the place in which temperature information should be stored
2. The exchanged information has the same format;
   information should be provided as a JSON file compliant with a specific JSONSchema

Breaking fragmentation

Let us just consider Application layer protocols, like MQTT and AMQP. The *topic* communication implies that there is an agreement on the semantics among application:

1. The topic URI is interpreted in the same way;
   this is the place in which temperature information should be stored
2. The exchanged information has the same format;
   information should be provided as a JSON file compliant with a specific JSONSchema
3. The exchanged information has the same semantics;
   no misunderstanding is possible on the meaning of the received value:
   `mouse` *means '*  *' for all participants, and is never intended as '*  *'*

Breaking fragmentation

But how do we agree on all this, if applications are developed in different places, different times, by different people, with different purposes?

The Resource Description Framework (RDF) can enrich with semantics all the Application and Information level of the IoT.

Semantic Internet of Things

What's a *Thing*?

IEEE definition

The Thing is any physical object ***relevant*** *from a user or application perspective.*

What's a *Thing*?

IEEE definition

*The Thing is any physical object **relevant** from a user or application perspective.*

W3C definition

*Things can be **virtual representations** of physical or abstract entities. They can be connected or not connected. Each thing can have one or more virtual representations. Things can have histories, and have identities, rich **descriptions**, services, access control and data handling policies. They have URIs.*

Ref. [1]

Internet of Musical Things (IoMusT) Ontology

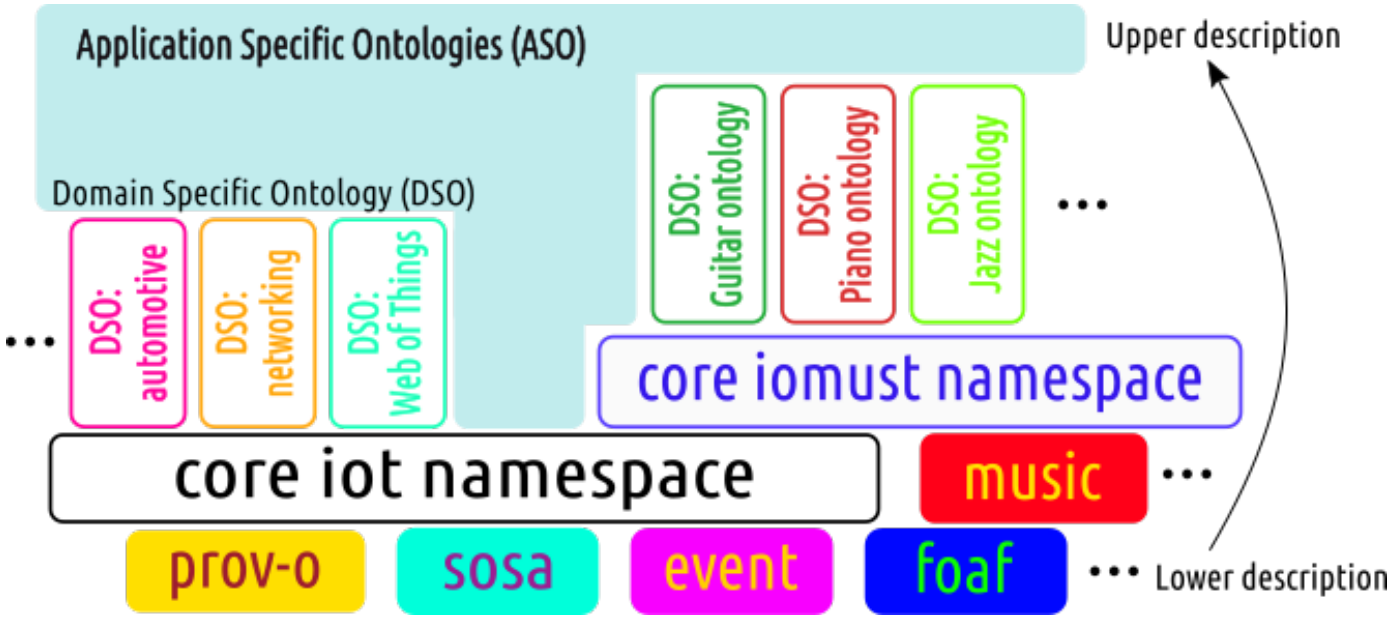
IEEE approach

The Internet of Musical Things is the ensemble of interfaces, protocols and representations of music-related information that enable services and applications serving a musical purpose based on interactions between humans and Musical Things or between Musical Things themselves, in physical and/or digital realms.

Music-related information refers to data sensed and processed by a Musical Thing, and/or exchanged with a human or with another Musical Thing [2].

Internet of Musical Things (IoMusT) Ontology

IEEE approach



- Is IoMusT a subcase of IoT?
- It presents an approach, a model to realize a semantic and fully interoperable IoT connecting different fields incrementally;
- Topic completely unexplored in literature: music+IoT+semantics [3];

IoMusT Ontology

	ns:Bob	ns:Wardrobe	ns:SmartCar	ns:Violin	ns:SmartViolin	ns:TShirt	ns:StageLight	ns:VR_HeadSet	ns:HeartBeatSensor for music experiment
foaf:Person	✔								
iot:Thing		✔	✔	✔	✔	✔	✔	✔	✔
iot:SmartThing			✔		✔		✔	✔	
iot:ConnectedThing					✔		✔	✔	✔
iot:WearableThing						✔		✔	✔
mo:Instrument				✔	✔				
iomust:MusicalThing				✔	✔		✔		✔
iomust:SmartMusicalThing					✔		✔		
iomust:SmartInstrument					✔				
iomust:StageEquipment item							✔		
iomust:WearableMusicalThing									✔

IoMusT Ontology

Rule Example 1 (not reversible!)

```
?s a iot:Thing, mo:Instrument        ⇒        ?s a iomust:MusicalThing
```

Where the Musical Thing is *a thing used to produce or enjoy music, with reference to its context*.

IoMusT Ontology

Rule Example 1 (not reversible!)

```
?s a iot:Thing, mo:Instrument  ⇒  ?s a iomust:MusicalThing
```

Where the Musical Thing is *a thing used to produce or enjoy music, with reference to its context*.

Rule Example 2

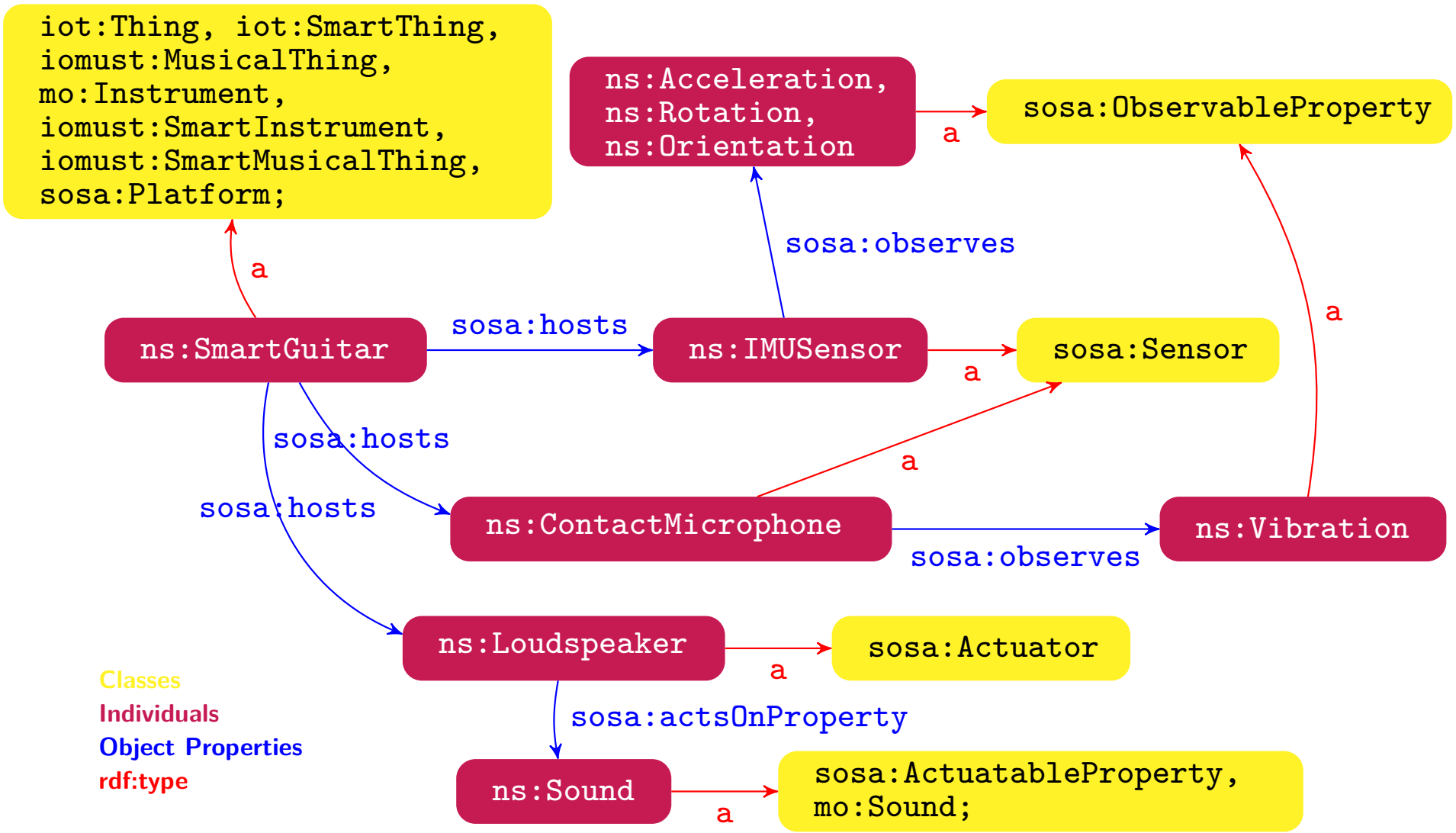
```
?e co:elementOf ?s; a iomust:MusicalThing.  ?s a co:collection
                                          ⇔ ?s a iomust:StageEquipment
```

IoMusT Ontology example I

How do we describe a Smart Guitar with:

1. An IMU sensor unit;
2. A contact microphone;
3. A loudspeaker;

IoMusT Ontology example I

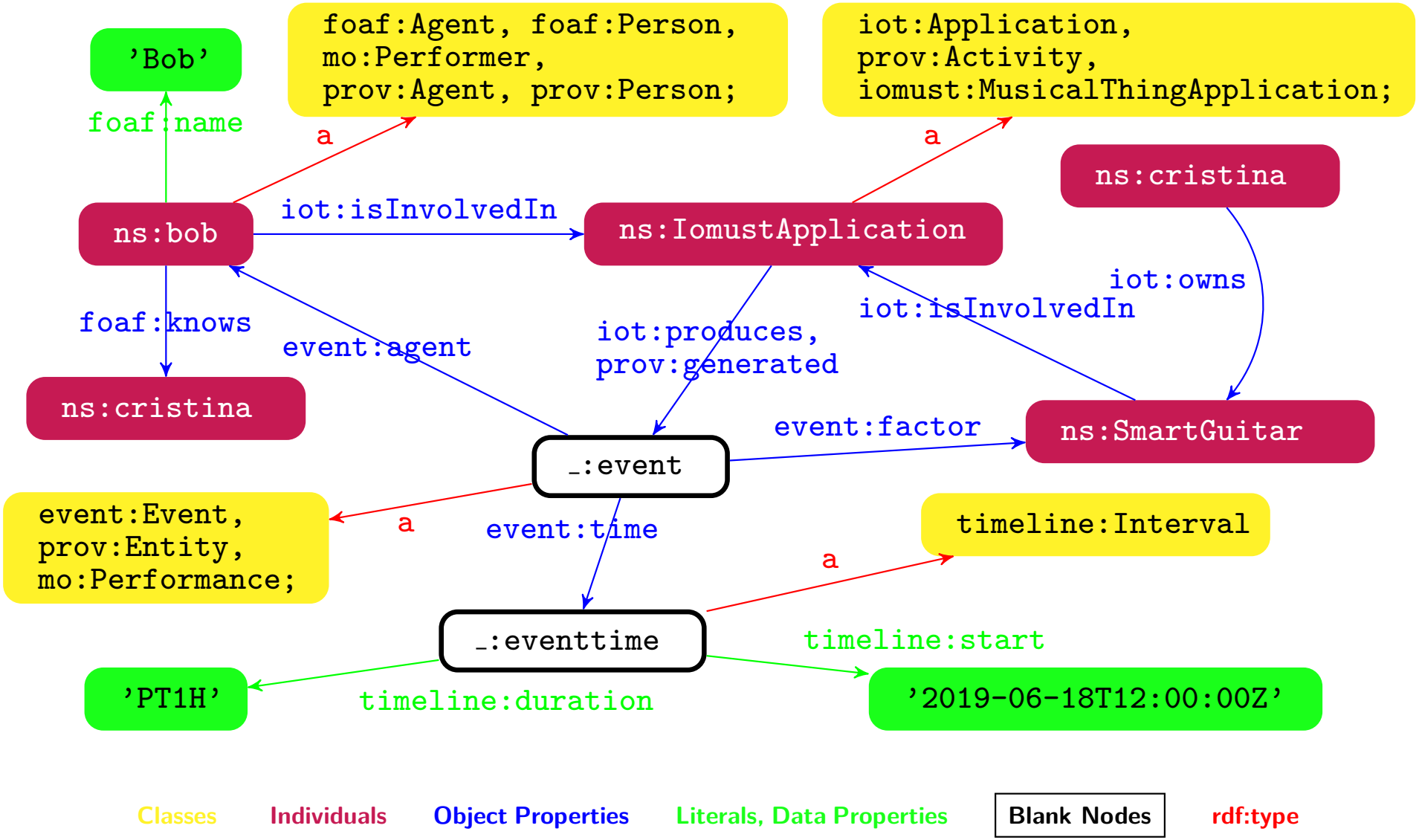


IoMusT Ontology example II

How do we describe an IoMusT application:

1. A music performance made by someone;
2. Using a Smart guitar belonging to someone;
3. Starting at a specific time;
4. Having a specific duration;

