A Rule-Based Software Agent on Top of Personal Data Stores

Wout Slabbinck[1,*], Ruben Dedecker[1], Julián Andrés Rojas[1] and Ruben Verborgh[1]

[1]IDLab, Departement of Electronics and Information Systems, Ghent University - imec, Belgium

Abstract

As a response to centralised platforms hoarding user data on the web, Personal Data Stores (PDSs) are becoming more prominent. In this emerging space of decentralized PDS platforms, we see a growing need for automated agents to take over tasks that are normally handled by those data platforms. These intelligent agents already have their place in our daily lives, however, they are currently largely missing from the domain of Personal Data Stores. To address this need, we created a demonstrator software web agent that acts on the real world and personal data via condition–action rules. We applied this demonstrator to a smart home environment use case, where smart home appliances can be actuated and monitored via a personal data store. The generic architecture used by the implementation leads to maximal re-use of the agent, enabling multi-agent systems and researching more complex use cases.

Keywords

Intelligent software web agents, Personal Data Stores, Linked Data, Solid

1. Introduction

Personal Data Stores are a reaction from emerging anti-consumer business models becoming more prevalent in the online space. By building on existing web standards with the vision to separate data from applications, the Solid Protocol[1] facilitates the implementation of a decentralized ecosystem of PDSs. Data is stored in a Solid pod, a personal data vault, and Solid apps use the protocol to interact with the data on a pod.

The current Solid ecosystem is very application-oriented. It focuses on web and native applications that build on direct read/write interaction with the data pods. These applications are less suited for automated background processes such as automating notifications and more complex actions such as continuous integration with third parties, e.g., Internet of Things (IoT) devices. In centralised data platforms, these tasks are handled by dedicated applications that run on the platform. In decentralised networks of PDSs, these tasks require independent and automated web agents capable of executing diverse tasks. Based on the work of Kirrane [1],

ISWC 2023 Posters and Demos: 22nd International Semantic Web Conference, November 6–10, 2023, Athens, Greece

*Corresponding author.

wout.slabbinck@ugent.be (W. Slabbinck); ruben.dedecker@ugent.be (R. Dedecker); julianandres.rojasmelendez@ugent.be (J. A. Rojas); ruben.verborgh@ugent.be (R. Verborgh)

https://rubendedecker.be/ (R. Dedecker); https://julianrojas.org/ (J. A. Rojas); https://ruben.verborgh.org/ (R. Verborgh)

0000-0002-3287-7312 (W. Slabbinck); 0000-0002-3257-3394 (R. Dedecker); 0000-0002-6645-1264 (J. A. Rojas); 0000-0002-8596-222X (R. Verborgh)

CEUR Workshop Proceedings (CEUR-WS.org)

[1]https://solidproject.org/TR/protocol

who formalised a hybrid web agent architecture, we provide a reference implementation that operates on behalf of a user utilizing a PDS. We created a demonstrator Solid Agent that is capable of reasoning over PDSs and actuates in the context of smart home scenarios. It is designed to be generic and extensible so that the agent is not limited to the concrete use case presented in this paper. The agent uses Notation3 (N3) [2] rules to reason over RDF graphs present in PDSs, which contain the state of the world from the agent's point of view, and constitute the input for executing concrete tasks.

2. Related Work

Käfer et al. [3] presented a semantic user agent modelled on Abstract State Machines (ASMs) using the *Linked Data-Fu* engine. This engine interacts with the environment via HTTP and applies condition-action rules which are represented using N3.
Schraudner et al.[4] also used Linked Data-Fu to model multi-agent systems (MAS) using the *stigmergy* communication paradigm. With this concept, the agents act on the environment rather than communicating directly with each other. Linked Data-Fu seems to be currently unmaintained and no open-source implementations are available.
Zimmerman et al. [5] introduced *pody*, which defines a Web Agent embodied in a Solid pod. Through this embodiment, they envision that podies can participate as part of a MAS. No open source implementations are currently available.
Lastly, Kirrane [1] introduced a framework to assess task environment requirements of various agent use case scenarios. Furthermore, she defined a hybrid semantic web agent architecture consisting of five modular components: *Interface*, *Controller*, *Reactive*, *Deliberative* and *Learning*. Our work follows such architecture given its comprehensive requirement analysis for agents and aims to provide an initial reference implementation.

