Hypermedea: A Framework for Web (of Things) Agents

Victor Charpenay
victor.charpenay@emse.fr
Mines Saint-Etienne, Univ Clermont Auvergne, INP
Clermont Auvergne, CNRS, UMR 6158 LIMOS
Saint-Etienne, France

Antoine Zimmermann
antoine.zimmermann@emse.fr
Mines Saint-Etienne, Univ Clermont Auvergne, INP
Clermont Auvergne, CNRS, UMR 6158 LIMOS
Saint-Etienne, France

Maxime Lefrançois
maxime.lefrancois@emse.fr
Mines Saint-Etienne, Univ Clermont Auvergne, INP
Clermont Auvergne, CNRS, UMR 6158 LIMOS
Saint-Etienne, France

Olivier Boissier
olivier.boissier@emse.fr
Mines Saint-Etienne, Univ Clermont Auvergne, INP
Clermont Auvergne, CNRS, UMR 6158 LIMOS
Saint-Etienne, France

ABSTRACT

Hypermedea is an extension of the JaCaMo multi-agent programming framework to act on Web and Web of Things environments. In this demo, the performance of Hypermedea's Linked Data navigation and planning components are evaluated, both encapsulating computation-intensive algorithms.

CCS CONCEPTS

• **Information systems** → *RESTful web services*; • **Networks** → **Cyber-physical networks**.

KEYWORDS

Hypermedia, Web of Things, Jason, JaCaMo, Linked Data

ACM Reference Format:
Victor Charpenay, Antoine Zimmermann, Maxime Lefrançois, and Olivier Boissier. 2022. Hypermedea: A Framework for Web (of Things) Agents. In *Companion Proceedings of the Web Conference 2022 (WWW '22 Companion), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3487553.3524243

1 INTRODUCTION

Recently, the standardization of the Thing Description (TD) model by the World Wide Web Consortium (W3C)[1] renewed interest for Web agents. The TD model is part of an effort to build what is called the Web of Things (WoT), i.e. bring sensors, actuators and digitally tagged devices to the Web. The main idea behind WoT is similar to that of semantic Web services: if connected devices had a uniform (semantically described) interface, they could be dynamically combined to build 'physical mashups' [4]. Yet, it appears that most WoT systems require continuous monitoring to be properly controlled. As an example, a manufacturing execution system is designed to optimize continuous flows of materials and final products, instead of individual manufacturing processes.

As a consequence, WoT control systems require a combination of proactivity, by composing Web services, and reactivity, by acting in response to spontaneous changes in the system. The combination of the two is what characterizes agent-oriented programming languages. We introduce here Hypermedea, an open-source framework for Web and WoT agents[2]. Hypermedea is based on JaCaMo, a multi-agent oriented programming platform [1], whose relevance to develop Web agents has recently been highlighted [2]. In this demo, we more specifically show how JaCaMo, via Hypermedea, can be used in the context of WoT.

2 RELATED WORK

We have identified two main agent frameworks for controlling WoT systems: UberManufacturing, in its original form [8] as well as in an extended form that leverages the JaCaMo multi-agent platform [3], and Linked-Data-Fu [6, 10]. Both frameworks rely on Semantic Web technologies, as originally motivated by Hendler two decades ago [5], but they slightly differ in the level of complexity of their cognitive loop.

The UberManufacturing platform, introduced by Mayer *et al.* [8], is meant to provide a uniform control layer over actuators and human workers, where the two collaborate to produce goods (an assembled stool, in the original UberManufacturing publication). The control system can be considered as an agent that executes a basic cognitive loop: synthesize a plan based on an order (a stool to assemble) and a description of available producers (automated workstations and human workers); execute the plan step-by-step; if a producer agent signals an error during execution, re-plan with current step as goal. The control agent has a single high-level goal: to process orders as they are placed. The automated planner used in UberManufacturing is derived from the EYE proof engine (Euler Yet Another Proof Engine). EYE is a Prolog program that can derive proofs from a set of first-order logic statements [11]. In particular, EYE can be used as a Web service composition tool (then referred to as RESTdesc) to apply on semantically annotated TD documents. Since EYE is not restricted to automated planning, its performances are however not optimal for this task (as shown later, in Sec. 4).