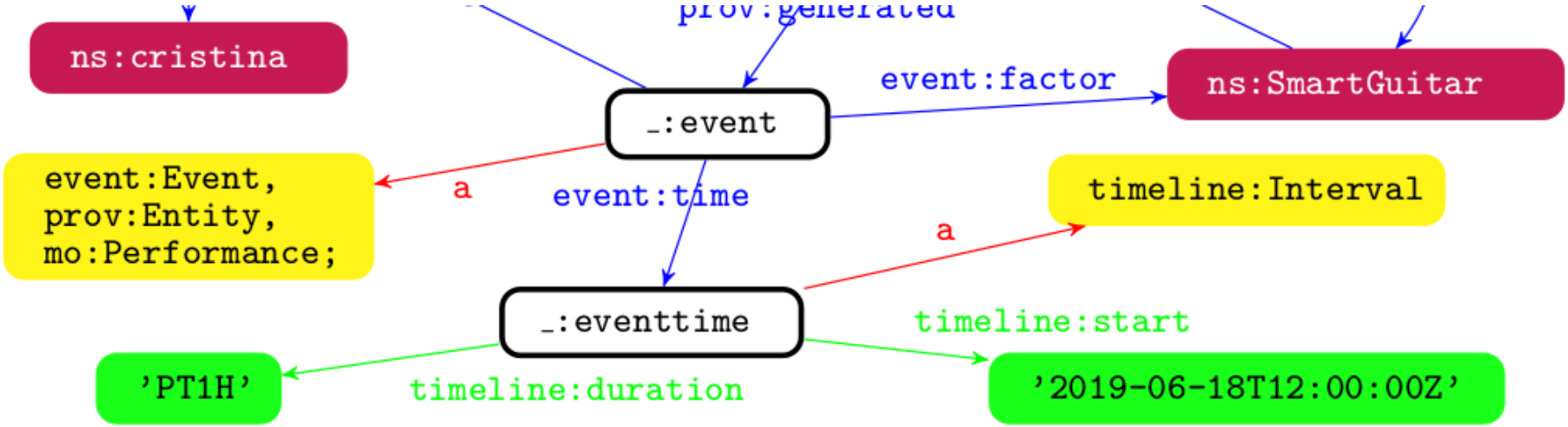
IoMusT Ontology example II



https://fr4ncidir.github.io/IoMusT/

Going further?

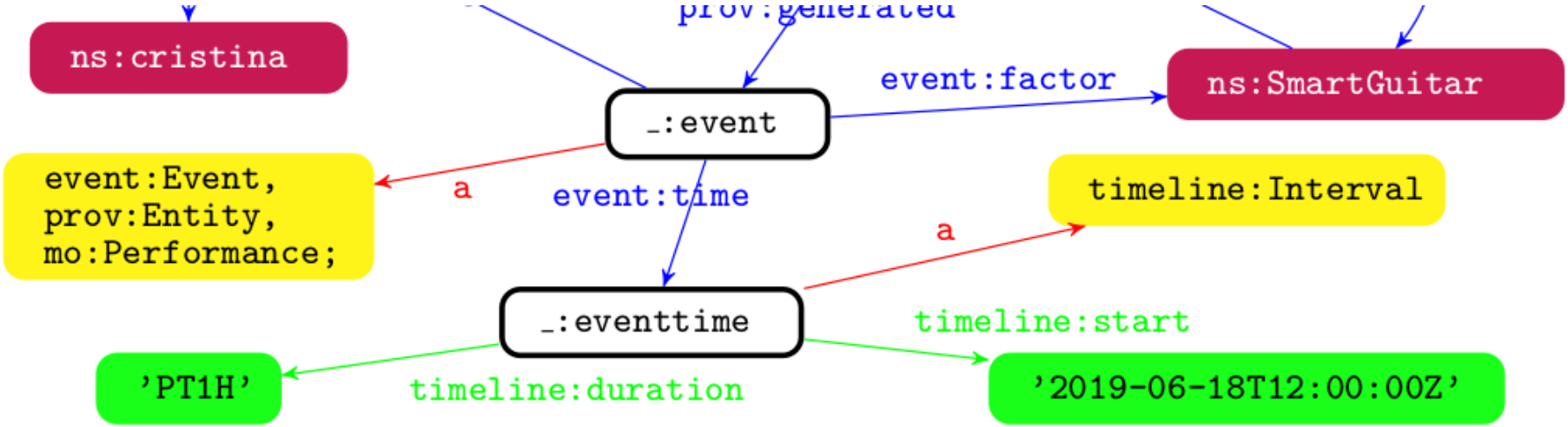
Semantic Web of Things



Any issues?

Going further?

Semantic Web of Things

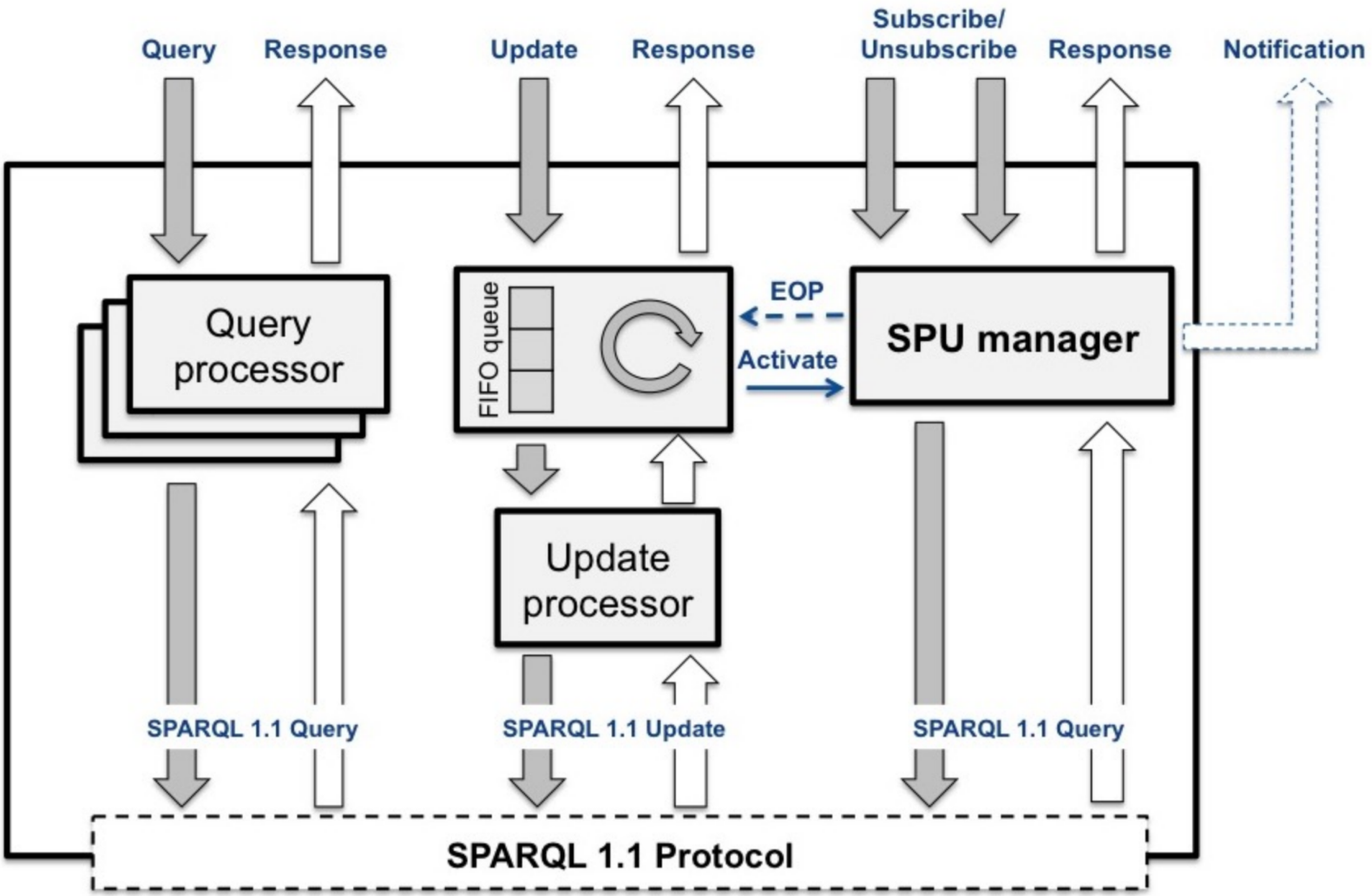


Any issues?

Once we setup the information (e.g. *interval* and *duration*), how do we keep up with evolving environments? What if a Smart Violin joins the performance?

SPARQL Event Processing Architecture

Prerequisite



Ref. [4]

Interacting through SEPA

SEPA Query

Client: `HTTP GET` with SPARQL 1.1 Query payload;
SEPA engine: returns JSON content with SPARQL variable bindings;

Interacting through SEPA

SEPA Query

Client: `HTTP GET` with SPARQL 1.1 Query payload;
SEPA engine: returns JSON content with SPARQL variable bindings;

SEPA Update

Client: `HTTP POST` with SPARQL 1.1 Update payload;
SEPA engine: returns update 'success' or 'failure';

Interacting through SEPA

SEPA Query

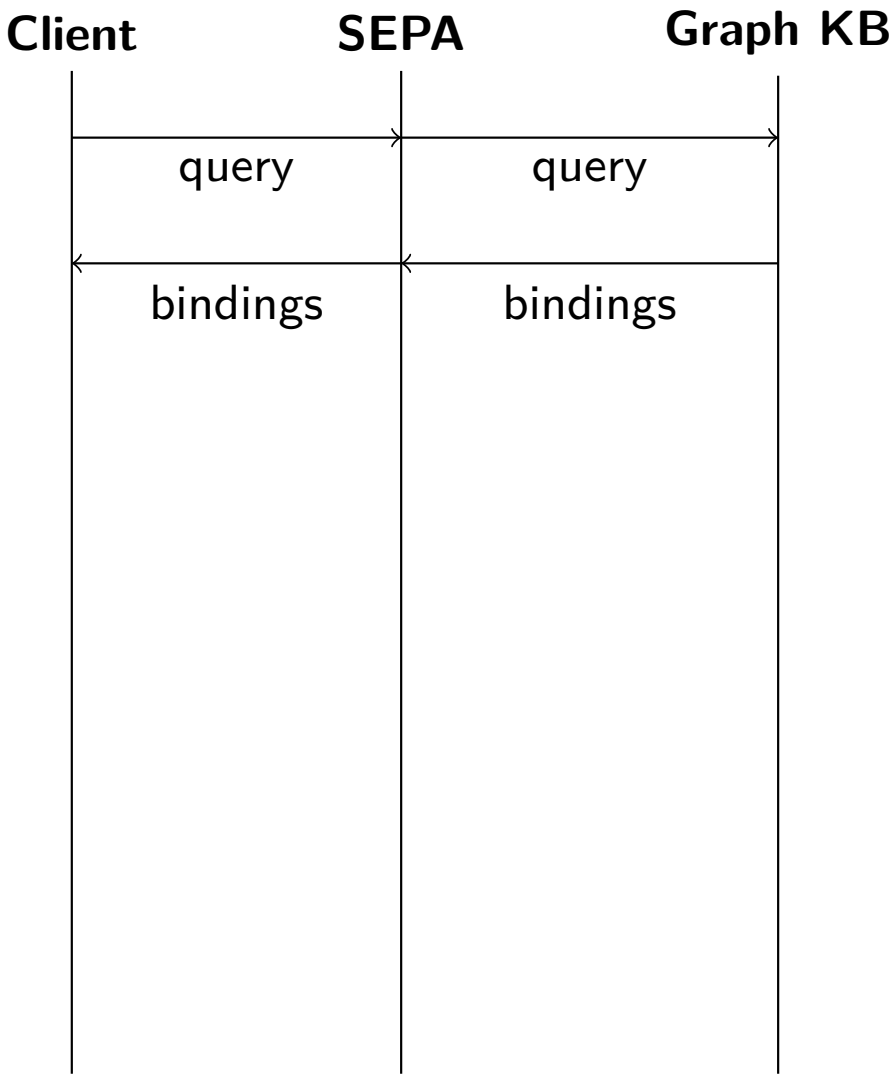
Client: `HTTP GET` with SPARQL 1.1 Query payload;
SEPA engine: returns JSON content with SPARQL variable bindings;

SEPA Update

Client: `HTTP POST` with SPARQL 1.1 Update payload;
SEPA engine: returns update 'success' or 'failure';

SEPA Subscription

Client: requests `WebSocket` opening, transmits SPARQL 1.1 Query;
SEPA engine: activates the subscription and transmits the first notification as JSON contents with query variable bindings.
From then on, JSON contents with `added` and `removed` bindings;



HTTP GET:

SPARQL 1.1 Query payload

Response payload:

JSON with SPARQL variable binding

HTTP POST:

SPARQL 1.1 Update payload

Response:

HTTP response code

WebSocket open:

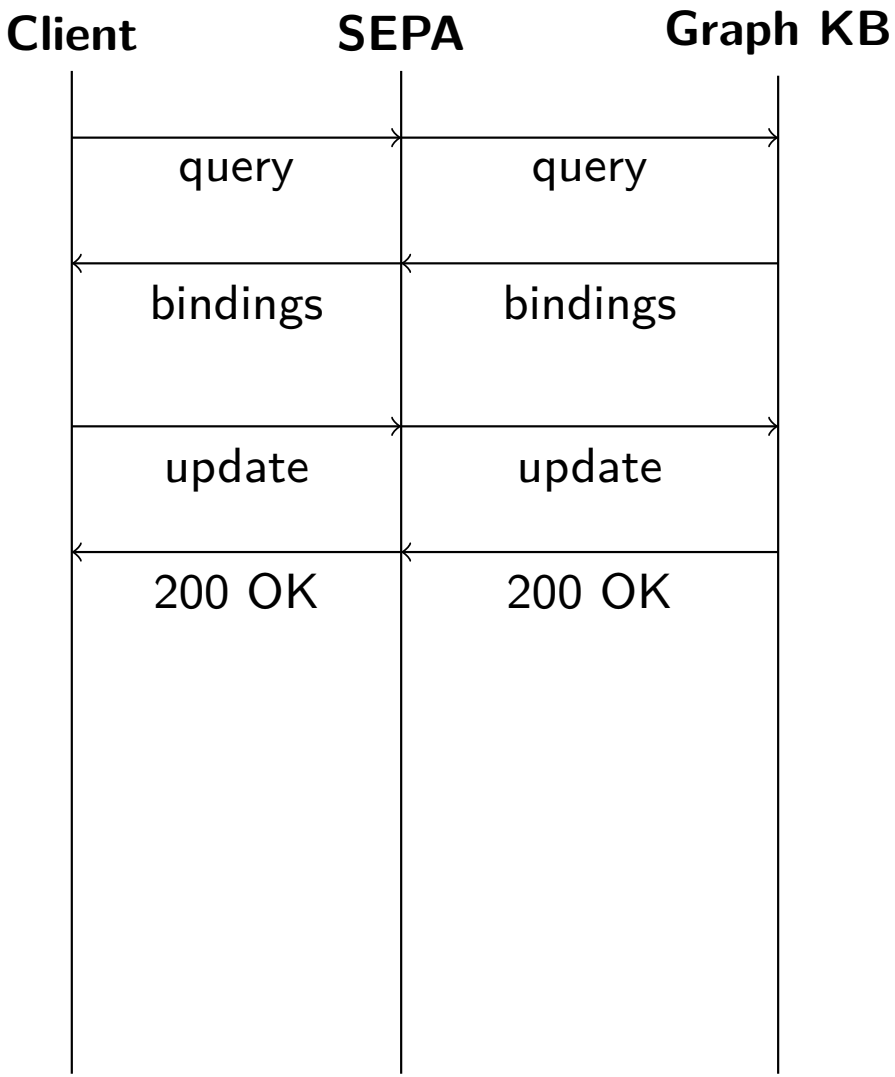
Engine SPU setup, 1st notification generation

1st notification:

Query-like JSON with SPARQL variable binding

Additional notifications:

JSON with Δ (added and removed) bindings only



HTTP GET:

SPARQL 1.1 Query payload

Response payload:

JSON with SPARQL variable binding

HTTP POST:

SPARQL 1.1 Update payload

Response:

HTTP response code

WebSocket open:

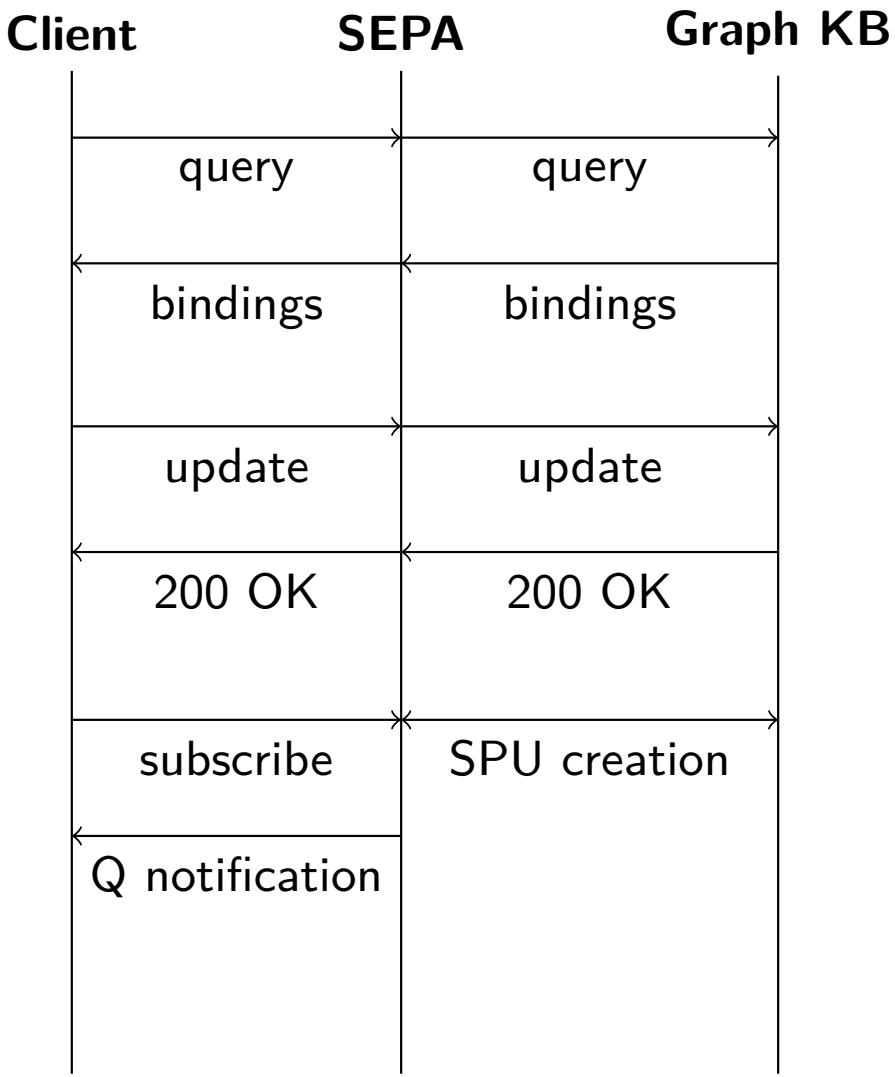
Engine SPU setup, 1st notification generation

1st notification:

Query-like JSON with SPARQL variable binding

Additional notifications:

JSON with Δ (added and removed) bindings only



HTTP GET:

SPARQL 1.1 Query payload

Response payload:

JSON with SPARQL variable binding

HTTP POST:

SPARQL 1.1 Update payload

Response:

HTTP response code

WebSocket open:

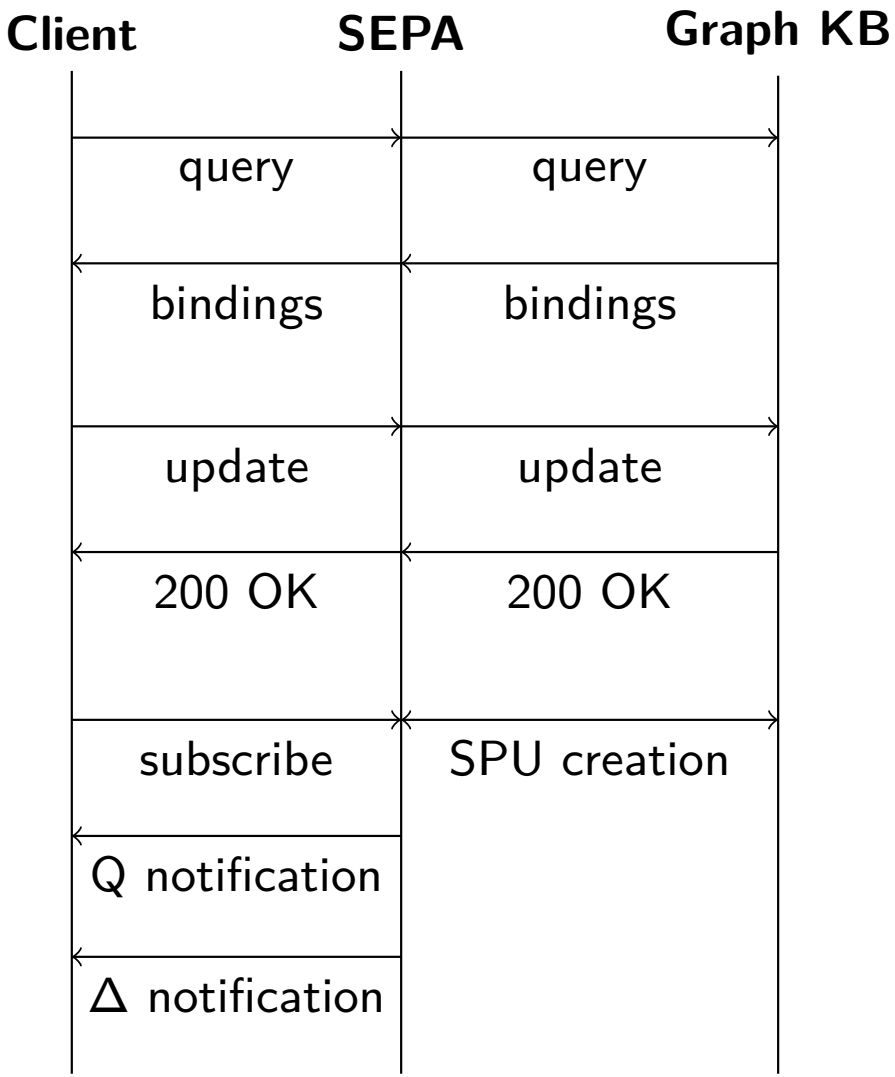
Engine SPU setup, 1st notification generation

1st notification:

Query-like JSON with SPARQL variable binding

Additional notifications:

JSON with Δ (added and removed) bindings only



HTTP GET:

SPARQL 1.1 Query payload

Response payload:

JSON with SPARQL variable binding

HTTP POST:

SPARQL 1.1 Update payload

Response:

HTTP response code

WebSocket open:

Engine SPU setup, 1st notification generation

1st notification:

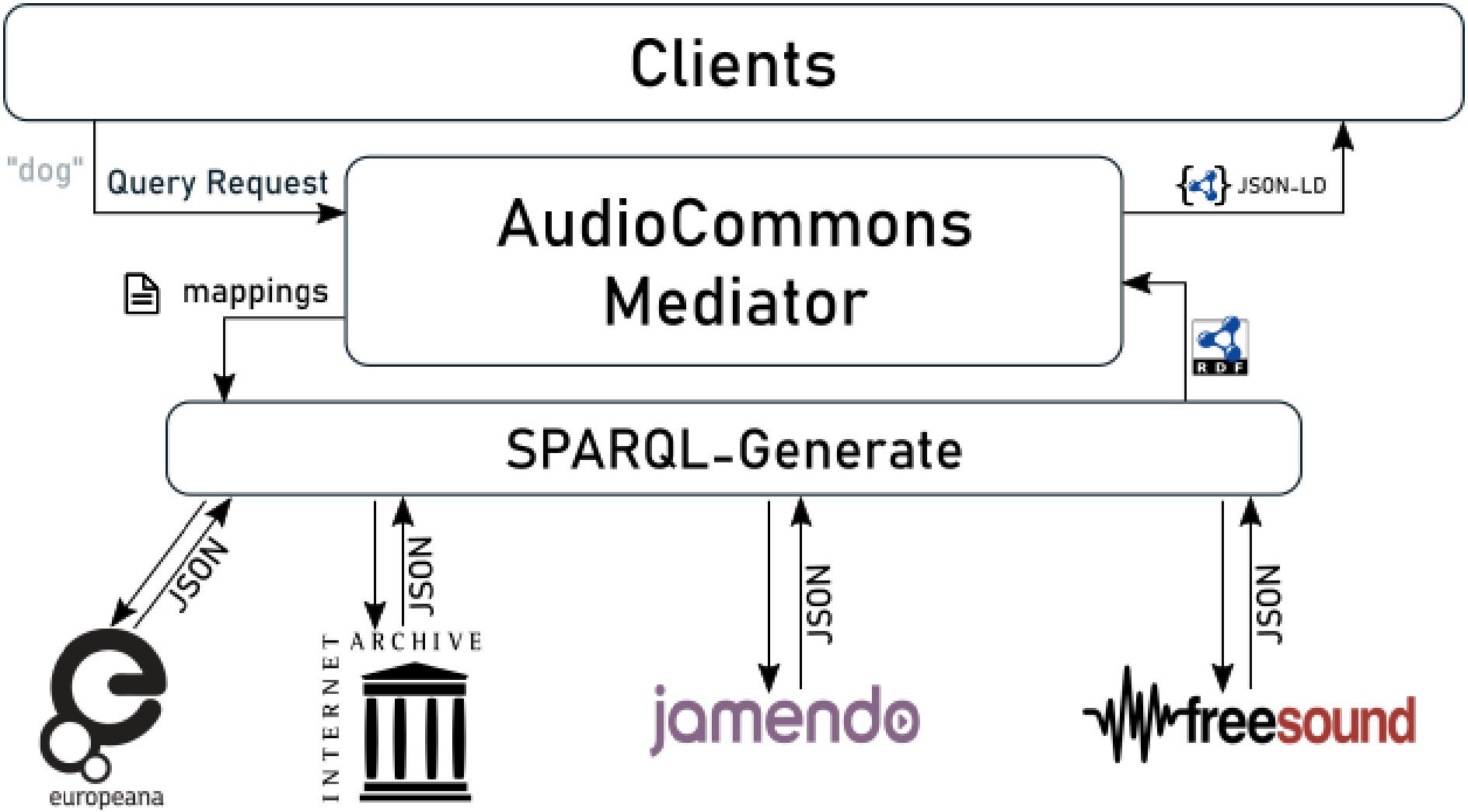
Query-like JSON with SPARQL variable binding

Additional notifications:

JSON with Δ (added and removed) bindings only

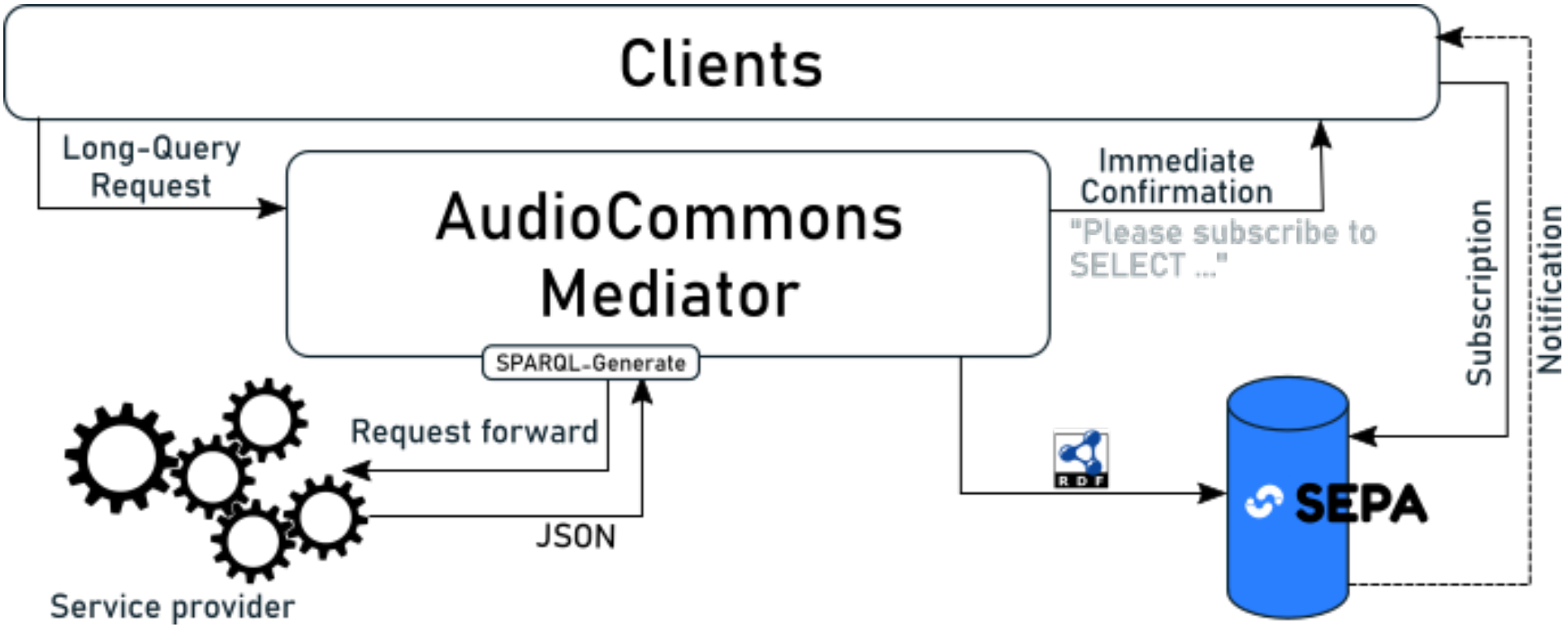
AudioCommons project I

SEPA usage example



AudioCommons project II

SEPA usage example



Ref. [5, 6, 7]

Web of Things I

Prerequisite

The Web of Things was introduced approximately in 2010 [8]. The fragmentation of IoT was addressed considering that

- the hardware available today is cheap and powerful;
- the World Wide Web stack, including HTTP protocol and JSON format for data exchange are broadly used;
- the RESTful design can be applied also to IoT systems and giving a broader meaning to the term ‘resource’

Web of Things II

Prerequisite

As discussed, the W3C provided its own definition of Thing, highlighting the fact that **they have URIs**.

The W3C WoT Working Group was created in this context to develop an interaction model and a common vocabulary to identify the Thing capabilities in the environment.

This Thesis work takes inspiration from the W3C drafts, and combines it with the aforementioned SEPA architecture [9].

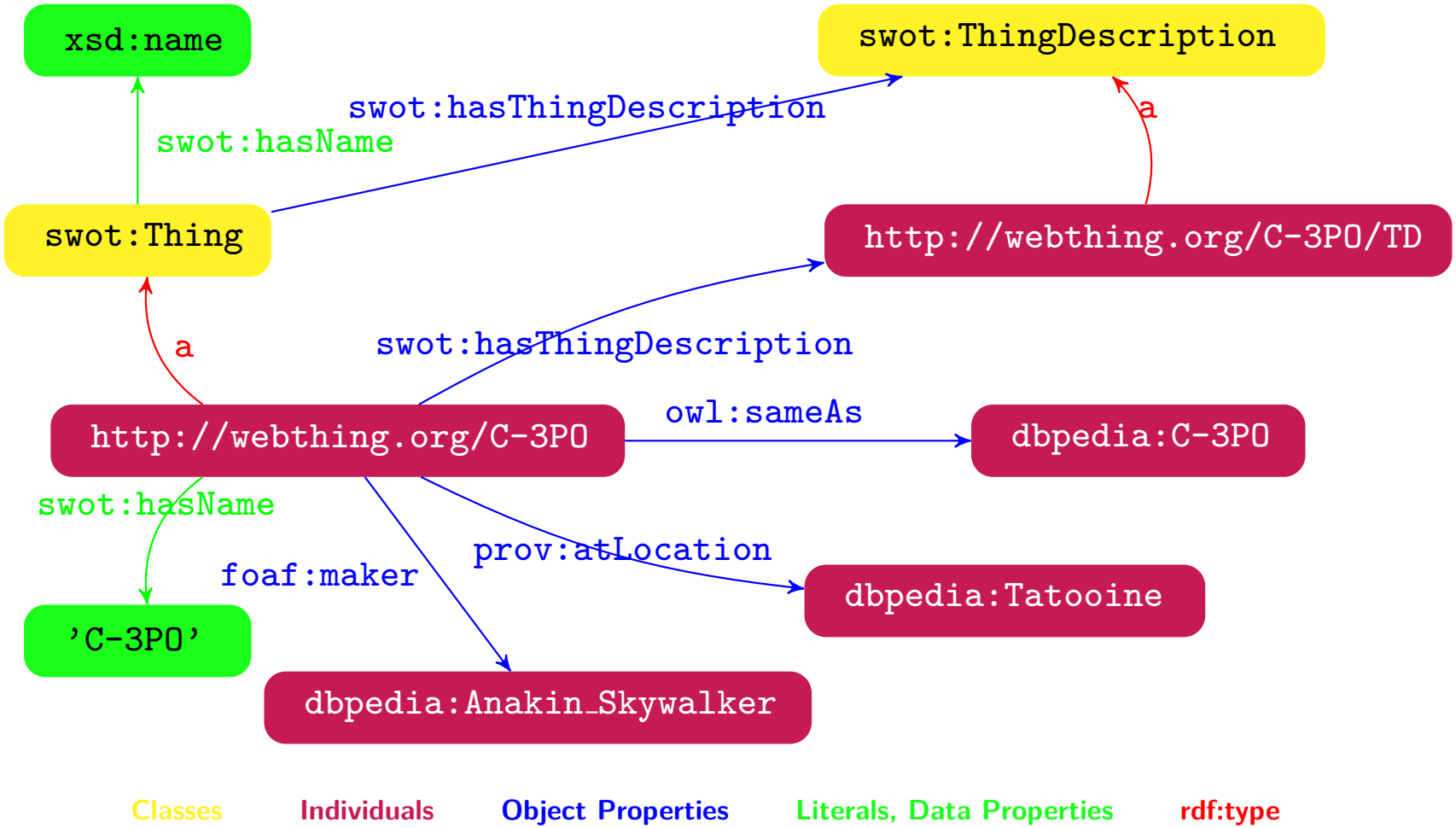
Comparison: W3C WoT vs SWoT ontology tools

W3C WoT	SWoT (this work)
Devices have their **Thing Descriptions** (TD). They provide it as small web servers, in JSON-LD;	Devices post their TDs as a graph into SEPA;
Clients still have to discover the devices by their own means, or discovery repos have to be implemented;	SEPA provides natively semantic discovery over the RDF graph of TDs;
Direct interaction after interpreting the TD;	SEPA-mediated interaction according to the SWOT dynamic ontology
Reduced data model;	Extended data model and protocol;
No shared context, since TDs are separate universed;	SEPA may contain a shared context subgraph, depending on applications;

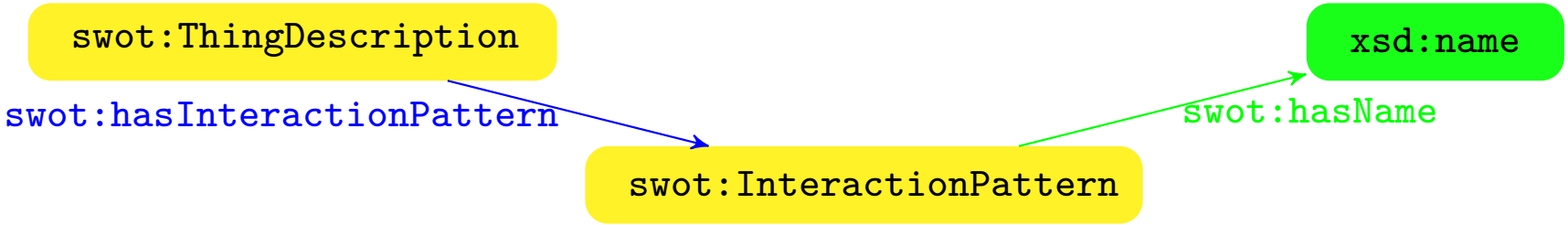
SWOT (static) Ontology

Classes Individuals Object Properties Literals, Data Properties rdf:type

SWOT (static) Ontology

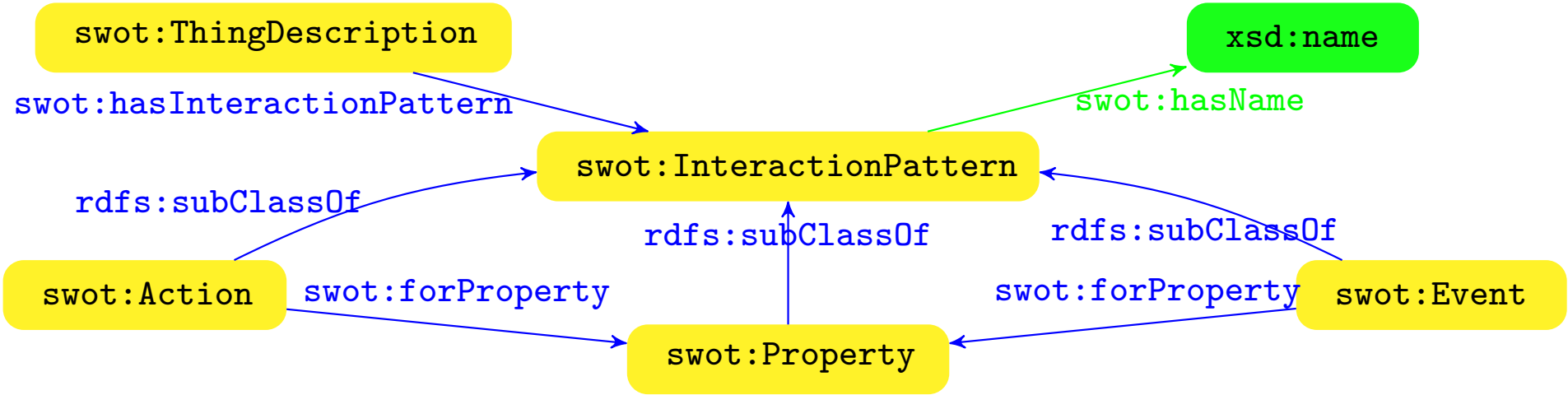


SWOT (static) Ontology



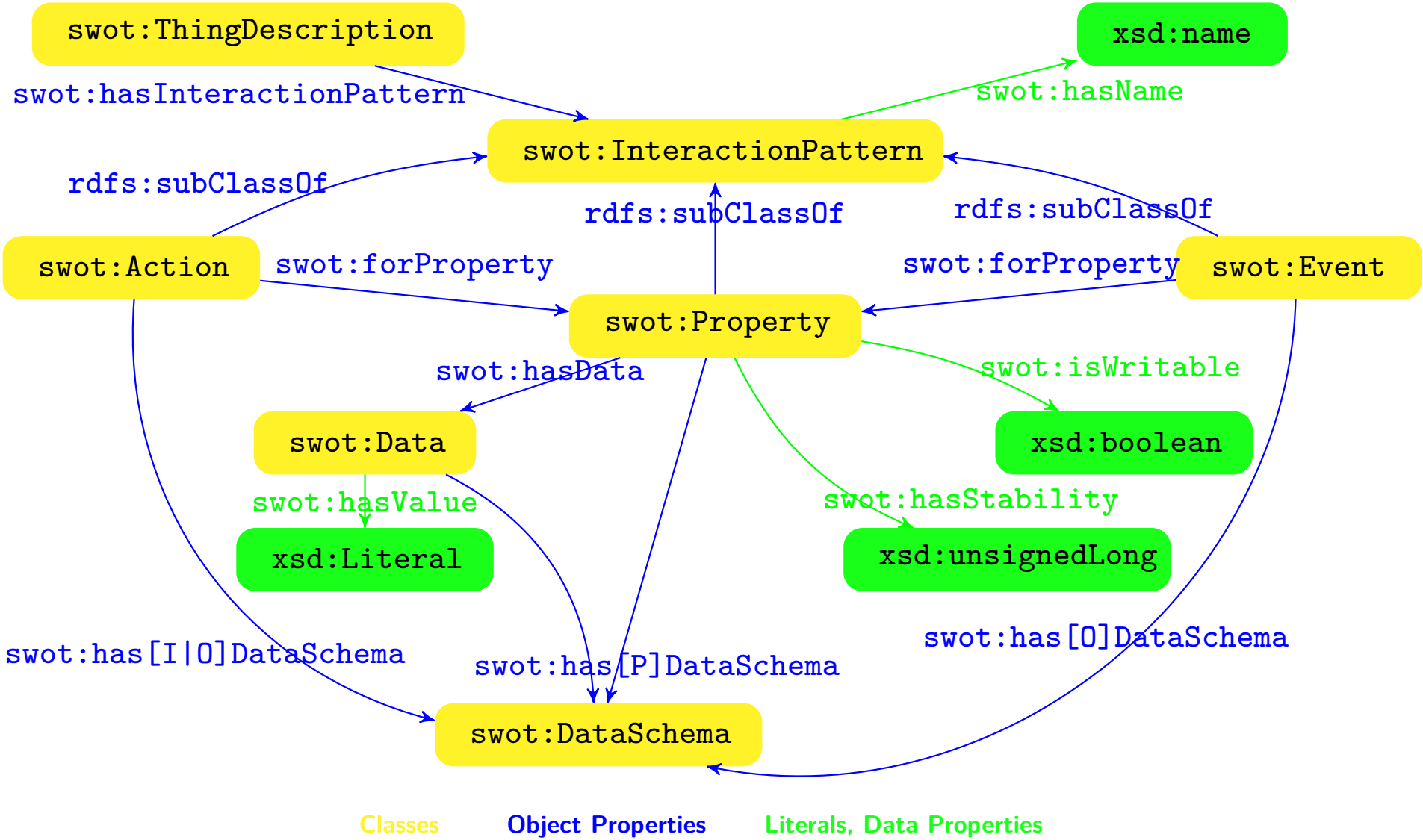
Classes Object Properties Literals, Data Properties

