

Crawl into the Dungeon with hypermedia agents

Noé Saffaf¹, Victor Charpenay¹

¹Mines Saint-Etienne, Laboratoire d'informatique, de modélisation et d'optimisation des systèmes (LIMOS)

Abstract

The emergence of Semantic Web technologies makes for a good opportunity for developing agent-based systems capable of interacting with our world through hypermedia. The growing number of Knowledge Graphs has been associated with the development of utilities to perform efficient Linked Data navigation. There is, however, certain gaps to fill for extending Linked Data navigation to the frontier of MAS systems, namely, by providing Linked Data affordance support suitable for agents while preserving their autonomous and supervisory aspect for taking decisions. Our project demonstrates a use case of an agent using our hypermedia artifact to connect the agent to Web environments. The presented application case is called “Crawl into the Dungeon”; a maze-like simulated environment, represented by a RDF graph, and we introduce an “adventurer” agent which implements hypermedia-driven behaviors from our artifact to perform self-decided graph navigation which translates into interactive actions with the dungeon in order to reach a defined goal.

Video: <https://youtu.be/DrHf5P2o3Ts>

Source Code: https://gitlab.emse.fr/noe.saffaf/atac_project

Keywords

Knowledge Graph, Multi-agent Systems, Linked Data navigation

1. Introduction

A motivation to bring autonomous agents and Linked Data applications together can be emphasized by the quote of Wooldridge, “Objects do it for free; agents do it because they want to” [1], underlining the autonomous aspect of the ideal agent that performs purposeful actions for satisfying a goal. Our focus on this project is to illustrate how agents can use a hypermedia extension tool to navigate through Linked Data, collect the data as part of the “state of mind” of the agent, and decide on actions, which in our case, are actions to interact with the dungeon. By using the Linked Data-Fu Spider extension, an intermediary software environment providing an interface for requesting triples through atomic requests and generate legible information for our agent; we administrate the navigation phase at the agent layer, and by that, attributing control directly to the agent itself. This entrenches the principle of decision-making authority of the agent, making them as we desire it; more autonomous. Furthermore, it also leverages the expressiveness of the used MAS framework language as we directly implement If-This-Then-That behaviors within the MAS framework and using its syntax rather than an external tool that would, by design, provide a more limited control delegated to the agent. In our case, we use the Jason language, component part of the JaCaMo framework, for which we enact sets of conditions and actions making use of the simplicity and the versatility of that language. This set of condition/action uses perceived knowledge of the agent (e.g. the existence of an item in the agent’s inventory found previously in the dungeon) to interact with our simulated dungeon hosted through a custom Knowledge Graph.

All the Agents Challenge (ATAC 2021)

✉ victor.charpenay@emse.fr (V. Charpenay)

🆔 0000-0002-9210-1583 (V. Charpenay)



© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

2. Linked Data-Fu Spider : An artifact for agent navigation

Hypermedea is a platform designed for programming hypermedia agents with a cross-platform design for multi-agent systems. It includes the Linked Data-Fu Spider extension, an CArtaGO [2] artifact providing operations for agents to perform HTTP requests and generates observable properties –states– perceivable for agents. Linked Data-Fu Spider implements Linked Data-Fu’s components [3] for performing unitary HTTP requests, and providing a usage interface agents can operate with. Additionally, the extension includes reasoning with OWL ontologies which generates more observable states obtained by inference from registered terminology axioms and collected data through navigation.

The JaCaMo [4] platform allows to build agents operating with the Belief–Desire–Intention model and capable of using our hypermedia extension to invoke operations and perceive generated observable properties, mapping them as beliefs from an external source. Through logical rules denoted as navigation rules, internal and external beliefs generate new sets of actions to be performed and triggers invocable operations in the usage interface of Linked Data-Fu Spider artifact. This cyclic process allows agent to iteratively perform HTTP requests and collecting data to decide on new actions to perform. Such implementation based after the Agent & Artifact meta-model, provides the benefit of segmenting the decision phase, performed solely by the agent in the agent layer; from the heavy computational data-processing proposed as a service from the artifact that includes –among others– reasoning over ontologies. This allows to the developer to implement navigation rules using directly the syntax of the used MAS platform rather than relying on a external tool. We represent the described architecture model as below (1) :