[1] https://www.w3.org/TR/wot-thing-description/

[2] https://github.com/Hypermedea/hypermedea

WWW '22 Companion, April 25–29, 2022, Virtual Event, Lyon, France

ACM ISBN 978-1-4503-9130-6/22/04…$15.00
https://doi.org/10.1145/3487553.3524243

UberManufacturing was later extended by Ciortea *et al.*, taking into account the latest advances in multi-agent system engineering [3]. In its extended form, UberManufacturing integrates EYE into JaCaMo. In contrast to the earlier proposal of Mayer *et al.* [8], agents run a complex decision loop involving three modalities: belief, desire and intention (BDI). BDI agents maintain an internal state (beliefs) and pre-defined plans that agents select according to their goals (desires) and execute (intentions). Ciortea *et al.* add the possibility of synthesizing new plans at runtime with EYE. They also demonstrate the use of multi-agent organizations to share a plan, such as a stool assembly plan, and collectively execute it.

Linked-Data-Fu, the second framework we identified, was initially designed to perform Linked Data navigation. To navigate and discover their environments, agents dereference resource identifiers (URIs) and follow hyperlinks embedded in the underlying Web resource. As Web environments may include many interlinked resources, Linked-Data-Fu is designed to program simple reactive agents (without planning) with optimized dereferencing of links [7]. Käfer and Harth showed however that Linked-Data-Fu agents can be combined with Linked Data Platforms to become a Turing-complete programming framework [7]. Linked-Data-Fu also supports automated reasoning via the Web Ontology Language (OWL): it infers statements on-the-fly based on OWL axioms found in a Linked Data environment, while navigating through it. Linked-Data-Fu being a pure rule engine, it supports a strict subset of all possible OWL axioms, though (e.g. the OWL 2 RL profile).

Hypermedea is an attempt to build upon the strengths of both UberManufacturing (planning) and Linked-Data-Fu (reasoning). Hypermedea agents are BDI agents that can synthesize plans from domains in the well-defined Planning Domain Definition Language (PDDL) [9] more efficiently than EYE and they are capable of Linked Data navigation and OWL reasoning, with full support of OWL 2 DL axioms (the largest decidable fragment of OWL).

3 ARCHITECTURE

3.1 The JaCaMo Platform

JaCaMo is a multi-agent oriented programming platform composed of 3 elements: Jason (a BDI agent architecture with its own agent-oriented programming language), CArtAgO (an infrastructure for designing agent environments) and Moise (a meta-model for building organizations of agents governed by norms) [1]. Moise is out of scope of the present paper, we briefly describe Jason and CArtAgO in the following.

Jason is among the most widely known languages to program BDI agents. A Jason agent maintains an internal state as a set of ground terms (analogous to facts in logic programming). Such terms are referred to as "beliefs". The agent's beliefs are influenced by what the agent perceives from the environment. The agent also maintains a set of goals to achieve as "desires". To act on its environment, the agent executes plans triggered by internal events (belief revision or change in its desires). A plan is a sequence of actions that the agent commits to—while other events may occur, triggering new plans. A plan committed to is called an "intention".

Jason actions are represented as functional terms (action name and parameters) whose return value is either success or failure. Actions generally have further effects in the environment in which the agent is situated, though. The semantics of Jason actions are environment-dependent. The role of CArtAgO, the Common "Artifacts for Agents" Open framework, is to provide a unified mechanism to interface agents with arbitrary environments. To do so, CArtAgO introduces the notion of "artifact", which is a tool that agents use to interact and act on their environment. A CArtAgO artifact is an abstract entity declaring "operations" and "observable properties". Operations are actions an agent is able to perform on the artifact and observable properties are what the agent can perceive from the artifact. Developers can implement CArtAgO artifacts as Java classes.

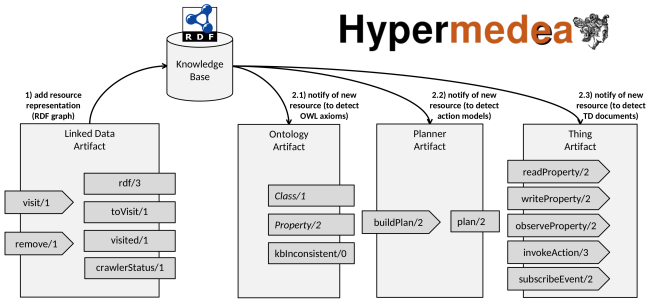


Figure 1: Technical overview of the 4 Hypermedea artifacts; artifacts have observable properties (rectangle boxes) and operations (arrow-shaped boxes); *Class* and *Property* are templates to be instantiated with terms defined in ontologies

3.2 Hypermedea Artifacts

Hypermedea is implemented as a collection of CArtAgO artifacts, as has already been suggested in the literature [2]. An overview of the observable properties and operations of all four Hypermedea artifacts is given in Fig. 1, following the graphical notation of the multi-agent oriented programming book [1]. Operations and observable properties are denoted with name and arity (e.g. `rdf/3` for the ternary property `rdf`). Each artifact is also documented in the Hypermedea code repository[3].