SWOT (static) Ontology

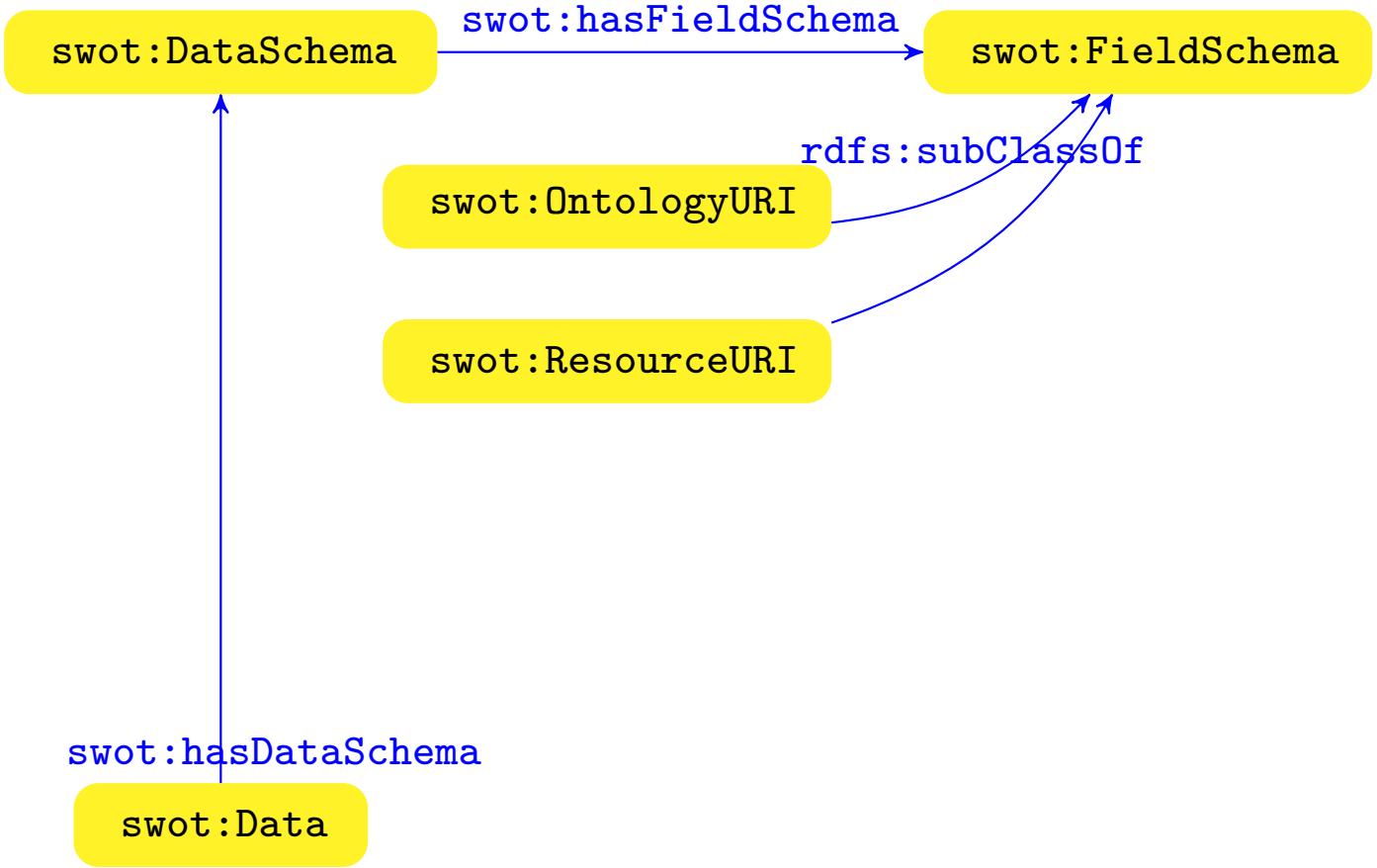


Classes Object Properties Literals, Data Properties

SWOT (static) Ontology



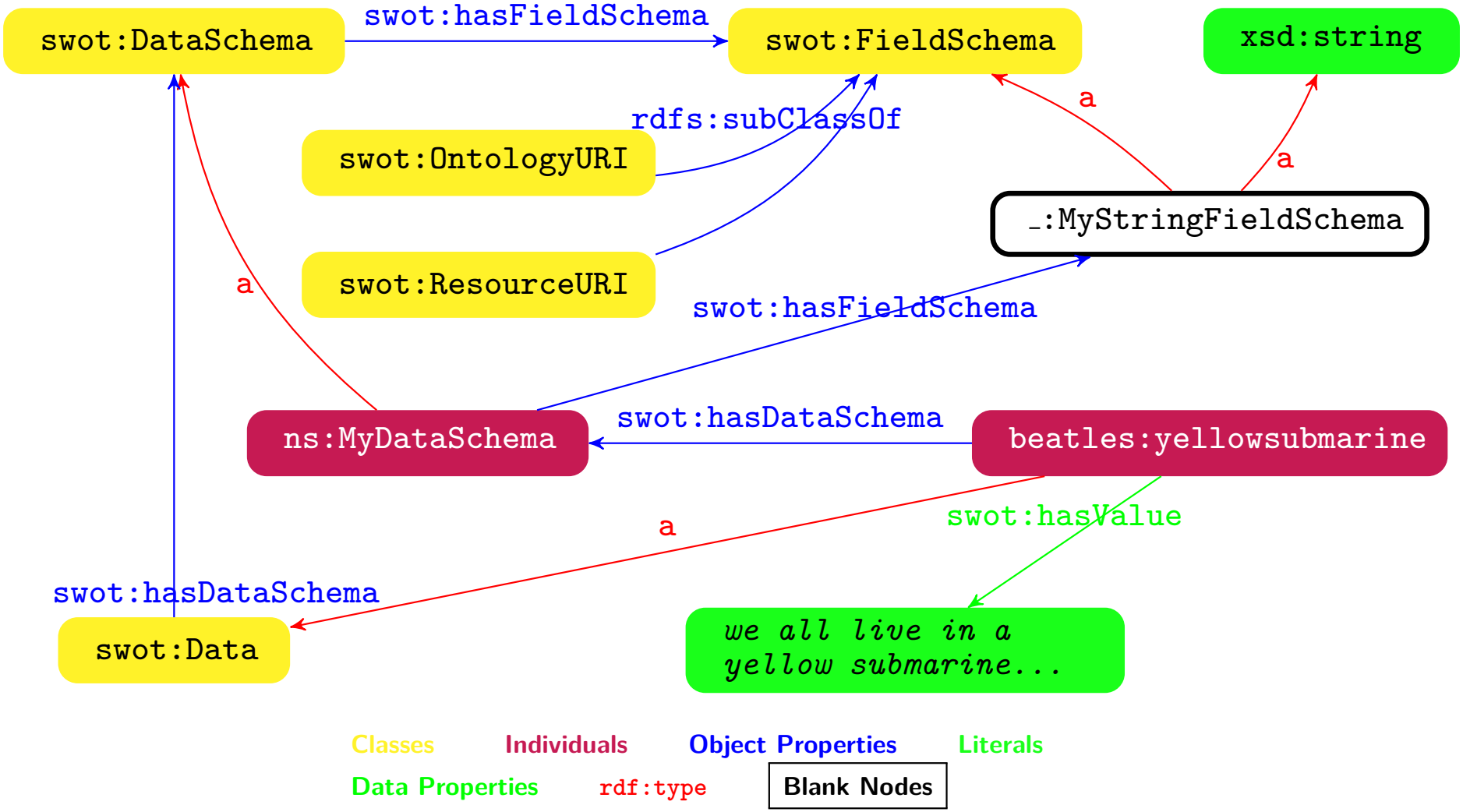
SWOT (static) Ontology



Classes Individuals Object Properties Literals

Data Properties rdf:type Blank Nodes

SWOT (static) Ontology



SWOT (dynamic) Ontology

Triggering an Action Request

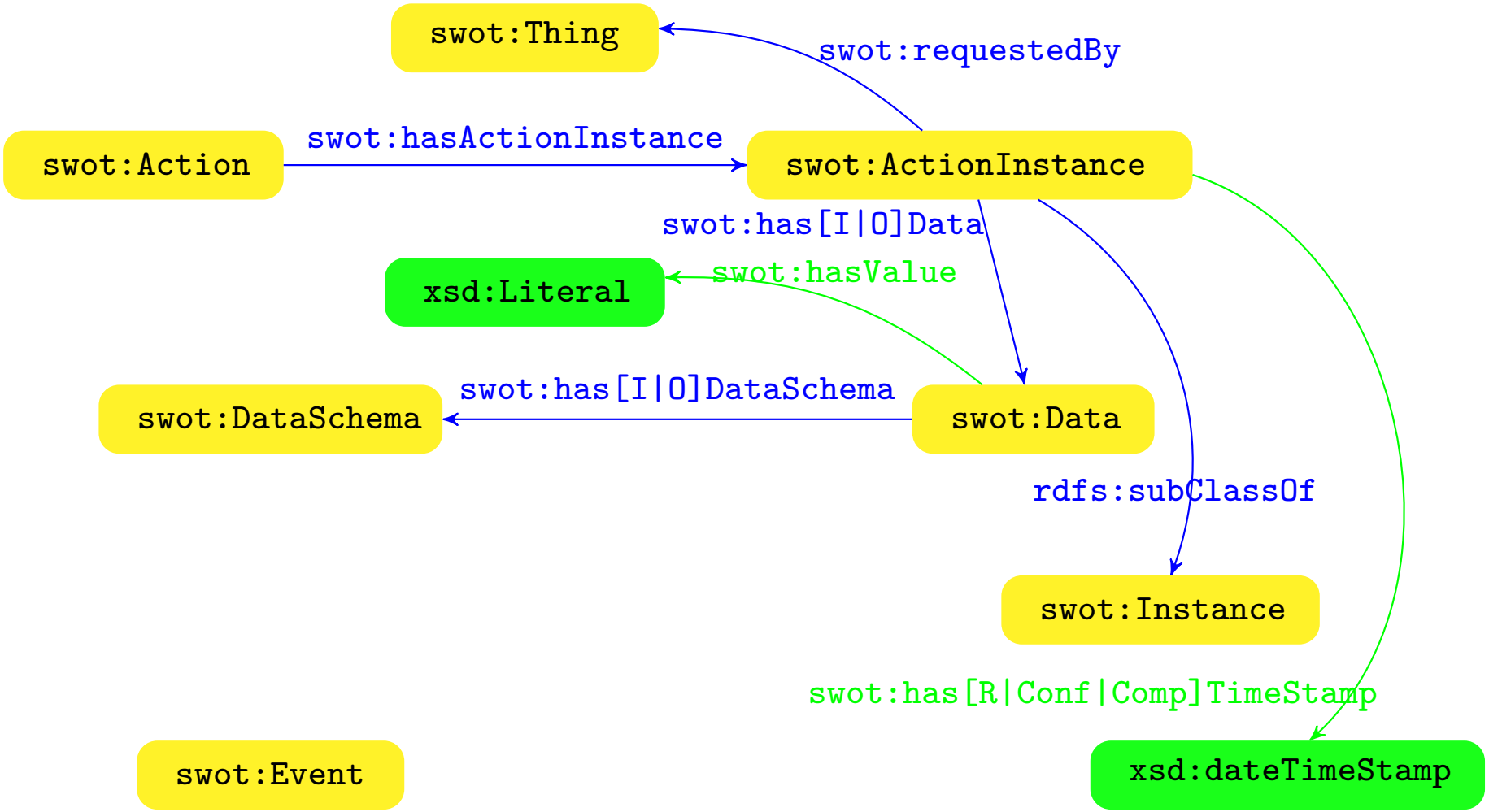
`swot:Action`

`swot:Event`

Classes Object Properties Literals, Data Properties

SWOT (dynamic) Ontology

Triggering an Action Request



SWOT (dynamic) Ontology

Triggering an Event Notification

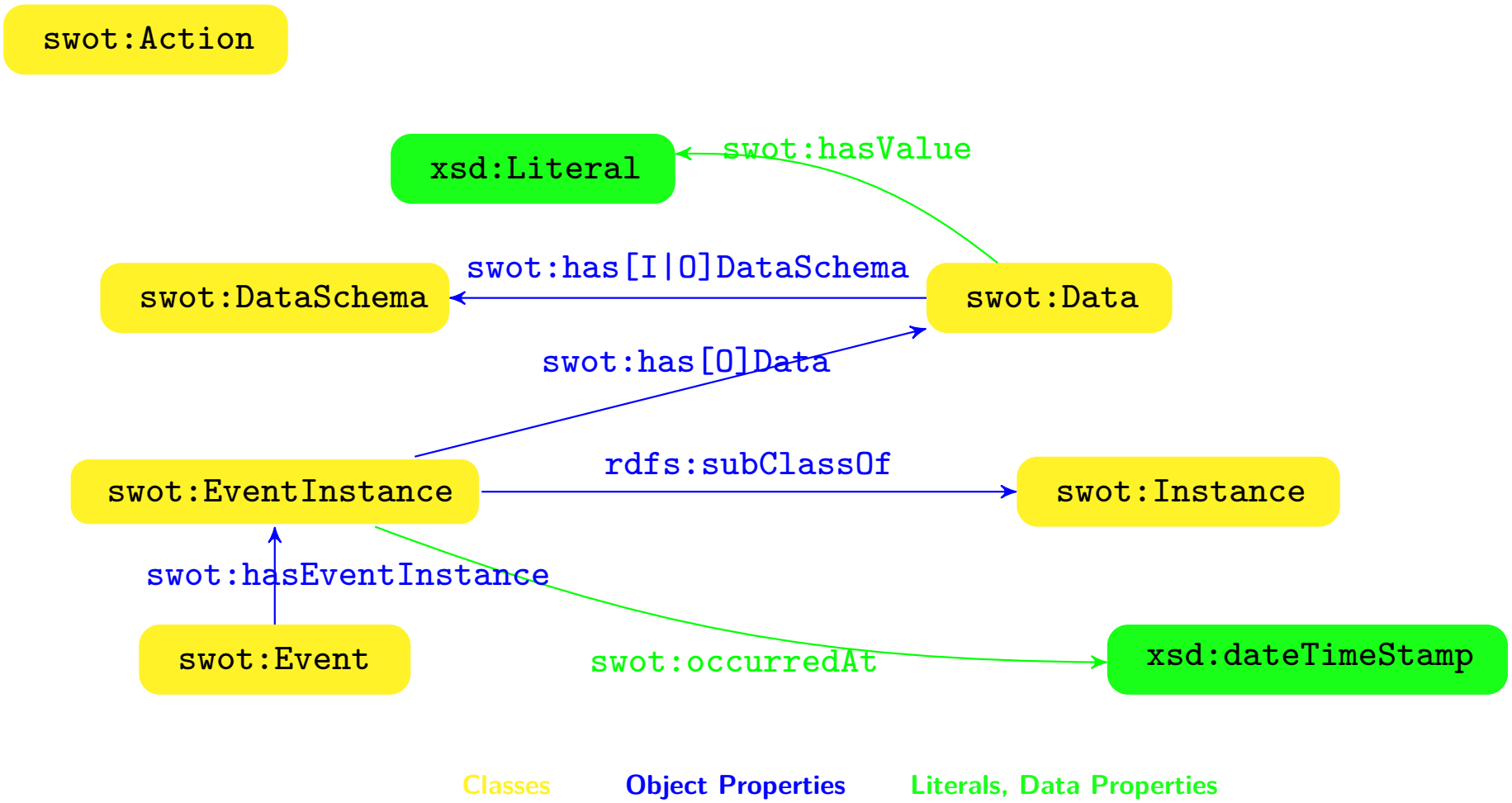
swot:Action

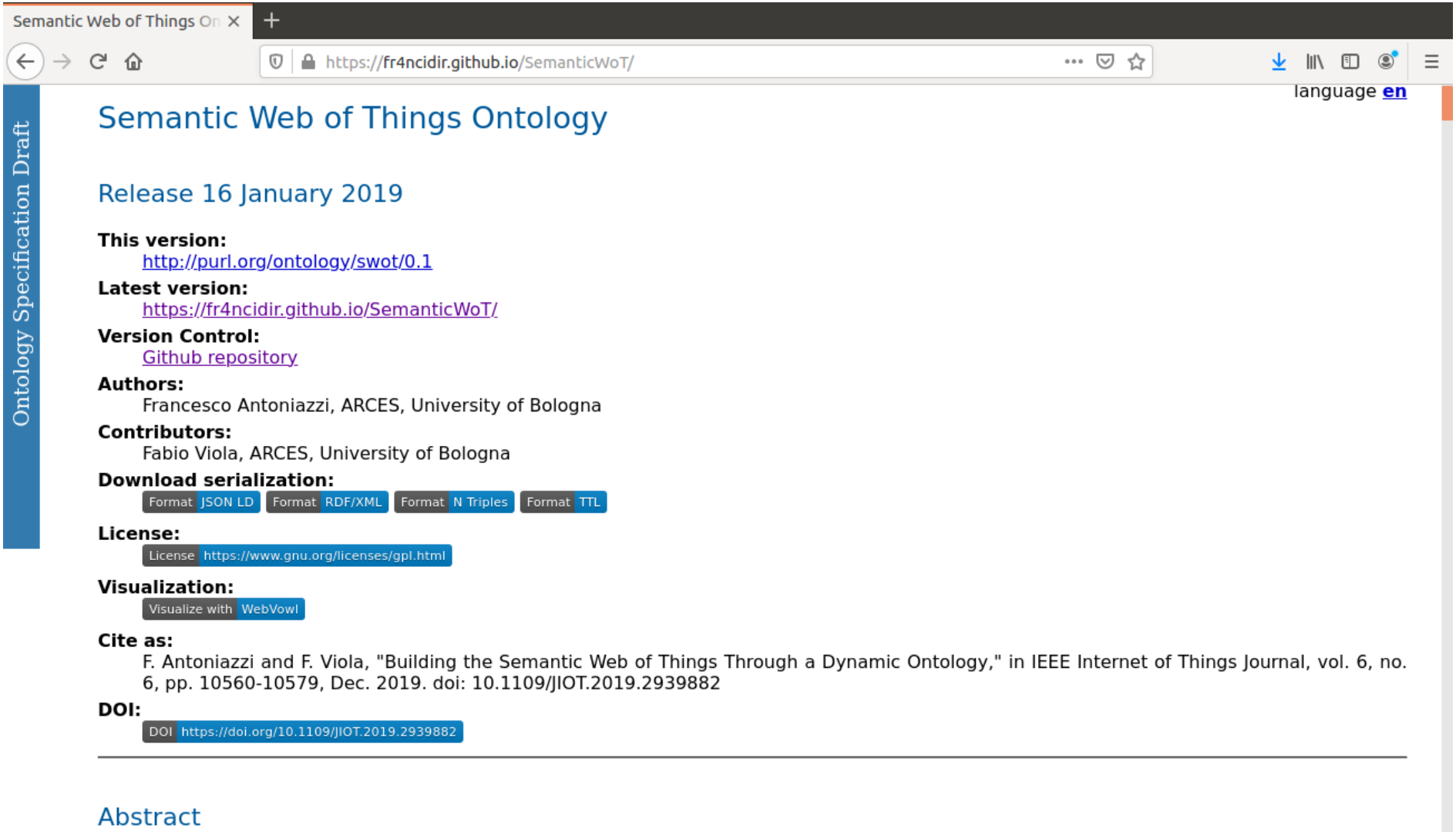
swot:Event

Classes Object Properties Literals, Data Properties

SWOT (dynamic) Ontology

Triggering an Event Notification





https://fr4ncidir.github.io/SemanticWoT/

Programming the SWoT

Cocktail Framework

A Python3 framework was developed to fully exploit the SWOT ontology presented, based on a SEPA engine:

1. semantic TD definition and posting;

All this by using SEPA's engine with SPARQL 1.1 Queries, Updates and Discoveries.

Cocktail Framework

A Python3 framework was developed to fully exploit the SWOT ontology presented, based on a SEPA engine:

1. semantic TD definition and posting;
2. event subscription and notification;

All this by using SEPA's engine with SPARQL 1.1 Queries, Updates and Discoveries.

Cocktail Framework

A Python3 framework was developed to fully exploit the SWOT ontology presented, based on a SEPA engine:

1. semantic TD definition and posting;
2. event subscription and notification;
3. action request and execution;

All this by using SEPA's engine with SPARQL 1.1 Queries, Updates and Discoveries.

Cocktail Framework

A Python3 framework was developed to fully exploit the SWOT ontology presented, based on a SEPA engine:

1. semantic TD definition and posting;
2. event subscription and notification;
3. action request and execution;
4. full semantic discovery;

All this by using SEPA's engine with SPARQL 1.1 Queries, Updates and Discoveries.