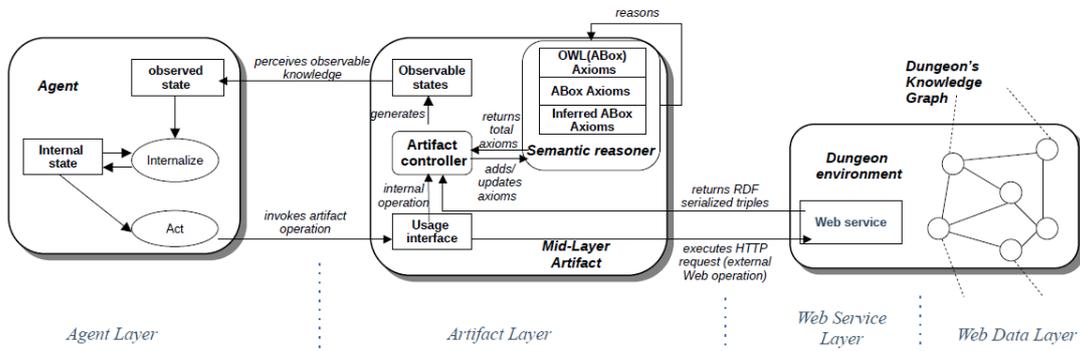


Figure 1: Agent & Artifact Model

At the initiation phase, an agent should indicate the vocabulary used by the environment he desires to interact with. In the case of our dungeon, we provide a custom OWL ontology for the agent to register through the artifact. This includes definition of concepts from our dungeon such as the *Room* class or the *hasItem* property. Registering an ontology assigns the TBox axioms in the OWL reasoner within the artifact and are conserved for inference processing. Below illustrates a couple of examples on how the artifact converts RDF triples obtained from HTTP responses into structurally valid and reusable observable properties, mapped as beliefs in our JaCaMo framework.

RDF Triple	Belief in JaCaMo
<i>:room1 a dg:Room</i>	<i>room(room1)</i>
<i>:silverKey a dg:Key</i>	<i>key(silverKey)</i>
<i>:silverKey a dg:Key</i>	<i>item(silverKey)</i>
<i>:room1 dg:hasItem :silverKey</i>	<i>hasItem(room1,silverKey)</i>

A triple representing a class assertion of a room will, as an example, generate a unary observable property. A property will generate a binary observable property (e.g. *hasItem*). Note that, in our ontology of the Dungeon, we define the class *Key* as a subclass of *Item*, therefore, if a reasoner is applied to our LDFU-Spider artifact, the produced set of observable properties also includes inferred properties such as an instance of *Key* also being an instance of *Item*. We use in our scenario the Hermit OWL reasoner [5].

Currently, the main functionalities supported by our artifact can be enumerated as follow:

- Register/Unregister operations to add and modify ontologies for our semantic reasoner
- HTTP operations (GET, POST, PUT, DELETE) for triple acquisition/manipulation from Linked Data platforms, and add new beliefs to the data base RDF triples and unary/binary predicates if the vocabulary has been previously registered
- Operations to insert/remove custom triples
- Reasoning

3. The dungeon environment

To put our software extension for navigating agents into application, and to demonstrate a simple and practical for Linked Data navigation by agent. We introduce the following model : An “adventurer” agent can interact with an external and custom environment; the Dungeon, represented by entities such as rooms, doors and items. Initially, the adventurer starts in a specific room (*room1*) and has to reach a goal room (*room4*). We consider three main entities for our simplified dungeon, which are rooms, doors and keys. We also associate every entity to a URI node providing a unique RDF document describing properties of that entity. A room can contain properties such as *hasItem*, linking to other items of the dungeon, or a key can own *hasInteractable* properties referring to doors that can be opened with that key. We propose a schematic representation of our dungeon and its Knowledge Graph implementation (2).