We now describe the Hypermedea artifacts in the order in which they are likely to be used as the BDI agent acts "intentionally": the Linked Data artifact first (for perceiving the environment), then the ontology artifact (for expanding perception with inferred statements), then the planner artifact (preceding action) and, finally, the Thing artifact (for acting on the environment).

A *Linked Data artifact* is responsible for retrieving RDF representations of resources. It is up to agents to decide what resource to visit. When an agent calls the `visit/1` operation with a URI as argument, a worker thread dereferences the URI and retrieves a representation of the resource in the form of an RDF graph (indexed by the dereferenced URI). Currently supported URIs are HTTP(S) and file (pseudo-)URIs. In order to let the agent manage navigation, observable properties indicate which URIs are being dereferenced (`toVisit/1`), which ones have been already dereferenced (`visited/1`) and whether at least one worker thread is being active

[3] https://hypermedea.github.io/javadoc/latest/

(`crawlerStatus/1`, with `active`, `idling` or `error` as possible values). The Linked Data artifact manages a fixed number of worker threads in a thread pool. In order for agents to "forget" (`remove/1`) at a later time about resources (e.g. if the set of visited resources is logically inconsistent, as detailed in the next paragraph), all named graphs are kept in an in-memory knowledge base (Apache Jena). Each triple included in the representation of a visited resource is also added to the state of the artifact (`rdf/3`). The resource's URI is kept as annotation.

The *ontology artifact* listens to changes in the knowledge base and processes the OWL definitions found in incoming named graphs. OWL declarations of classes, object properties and datatype properties are used for generating unary and binary observable properties[4] from any subsequently found RDF triple. This syntactic transformation from URIs to shorter names aims at closing the gap between the RDF models and predicate representations, more commonly used in agent-oriented programming languages. More importantly, however, the ontology artifact also performs OWL reasoning. It materializes any class or property assertion that can be inferred from the knowledge base as new unary and binary predicates. If the knowledge base turns out to be inconsistent, a flag is raised (`kbInconsistent/0`), so that all agents focusing on the artifact are notified. Agents can then resolve conflicts in the knowledge base by using `remove/1` of the Linked Data artifact. The ontology artifact consequently deletes inferred assertions invalidated by the removal. The resolution strategy is entirely left to agents, to preserve their autonomy. The ontology artifact is mainly implemented using the ONT-API, an implementation of the OWL API over Apache Jena. The current implementation includes the HermiT OWL reasoner[5], although any reasoner compliant with the OWL API could be used as a substitute.

By default, BDI agents execute actions according to pre-defined plans stored in their plan library. However, Jason allows for meta-programming, allowing agents to add synthesized plans to their plan library at runtime. This feature of Jason is particularly useful in the context of WoT: as multiple Things expose their services simultaneously, agents must select and order these services according to their current goals (or intentions, in the case of BDI agents). The *planner artifact* takes advantage of the meta-programming feature of Jason to expose synthesized plans as observable properties (`plan/1`). Agents can trigger planning with the `buildPlan/2` operation, taking two parameters: a planning domain (the description of available services) and a planning problem (an initial state and a goal state). It is then up to agents to add the synthesized plan to their plan library, to modify it (e.g. in case of failure while executing the plan) or to ignore it (e.g. if the state of the environment has drifted away from the initial state used for planning). The input planning domain and problem that are provided to the artifact to build a plan are serialized as PDDL definitions, such that they can be processed by any PDDL planner. The planner artifact uses PDDL4J[6], an open-source library to manipulate PDDL definitions, and FF-X[7], an extended version of the well-known Fast-Forward (FF) planner.

[4] 'object properties' and 'datatype properties' refer to OWL constructs while 'observable properties' refer to CArtAgO constructs, defined at another level.
[5] https://github.com/owlcs/hermit-reasoner
[6] https://github.com/pellierd/pddl4j
[7] https://fai.cs.uni-saarland.de/hoffmann/ff.html

The *Thing artifact* class is meant to have as many instances as there are Things in the environment[8]. Every time an incoming resource representation includes a full TD document, a new instance of the artifact is created, acting as a proxy for the physical Thing. A Thing artifact instance provides an operation for each operation type defined in the TD standard (`readProperty/2`, `writeProperty/2`, `observeProperty/2`, `invokeAction/3` and `subscribeEvent/2`). The Thing artifact class currently supports one protocol binding only (HTTP) but more bindings may be added in the future without modifying its interface. New protocol bindings for the Robot Operating System (ROS) and OPC Unified Architecture (OPC UA) are currently under development.