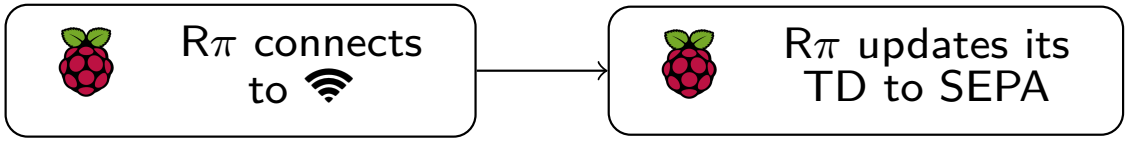
SWoT Application Workflow

Basic example with two devices

Rπ connects to

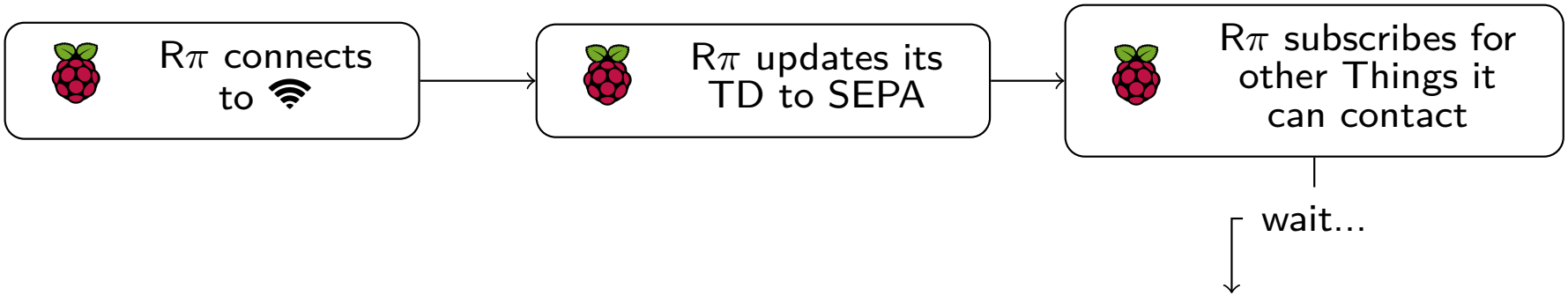
SWoT Application Workflow

Basic example with two devices



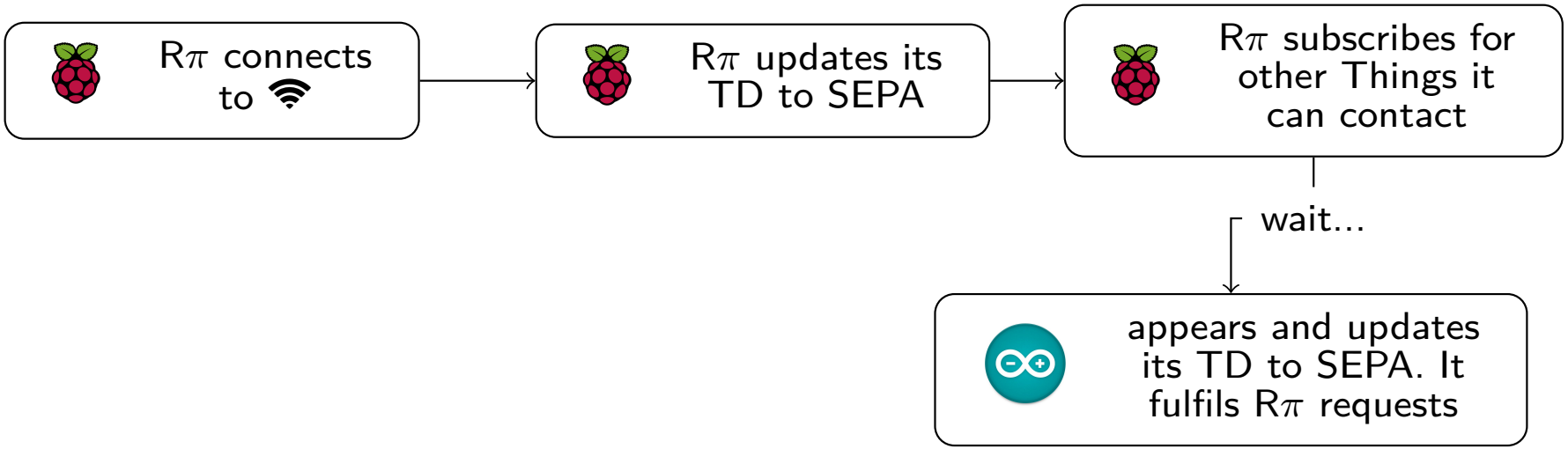
SWoT Application Workflow

Basic example with two devices



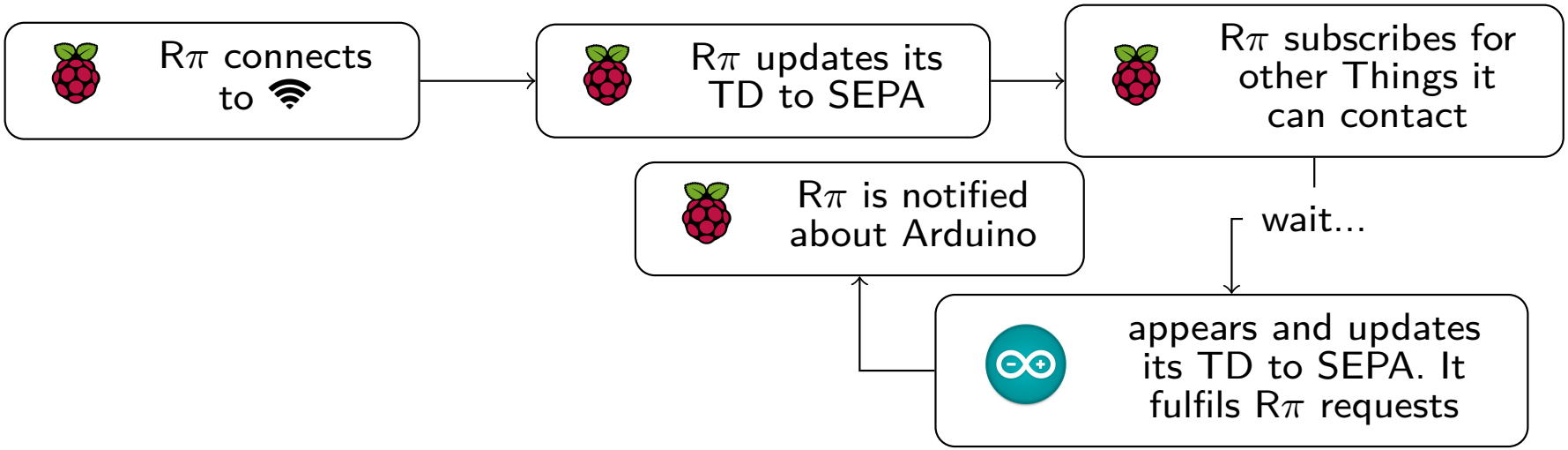
SWoT Application Workflow

Basic example with two devices



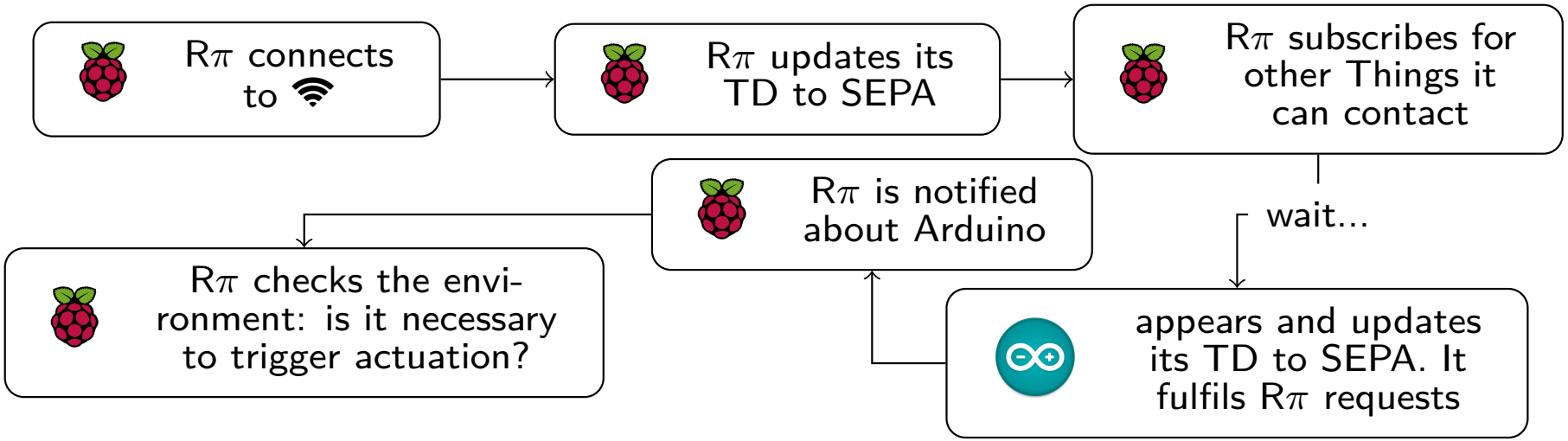
SWoT Application Workflow

Basic example with two devices



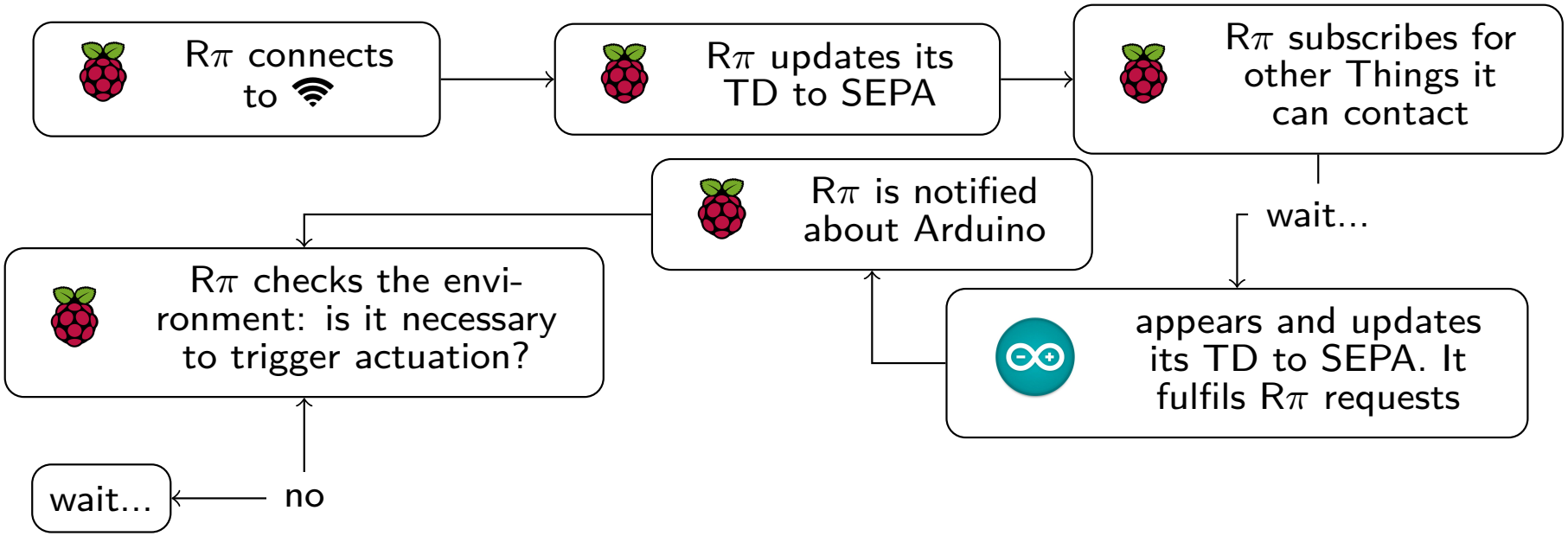
SWoT Application Workflow

Basic example with two devices



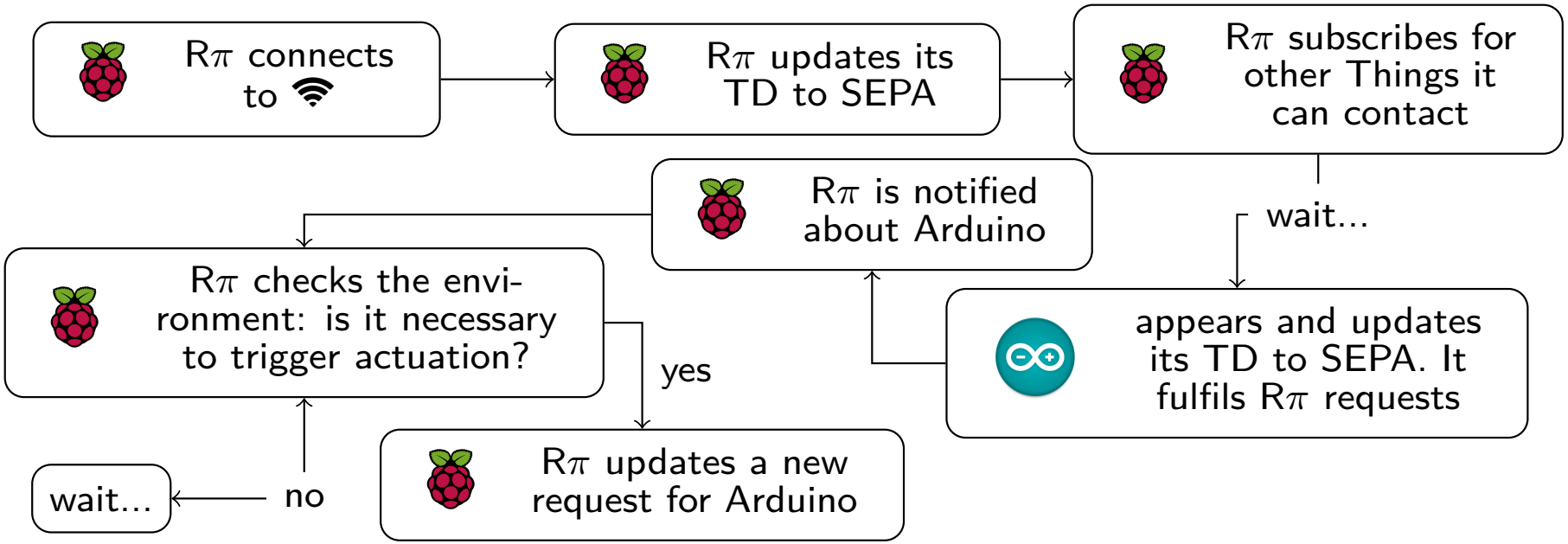
SWoT Application Workflow

Basic example with two devices



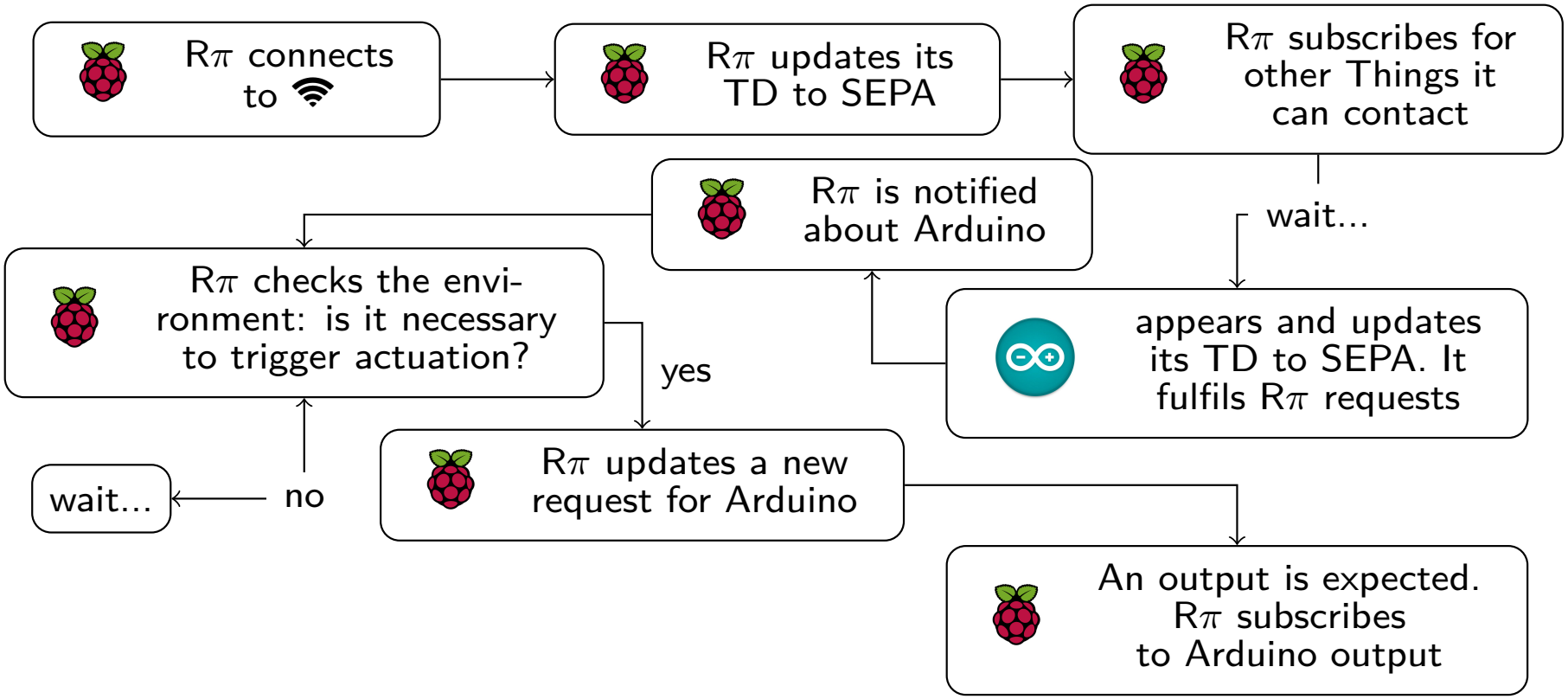
SWoT Application Workflow

Basic example with two devices



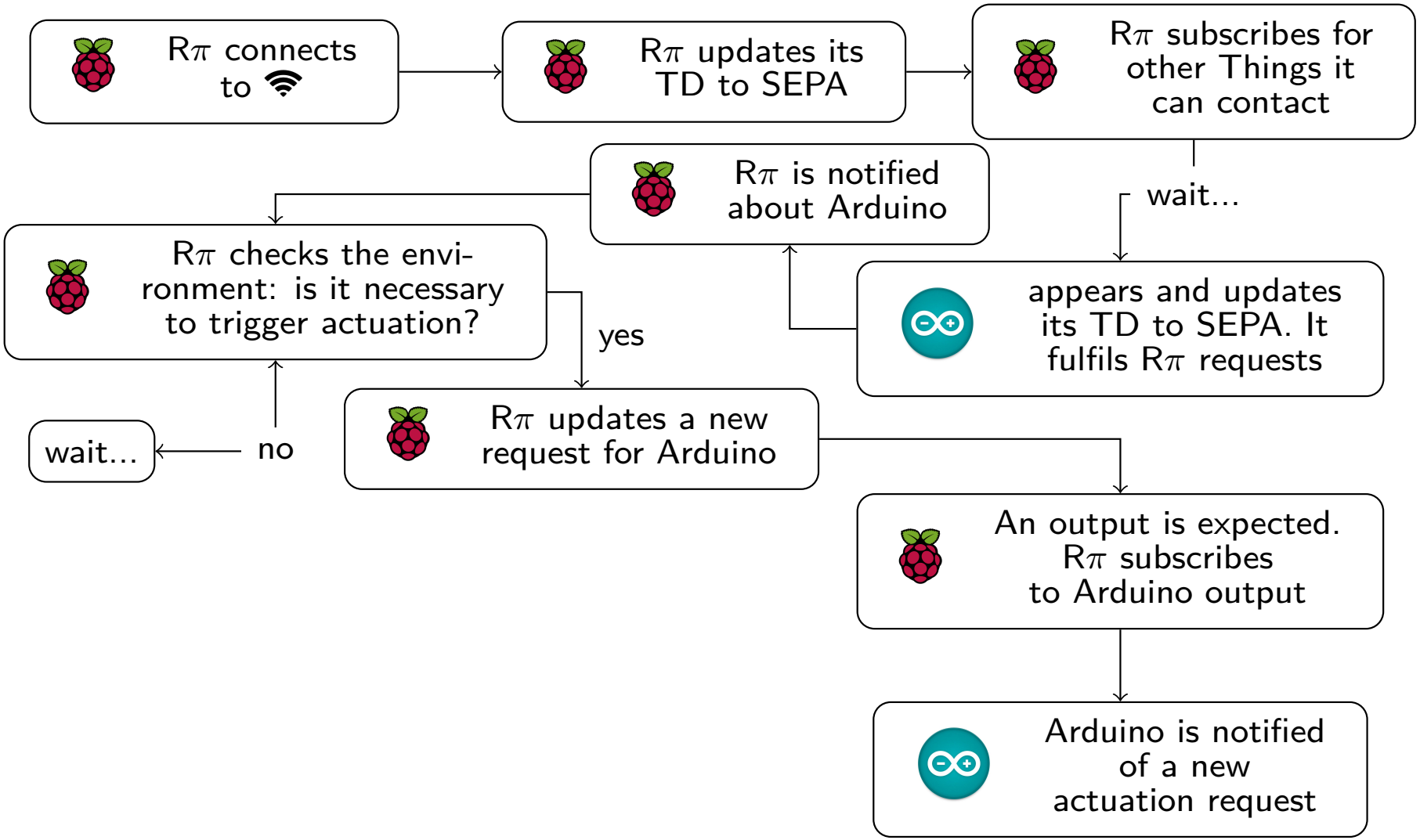
SWoT Application Workflow

Basic example with two devices



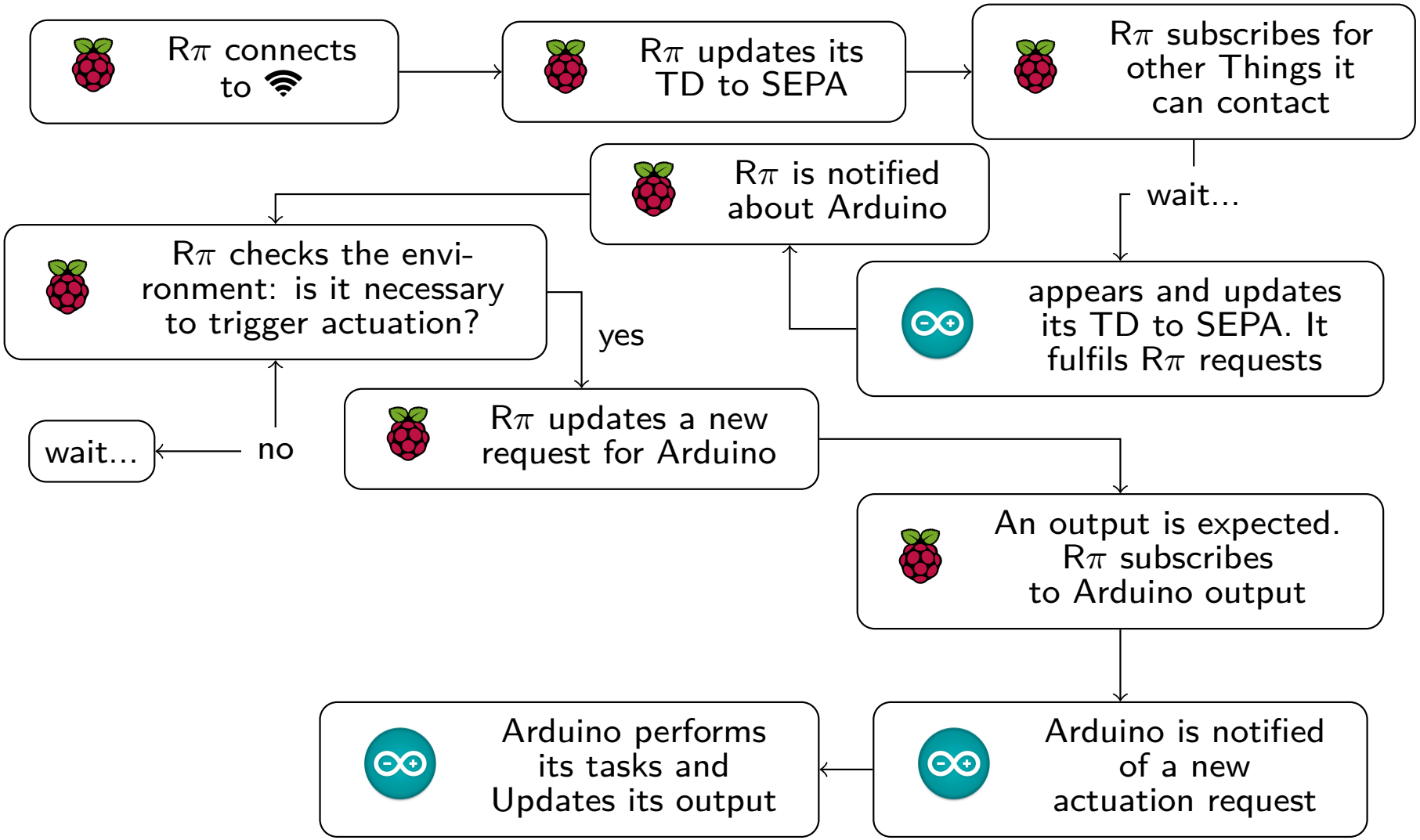
SWoT Application Workflow

Basic example with two devices



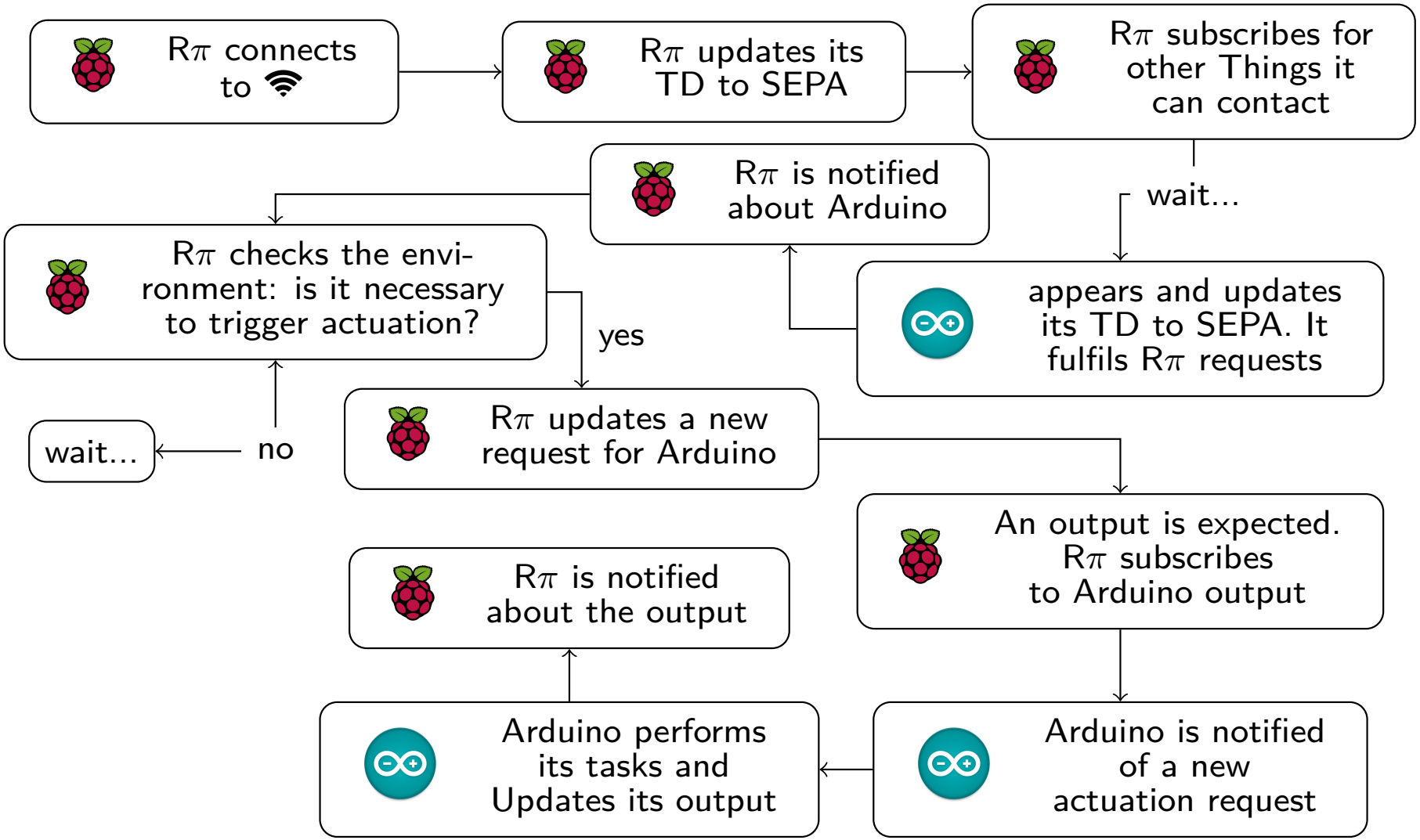
SWoT Application Workflow

Basic example with two devices



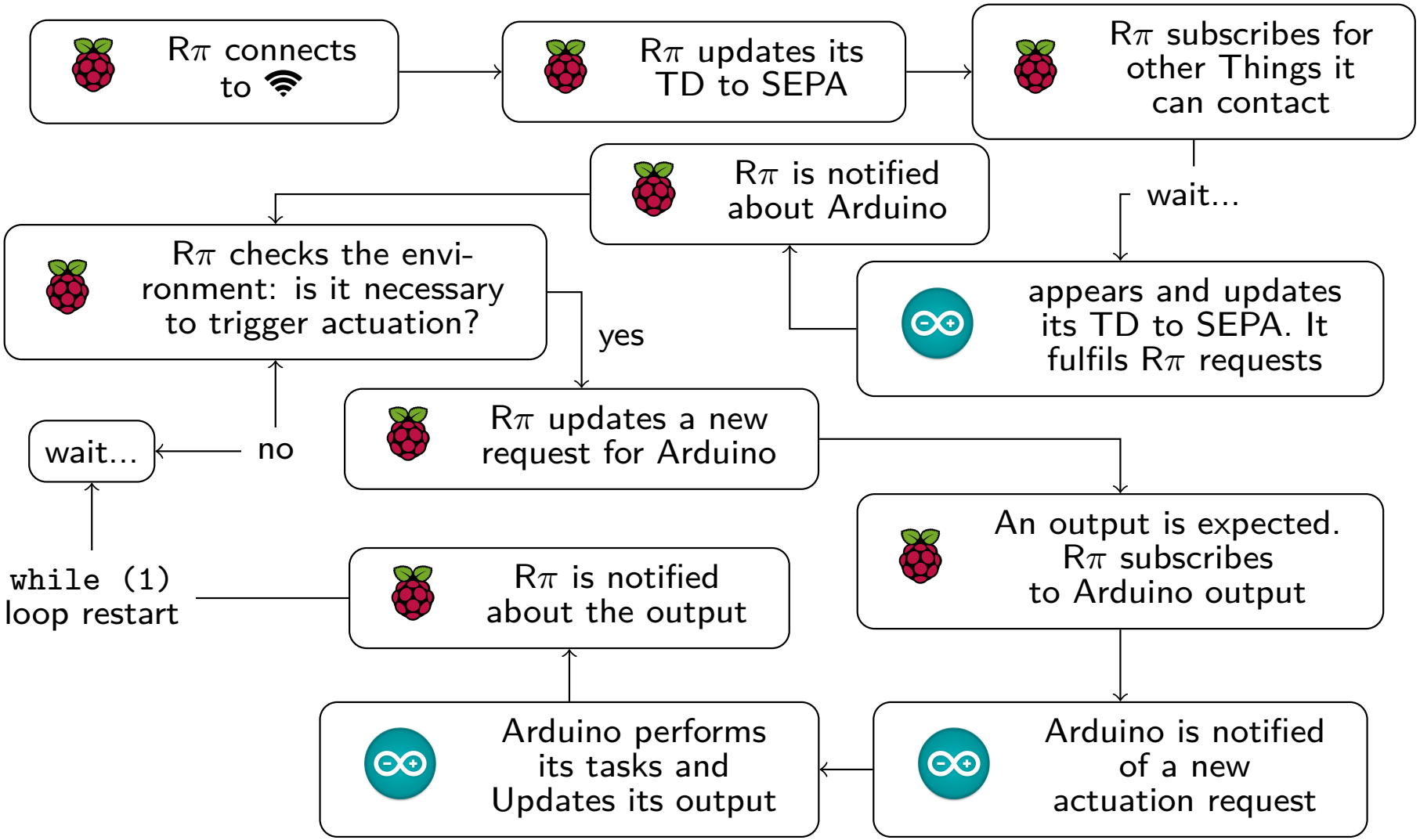
SWoT Application Workflow

Basic example with two devices



SWoT Application Workflow

Basic example with two devices



Future Research...

Future Research...

- From the Internet of Musical Things to the (Semantic) Web of Musical Things;