To generate this Knowledge Graph on a local server accessible by our agent through the intermediary of the artifact, we use the Linked Data platform Apache Jena Fuseki as an embedded server for our Java project. The root URI of our dungeon is “http://localhost:3030/atacDungeon/”. The objective of the agent is to reach a certain goal in the dungeon (a particular room in this implementation). To do so, he is capable of multiple actions, each interacting with its environment. The basic actions implemented as plans in JaCaMo are listed below:

- **investigate** : The agent searches for ITEMS in the room, and will form an HTTP get request to fetch all triples concerning every item present in the room.
- **take(ITEM)** : The agent takes the ITEM passed as parameter from the current room. It will add the item to its own inventory by creating a *myInventory(ITEM)* belief.
- **look_doors** : The agent looks for doors inside the room, and similarly to investigate, will form a get request to retrieve the according triples.

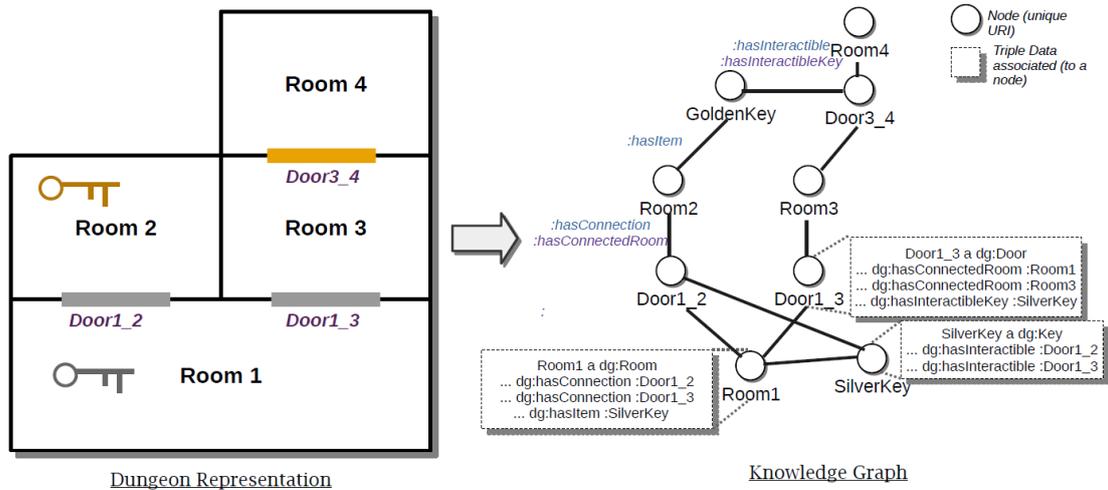


Figure 2: Dungeon representation

- **move(DOOR)**: The agent looks at the restriction of the DOOR in parameter (e.g. a specific key) and check if it has in its inventory the requested item. If so, the agent “moves” by changing in its belief base his current room, and request all triples for the new room.

To understand how such actions are implemented, let us assume our agent is currently in a specific room that we represent as a belief *currentRoom(C_ROOM)*, and a GET request associated to the URI of that room has already been performed, generating observable properties about that room. To investigate the room would be for our agent to perform a GET request to any item found in that room using the *hasItem(ROOM, ITEM)* belief. A simple implementation in JaCaMo can be done as follow (3) :

```

+!investigate : currentRoom(C_ROOM) <-
  for (hasItem(C_ROOM, ITEM)){
    get(ITEM);
  }.

```

Figure 3: Simple Investigate plan

The adventurer agent is autonomous, on a model-based and with a cyclic implementation. The agent’s lifecycle includes an observation phase with a goal verification step to check at each round if the new agent’s state meets the goal conditions, and an action phase where different conditions are checked in a listed and preferential order. Only one action is triggered per cycle and other actions are skipped whenever one above on the list is executed (4).