4 PERFORMANCE EVALUATION

By integrating with Jason via CArtAgO artifacts designed for Web and WoT agents, Hypermedea inherits the expressiveness of the Jason language. We now evaluate Hypermedea with the intention to show that a higher expressiveness does not come with degraded performances compared to UberManufacturing and Linked-Data-Fu.

We evaluate the performances of Linked Data navigation together with OWL reasoning[9]: reasoning should indeed occur incrementally, during Linked Data navigation. We then evaluate the performances of PDDL planners against EYE, the proof engine used for deriving plans in UberManufacturing. Automated reasoning and planning are known to be computation-intensive tasks. In contrast, HTTP communication as performed by the Thing artifact can hardly be optimized from a client perspective only. Since most tools rely on the same Apache HTTP client library, we do not evaluate the Thing artifact.

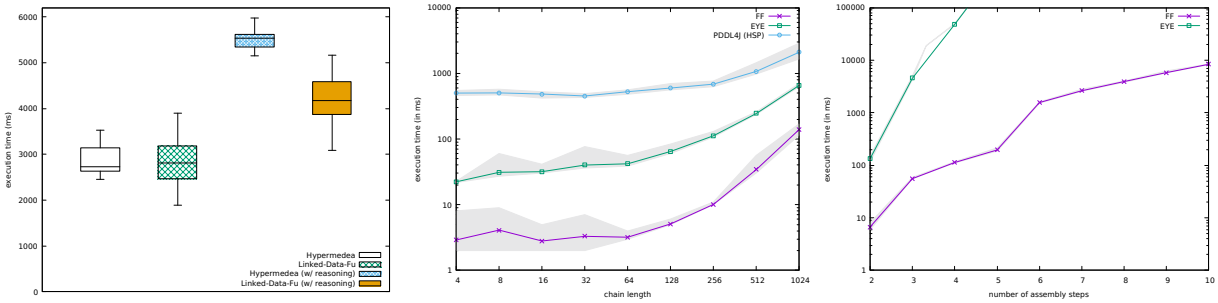
4.1 Linked Data Navigation and Reasoning

To evaluate Linked Data navigation and reasoning, we use the dataset that describes an experimental factory line located on the ground floor of Mines Saint-Étienne's facility. We also include the description of the remainder of the building, such that the whole dataset includes 7,380 RDF triples defined in 290 interlinked resources. It is exposed online according to the Linked Data principles[10] The assertions in the dataset reference five ontologies that declare a total of 149 classes, 72 object properties and 11 datatype properties. The ontologies also have 207 logical axioms, including 123 sub-class axioms. When applied on the production line dataset, 1150 assertions can be inferred from all logical axioms. Most non-trivial axioms being inferred are adjencies between rooms.

We compare Hypermedea with Linked-Data-Fu (the two versions of UberManufacturing, by Mayer *et al.* [8] and Ciortea *et al.* [3] do not support Linked Data navigation). Results in terms of execution time are shown on Fig 2a. Results suggest that, in average, Hypermedea and Linked-Data-Fu have comparable performances for pure Linked Data navigation (i.e. without reasoning).

In terms of reasoning, Linked-Data-Fu outperforms Hypermedea: the overhead caused by OWL reasoning is at least twice as important for Hypermedea (median: 2.7s) as for Linked-Data-Fu

[8] in contrast, other artifacts may be singletons.
[9] all evaluations were run on a Intel Core i7 processor (1.8 GHz, 8 cores).
[10] https://ci.mines-stetienne.fr/kg/



(a) Linked Data navigation **(b) Planification (RESTdesc benchmark)** **(c) Planification (SIRAM scenario)**

Figure 2: Performance evaluation of Hypermedea (average execution time over 10 runs)

(median: 1.2s). It is however worth noting that, in order to have a fair comparison, the evaluation dataset does not include axioms that lie outside the OWL RL profile (which is the largest fragment of OWL supported by Linked-Data-Fu). Linked-Data-Fu can therefore implement dedicated optimization techniques that do not apply to OWL reasoners such as HermiT (which support a broader set of axioms). A total execution time below 6s is still low, making Hypermedea useful in practice.