3. Demo

Smart home applications such as security cameras, smart speakers and smart lights are becoming increasingly popular in households. Ideally, this data would be securely stored and managed in a personal data vault, so that managed access can be granted to external systems. To achieve this, we created a web agent based on the hybrid agent architecture [1], that synchronises the state of a smart device with a corresponding resource in a Solid pod. Figure 1 shows the setup of this demo. It consists of a Philips Hue smart bulb that is actuated wireless by the Hue Bridge via Zigbee[2]. Rather than directly integrating with Philips Hue, which would limit our use cases to that particular system, we integrated with openHAB[3], which is an open-source platform with support for multiple smart home appliances (including Philips Hue). The state of the openHAB item is then synchronised with a corresponding `ldp:resource` in a Solid pod via the autonomous Solid Agent, such that the state of our smart home device is stored in our PDS and external systems can interact with this store to actuate our home environment.

[2] https://en.wikipedia.org/wiki/Zigbee
[3] https://www.openhab.org/

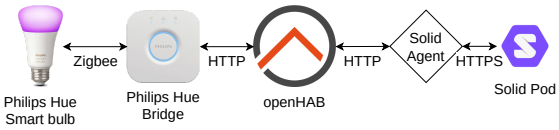

Figure 1: Setup of the demonstration: The Solid Agent synchronises the state of the Smart Bulb, through openHAB with a resource in the Solid Pod.

3.1. Architecture of the Solid Agent

Figure 2 shows the architecture of our agent. For external observers, the Solid Agent acts as one agent. However, internally it consists of multiple individual actors working together, identifiable with their own WebID[4]. We implemented two interfacing actors that percept and actuate external things on the web: the **OpenHAB Actor** and the **Solid Actor**. At the core, the **Orchestration Actor** routes the inputs from the interfacing actors to the reasoning engine and executes actions on the resulting output. These actions are actuated by calling other actors such as, but not limited to, the above-mentioned actors.

The **OpenHAB Actor** monitors an OpenHAB platform for changes in the state of items. The following functions were implemented: (i) retrieve the state of an item, (ii) map an item state object to an RDF representation of the said item, and (iii) subscribe to an item. The agent can be actuated directly to change the state of an item. To accomplish this actuation, two additional functions are implemented: (i) map an RDF representation of the state of an item to an item state object, and (ii) store the state of an item.

The **Solid Actor** works in the environment of Solid Pods. This agent monitors an `ldp:resource` for read/write updates.

The **Orchestration Actor** implements an *Interface*, *Reactive* and *Controller* component [1]. The *Interface* component consists of a sensing part through an `ldp:inbox` and an actuation part via HTTP. The inputs are in the form of ActivityStreams2 (AS2) [6] payloads and are either for the initialisation of the agent or from other internal actors that communicate updates to the agent. The *Reactive* component consists of the EYE [7] reasoning engine, a state knowledge base and `IF-THEN` Rules written in N3. The *Controller* component forms the glue between the other components in this actor. It retrieves a payload from the *Interface* component and passes this payload as input to the Reactive component which results in a set of zero, one or more actions. When applicable, this set of actions is forwarded to the *Interface* component to be actuated. The implementation of this actor is based on the KoreoGrafeye[5] library.