  In collaboration with Università di Trento

Future Research...

- From the Internet of Musical Things to the (Semantic) Web of Musical Things;

  In collaboration with Università di Trento

- enhance SPARQL-Generate;

  PostDoc at École des Mines de St-Étienne

Future Research...

- From the Internet of Musical Things to the (Semantic) Web of Musical Things;
  In collaboration with Università di Trento
- enhance SPARQL-Generate;
  PostDoc at École des Mines de St-Étienne
- Apply the workflow previously presented to create Industry 4.0 environments; *École des Mines de St-Étienne: IT'M Factory*

Future Research...

- From the Internet of Musical Things to the (Semantic) Web of Musical Things;
  In collaboration with Università di Trento
- enhance SPARQL-Generate;
  PostDoc at École des Mines de St-Étienne
- Apply the workflow previously presented to create Industry 4.0 environments; *École des Mines de St-Étienne: IT'M Factory*
- Study additional coordination mechanisms on these environments;
  École des Mines de St-Étienne: Plateforme Territoire

References

[1] X. Liu and O. Baiocchi, "A comparison of the definitions for smart sensors, smart objects and things in iot," in *2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pp. 1–4, IEEE, 2016.

[2] L. Turchet, C. Fischione, G. Essl, D. Keller, and M. Barthet, "Internet of Musical Things: Vision and Challenges," *IEEE Access*, vol. 6, pp. 61994–62017, 2018.