4.2 Automated Planning

Automated planning has been the subject of the International Planning Competition (IPC) for many years. The EYE proof engine, because it is a more generic tool, has never been evaluated against IPC benchmarks. It has, however, been tested against a simpler benchmark consisting of chaining Web services with a single statement as pre-condition and a single (but different) statement as effect[11]. We therefore ran FF-X against the RESTdesc benchmark with growing chain lengths. Results are shown on Fig. 2b. FF-X consistently outperforms EYE, roughly tenfold. We also ran an alternative planner implementation provided by PDDL4J[12]. The main difference between FF-X and this PDDL4J implementation is that the former is native while the latter runs in the JVM. The JVM seems to introduce a significant overhead, as PDDL4J's planner is itself outperformed by EYE.

The RESTdesc benchmark is rather straightforward compared to IPC benchmark scenarios, though. We therefore compare FF-X and EYE in a benchmark that combines two classical IPC tasks: the assembly benchmark, by Drew McDermott, and the gripper benchmark, by Jana Koehler. In the combined task, an Antomated Guided Vehicle (AGV) equipped with a gripper has to concurrently find its way throughout the factory floor and to pick (place) items stored at source (target) locations of the AGV's path. Such an augmented AGV has e.g. been prototyped in the SIRAM research project[13]. As shown on Fig. 2c, FF-X clearly outperforms EYE, the more significantly when the number of assembly steps grows. For an assembly process with 5 steps, it takes more than a thousand times longer for EYE to find a plan (600s) than for FF-X (200ms).

[11] https://github.com/RubenVerborgh/RESTdesc-Composition-Benchmark
[12] PDDL4J's primary purpose is however to read and write PDDL definitions.
[13] https://siram.mecaconcept.com/

5 CONCLUSION

Hypermedea has been used and tested in the SIRAM project and in a series of summer schools[14]. Its use in these two real-world scenarios is to be demonstrated at WWW 2022, beyond performance evaluation. Its design was driven by WoT applications but could find a broader audience, in particular for social applications, against personal information management systems such as Social Linked Data (Solid)[15]. As mentioned in the paper, further work is already planned on extending the Thing artifact with protocol bindings for robotics and industrial computing.

ACKNOWLEDGMENTS

This work was partially funded through the following projects: SIRAM, HyperAgents (grant ANR-19-CE23-0030-01) and CoSWoT (grant ANR-19-CE23-0012-04).

REFERENCES

[1] Olivier Boissier, Rafael H. Bordini, Jomi Fred Hübner, and Alessandro Ricci. 2020. *Multi-agent oriented programming: programming multi-agent systems using JaCaMo.* The MIT Press, Cambridge, Massachusetts.
[2] Andrei Ciortea, Olivier Boissier, and Alessandro Ricci. 2019. Engineering World-Wide Multi-Agent Systems with Hypermedia. In *EMAS 2019.*
[3] Andrei Ciortea, Simon Mayer, and Florian Michahelles. 2018. Repurposing Manufacturing Lines on the Fly with Multi-agent Systems for the Web of Things. In *AAMAS 2018.*
[4] Dominique Guinard and Vlad Trifa. 2009. Towards the Web of Things: Web Mashups for Embedded Devices. In *MEM 2009.*
[5] James Hendler. 2001. Agents and the Semantic Web. *IEEE Intelligent Systems* 16, 2 (2001), 30–37.
[6] Felix Leif Keppmann, Maria Maleshkova, and Andreas Harth. 2016. Semantic Technologies for Realising Decentralised Applications for the Web of Things. In *ICECCS 2016.*
[7] Tobias Käfer and Andreas Harth. 2018. Rule-based Programming of User Agents for Linked Data. In *LDOW 2018.*
[8] Simon Mayer, Dominic Plangger, Florian Michahelles, and Simon Rothfuss. 2016. UberManufacturing: A Goal-Driven Collaborative Industrial Manufacturing Marketplace. In *IoT 2016.*
[9] Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. 1998. *PDDL–The Planning Domain Definition Language.* Technical Report CVC TR-98-003.
[10] Steffen Stadtmüller, Sebastian Speiser, Andreas Harth, and Rudi Studer. 2013. Data-Fu: a language and an interpreter for interaction with read/write linked data. In *WWW 2013.*
[11] Ruben Verborgh and Jos De Roo. 2015. Drawing Conclusions from Linked Data on the Web: The EYE Reasoner. *IEEE Software* 32, 3 (2015), 23–27.

[14] https://ai4industry.wp.imt.fr/
[15] https://solidproject.org/