ATAC 2021 : Crawl into the Dungeon with Hypermedea and Linked-data-Fu

A practical use of agents to navigate and do things through linked-data

ATAC 2021 : Crawl into the Dungeon with Hypermedea and Linked-data-Fu

Why using agents for Linked-Data navigation?

- Operating **regulated navigation** on using an expressive agent-based framework (e.g. JaCaMo) for more flexibility.

Some pros to the JaCaMo platform : Negation operator, Dynamic conditional rules management for navigation, simple logical expression, integration of complex behaviors through CArtAgO

- Provides also more «classical» pros related to agent: *communication, workflow distribution, autonomy.*

Hypermedea Linked-Data Fu Spider

Hypermedea : A platform designed for programming agent with hypermedia affordances

Linked Data-Fu Spider : A CArtAgO artifact in the hypermedea library implementing the Linked Data-Fu HTTP atomic request, providing operations for agents for Linked Data navigation with high performance, generating observable properties and supporting ontologies integration and reasoning

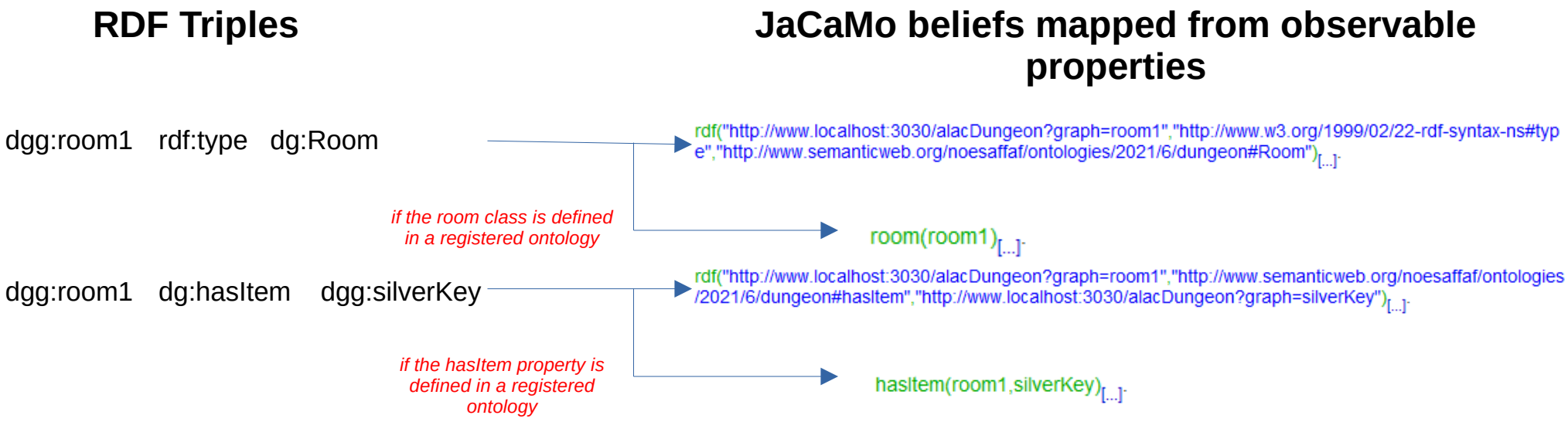
Observable Property : A state located in an environment that can be accessed by an agent, and mapped into an internal belief

Hypermedea Linked-Data Fu Spider

What are the possibilities?