3.2. Initialising the Solid Agent for a synchronisation task

Given its generic design, the agent must be configured to autonomously execute its tasks. An AS2 payload must be sent to the Inbox of the Solid Agent containing: (i) a reference to the Solid Actor and the location of the state resource, (ii) the openHAB Actor together with the openHAB endpoint, an access token and the smart home appliances and; (iii) the task of the agent, which is formulated as a set of *condition-action rules*. A complete example of this configuration and a

[4]https://www.w3.org/2005/Incubator/webid/spec/identity/
[5]https://github.com/eyereasoner/Koreografeye

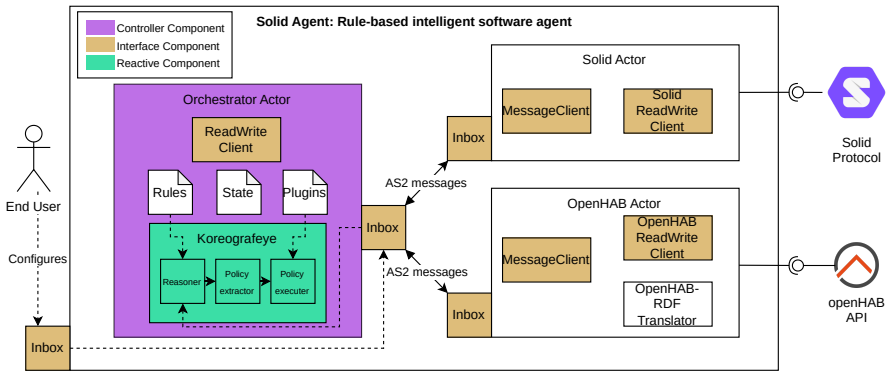

Figure 2: An Orchestration Actor (left), a Solid Actor (top right) and an openHAB Actor (bottom right) compose the agent's architecture. Message Clients in Interface Components are used to subscribe to resources in the environment (e.g. a state resource on a Solid pod).

video demonstration can be found in the Solid Agent repository[6]. This Inbox makes possible agent-to-agent communication, thus enabling the implementation of MAS systems.

4. Conclusion

In this paper, we presented a web agent demonstration that incorporates several components from the hybrid architecture proposed in [1]. Our implementation supports condition-action N3 rules and interacts, among others, with Solid PDSs. The generic nature of our reference implementation aims for re-usability and to enable researching more complex MAS scenarios. In future work, we target the evaluation of the approach and the implementation of the missing architectural components.

Acknowledgements

Supported by SolidLab Vlaanderen (Flemish Government, EWI and RRF project VV023/10).

References

[1] Sabrina Kirrane, Intelligent software web agents: A gap analysis, Journal of Web Semantics 71 (2021) 100659. Publisher: Elsevier.
[2] T. Berners-Lee, D. Connolly, L. Kagal, Y. Scharf, J. Hendler, N3logic: A logical framework for the world wide web, Theory and Practice of Logic Programming 8 (2008) 249–269.
[3] T. Käfer, A. Harth, Rule-based Programming of User Agents for Linked Data, in: Proceedings of the 11th Workshop on Linked Data on the Web, 2018.
[4] D. Schraudner, V. Charpenay, An HTTP/RDF-Based Agent Infrastructure for Manufacturing Using Stigmergy, in: The Semantic Web: ESWC 2020 Satellite Events, Lecture Notes in

[6] https://zenodo.org/record/8338026

Computer Science, Springer International Publishing, Cham, 2020, pp. 197–202. doi:10.1007/978-3-030-62327-2_34.

[5] A. Zimmermann, A. Ciortea, C. Faron, E. O'Neill, M. Poveda-Villalón, Pody: a Solid-based Approach to Embody Agents in Web-based Multi-Agent-Systems, in: 11th International Workshop on Engineering Multi-Agent Systems (EMAS2023), 2023.

[6] J. Snell, E. Prodomou, Activity Streams 2.0, 2017. URL: https://www.w3.org/TR/activitystreams-core/.

[7] R. Verborgh, J. De Roo, Drawing Conclusions from Linked Data on the Web: The EYE Reasoner, IEEE Software 32 (2015) 23–27. doi:10.1109/MS.2015.63, conference Name: IEEE Software.