Through the simple and reusable structure of the generated observable properties, it gives flexibility to the user to define their own rules based on the MAS framework they use, which in our case, is JaCaMo. The JaCaMo syntax offers possibilities to defines conditions with negation, logical or even relational operators with simplicity. Additionally, conditions can be generic by considering more high-level abstraction beliefs such as *item(KEY)* rather than low-level beliefs like *key(KEY)* as an incidence of the artifact’s reasoning functionality. Combined, these elements allow to write versatile rules close to the developer’s intention, and that our adventurer agent

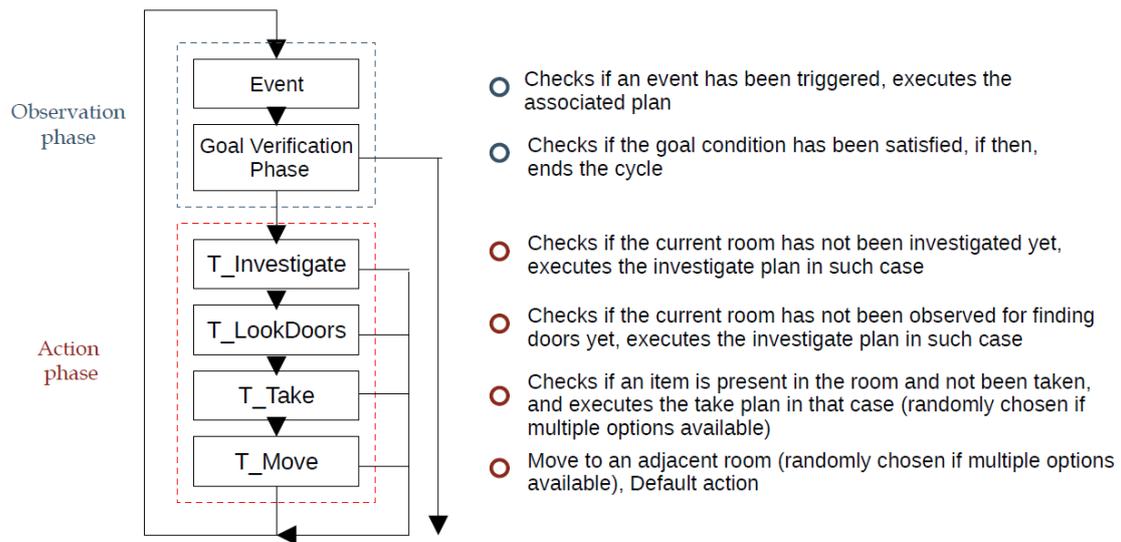


Figure 4: Adventurer agent's life-cycle

implementation may take more advantage of. For instance, extended scenarios may consider elements such as quantifiable or boolean attributes to develop more sophisticated conditions such as :

```
if (isPoisoned(BOOL) & BOOL & characterHP(HP) & roomHPRestriction(ROOM, HP_MIN)
    & HP > HP_MIN) do { get(ROOM) }
```

The above condition stipulates some condition which infers internal information such as the navigating agent's HP to collected information such as HP restrictions about a specific room.

4. To expand the project

As we have proved through our scenario a simple example on how agents may access to Linked Data and adapting its behavior based on the received information by implementing simple and intuitive rules for navigation, our project aims to encourage future MAS developers to consider agents for tasks on Open Linked Data through the usage of a hypermedia service. Here, we presented an overture on how agents can accomplish bounded navigation, efficiently restricting the number of requests to external Web resources through the decision of a unique action, this, by implementing accessible, intuitive and succinct conditions with possible reusability over multiple MAS frameworks.

5. Acknowledgments

This work was partially funded by the ANR/SNF project HyperAgents (grant no. ANR-19-CE23-0030).

References

- [1] M. Wooldridge, An introduction to multiagent systems, John wiley & sons, 2009.

- [2] A. Ricci, M. Viroli, A. Omicini, CArtaGO: A framework for prototyping artifact-based environments in mas, in: International Workshop on Environments for Multi-Agent Systems, Springer, 2006, pp. 67–86.
- [3] S. Stadtmüller, S. Speiser, A. Harth, R. Studer, Data-fu: A language and an interpreter for interaction with read/write linked data, in: Proceedings of the 22nd International Conference on World Wide Web, Association for Computing Machinery, New York, NY, USA, 2013, p. 1225–1236.
- [4] O. Boissier, R. H. Bordini, J. F. Hübner, A. Ricci, A. Santi, Multi-agent oriented programming with JaCaMo, *Science of Computer Programming* 78 (2013) 747–761. doi:<https://doi.org/10.1016/j.scico.2011.10.004>.
- [5] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, Z. Wang, HermiT: an owl 2 reasoner, *Journal of Automated Reasoning* 53 (2014) 245–269.