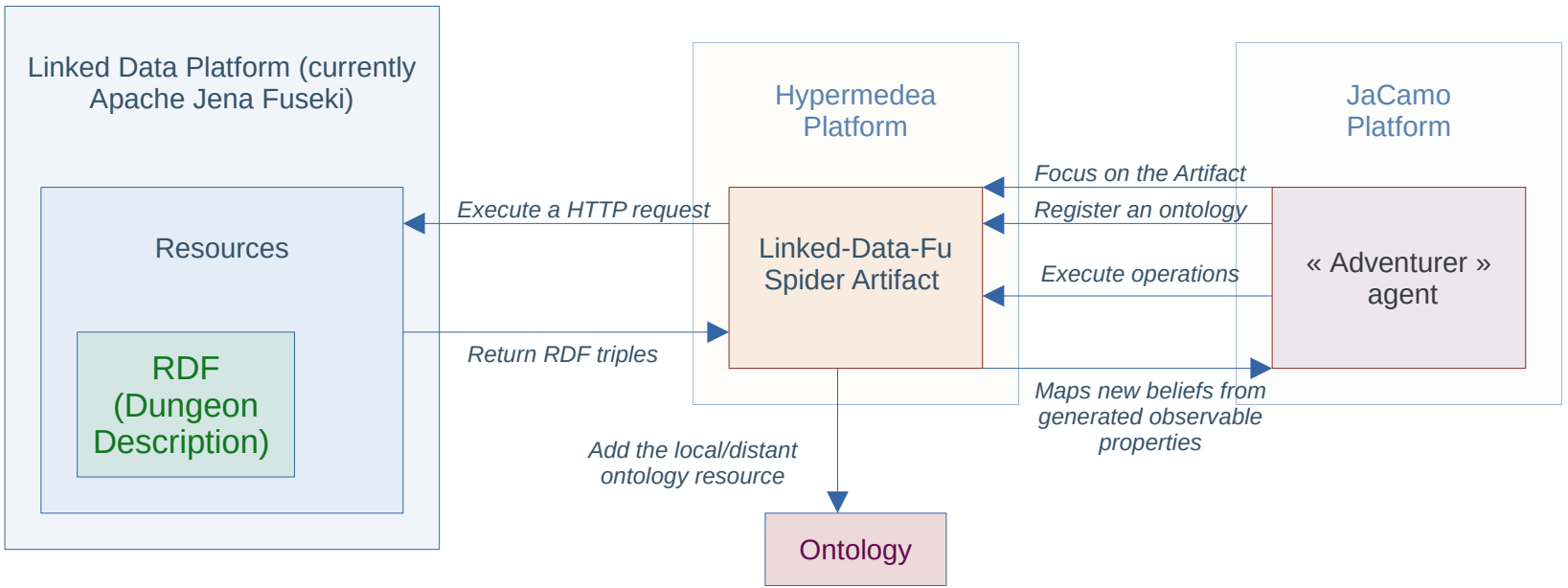
- Register/Unregister ontologies
- Execute CRUD operations (GET, POST, PUT, DELETE) to manipulate data from Linked Data platforms + generate new observable properties
- Internal manipulation of observable properties
- Reasoning

Hypermedea Linked-Data Fu Spider



Crawl Into The Dungeon

What is Crawl Into The Dungeon ?



Crawl Into The Dungeon

The dungeon is defined with a **RDF Graph structure representation.**

Each class instance of the dungeon owns a graph.

The root namespace is (may change): *http://www.localhost:3030/alacDungeon*

Principal entities constituting the dungeon are :

- **Rooms,** indirectly connected to each other through doors.
- **Doors,** connectors linking two rooms.
- **Keys,** items capable of opening certain doors.

Example of a room in TTL format located at :
http://www.localhost:3030/alacDungeon/room1/

```
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dg: <http://www.semanticweb.org/noesaffaf/ontologies/2021/6/dungeon#> .
@prefix dgg: <http://www.localhost:3030/atacDungeon/> .

dgg:room1 rdf:type dg:Room ;
          rdfs:label "Main Room" ;
          dg:hasConnection dgg:door1_2 ;
          dg:hasConnection dgg:door1_3 ;
          dg:hasItem dgg:silverKey ;
```

Crawl Into The Dungeon

Main actions the « adventurer » agent can execute to interact with the dungeon. Each calls a GET request using the LDFU-Spider Artifact :

- ***!investigate***
- ***!take(ITEM)***
- ***!look_doors***
- ***!move(DOOR)***

Crawl Into The Dungeon

Investigate action/plan

```
// The investigate plan searches for all items in the current room (C_ROOM),
// Look for all item it has, construct the IRI based on the item, and
// fetch all new triples through a sending GET request using the Linked-Data-Fu
// Spider extension.
+!investigate : currentRoom(C_ROOM) & rootEnvUri(ROOT_ENV_IRI) <-
    .concat("Investigating : ", C_ROOM, MSG);
    .print(MSG);
    //Look for all the items in the room
    for (hasItem(C_ROOM, ITEM)){
        .print("Found a ", ITEM, " !");
        .concat(ROOT_ENV_IRI, ITEM, ITEM_IRI);
        //Check if the triples has not already been crawled, and execute a GET
        //request through an external operation available in the Linked-Data-Fu
        //Spider extension.
        if (not key(ITEM_IRI)){
            get(ITEM_IRI);
        }
    }
```