[3] L. Turchet, F. Antoniazzi, F. Viola, F. Giunchiglia, and G. Fazekas, "The internet of musical things ontology," *Journal of Web Semantics*, vol. 60, p. 100548, 2020.

[4] L. Roffia, P. Azzoni, C. Aguzzi, F. Viola, F. Antoniazzi, and T. Salmon Cinotti, "Dynamic linked data: A sparql event processing architecture," *Future Internet*, vol. 10, no. 4, p. 36, 2018.

[5] M. Ceriani and G. Fazekas, "Audio commons ontology: a data model for an audio content ecosystem," in *International Semantic Web Conference*, pp. 20–35, Springer, 2018.

[6] A. Xambó, J. Pauwels, G. Roma, M. Barthet, and G. Fazekas, "Jam with jamendo: Querying a large music collection by chords from a learner's perspective," in *Proceedings of the Audio Mostly 2018 on Sound in Immersion and Emotion*, pp. 1–7, 2018.

[7] M. Lefrançois, A. Zimmermann, and N. Bakerally, "A sparql extension for generating rdf from heterogeneous formats," in *European Semantic Web Conference*, pp. 35–50, Springer, 2017.

[8] D. Guinard, V. Trifa, and E. Wilde, "A resource oriented architecture for the web of things," in *2010 Internet of Things (IOT)*, pp. 1–8, IEEE, 2010.

[9] F. Antoniazzi and F. Viola, "Building the semantic web of things through a dynamic ontology," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10560–10579, 2019.


Questions?