Crawl Into The Dungeon

Take action/plan

```
// The take plan searches takes an Item in the current room and removes it by
// by sending a DELETE request
+!take(ITEM) : currentRoom(C_ROOM) & rootEnvUri(ROOT_ENV_IRI) <-
    //Check if the item is in the room
    if(hasItem(C_ROOM,ITEM)){
        //Creating an intern belief to state that the key is in our inventory
        .print("Taking from ", C_ROOM, " the item ", ITEM);
        +myInventory(ITEM);
        //Execute a DELETE request to remove the triple in room's IRI using the
        //the Linked-Data-Fu Spider extension.
        .concat(ROOT_ENV_IRI, ROOM, ROOM_IRI);
        delete(ROOM_IRI, hasItem(C_ROOM,ITEM));
    } else {
        .print("The item is not available in this room")
    }
    .
```

Crawl Into The Dungeon

Look_doors action/plan

```
// The look_doors plan searches for all the doors the room is connected to
// and extract data about those doors
+!look_doors : currentRoom(C_ROOM) & rootEnvUri(ROOT_ENV_IRI) <-
    .print("looking doors from : ", C_ROOM);
    for (hasConnection(C_ROOM, CONNECTION)){
        .concat(ROOT_ENV_IRI, CONNECTION, CONNECTION_IRI);
        if (not door(CONNECTION_IRI)){
            //Extract the data through an HTTP request
            get(CONNECTION_IRI);
        }
        for(hasConnectedRoom(CONNECTION, ADJ_ROOM)){
            if (not (ADJ_ROOM = C_ROOM)){
                .print("There is a door called ", CONNECTION, " that leads to ",ADJ_ROOM);
            }
        }
    }
    .
```

Crawl Into The Dungeon

Move action/plan

```
// The move plan look at all the doors connected to the room to see if one matches
// the one passed in parameter, and for that door, it looks at all possible keys to
// open the door and check if one of them is in our inventory, if so, it "moves"
// by switching the current room and extract information about that room
+!move(CONNECTION) : currentRoom(C_ROOM) & rootEnvUri(ROOT_ENV_IRI) <-
    // Check if the door (connection) is valid)
    if (hasConnection(C_ROOM, CONNECTION) & hasConnectedRoom(CONNECTION, ADJ_ROOM) & not (C_ROOM = ADJ_ROOM)){
        //Check all the keys the door can interact with
        for(hasInteractibleKey(CONNECTION, KEY)){
            //Check if there is such a key in the inventory
            if (myInventory(KEY)){
                //Update the current room
                .print("You opened the door with the key ", KEY, " and moved to ", ADJ_ROOM);
                -+currentRoom(ADJ_ROOM);
                .concat(ROOT_ENV_IRI, ADJ_ROOM, ADJ_ROOM_IRI);
                if (not room(ADJ_ROOM_IRI)){
                    //Extract the new room's data through a request
                    get(ADJ_ROOM_IRI);
                }
            } else {
                .print("You need the key ", KEY, " to open this door");
            }
        }
    }
    .
```

Crawl Into The Dungeon

- The « adventurer » agent is **autonomous, model-based agent**. Each turn of event, he decides of an action in a predefined and preferred order (e.g. investigating the room first if not yet done). Actions such as moving are decided randomly when there are multiple options.

- The « adventurer » agent offers a simple demonstration of an agent capable of bounded navigation through restrictive rules that can be declared with the expressivety of JaCaMo programation.

(e.g. Key requierement for using a door and accessing to the next room’s data).

Further improvement

- Extending the Dungeon environment by adding more rooms and other new elements

- Enhancing the Action Cycle which decide which action should the agent make next

- Using a Linked Data platform other than Fuseki to support more complex requests to delegate the autority of whether an agent can acces a certain resource to the environment rather than the agent itself (e.g. The environment should certify an agent has the key to a door to acces a new room rather than the agent itself)