WAT: Autonomous Hypermedia-driven Web Agents for Web of Things Devices

Mahda Noura[a], Valentin Siegert[a] and Martin Gaedke[a]

[a]Technische Universität Chemnitz, Chemnitz, Germany

Abstract

To address the All the Agents Challenge (ATAC), we developed a solution for an Industry 4.0 use-case scenario that allows autonomous agents to reason on knowledge graphs, perceive, decide and act for reaching their goals, and coordinate the whole process in their Web of Things (WoT) environments. Our approach integrates the research works performed in the WoT, Semantic Web and the Multi-agent System (MAS) communities. Finally, We discuss the engineering properties of the proposed solution.
Video: **https://youtu.be/czM4L_0AeB4/**
Source Code: **https://github.com/ValentinSiegert/WAT**

Keywords

Multi-Agent System, Web of Things, Semantic Web, Knowledge Graph

1. System Overview

In this section, we present the approach for a multi-agent system (MAS) that can monitor and control a process to achieve a goal with the inclusion of physical devices a.k.a "Things". In our pursuit to address the main technological requirements of the ATAC challenge, we design and develop our solution based on *Hypermedia MAS* [1] where hypermedia is used for uniform interaction among heterogeneous entities. Figure 1 shows a high-level architectural view of the different components that constitute the system.

The *initializer* is responsible for initializing the set of *agents* and the *DomainArtifacts* of the process based on a domain specific language description (cf. marker 1 in Figure 1). As the *ThingArtifacts* will only be created by the agents during run time, the initializer does not handle their setup. The *agents* are the entities that can make decisions on the goals, the actions that are executed on the system artifacts to achieve a goal and on the interactions among the agents. The autonomous agents abstraction in our system is based on *Belief-Desire-Intention* (BDI) agents [2] – a type of cognitive agent described in terms of mental behaviors: *beliefs*- knowledge that the agent believes, *desire*- the desired state to be achieved by the agent, and *intention*- a sequence of steps used to achieve a desire.

All the Agents Challenge (ATAC 2021)
mahda.noura@informatik.tu-chemnitz.de (M. Noura); valentin.siegert@informatik.tu-chemnitz.de (V. Siegert); martin.gaedke@informatik.tu-chemnitz.de (M. Gaedke)
https://vsr.informatik.tu-chemnitz.de/about/people/mahdanoura/ (M. Noura);
https://vsr.informatik.tu-chemnitz.de/about/people/siegert/ (V. Siegert);
https://vsr.informatik.tu-chemnitz.de/about/people/gaedke/ (M. Gaedke)
0000–0002–5105–2463 (M. Noura); 0000–0001–5763–8265 (V. Siegert); 0000–0002–6729–2912 (M. Gaedke)

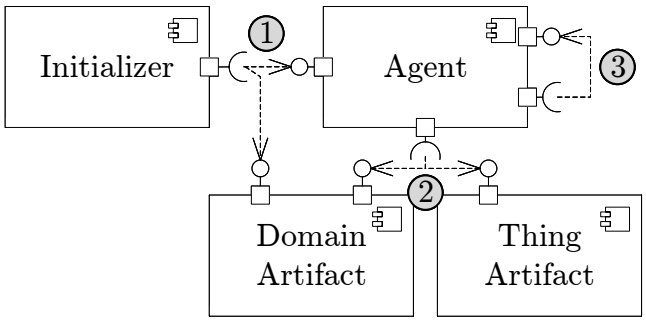

Figure 1: WAT Architecture with Components

The agents can use a set of controllable and observable *artifacts* to reach their goals (cf. marker 2 in Figure 1). The artifacts can be physical entities (*ThingArtifact*) that can sense and control the environment (e.g., devices in a manufacturing line) or domain entities (*DomainArtifact*) that enrich the conceptual knowledge or accomplishment of agents and their goals (e.g., knowledge graph crawler, AI planner). The `ThingArtifacts` are described uniformly via the W3C Things Description (WoT TD) model [3] which provides interoperability between devices. The WoT TD exposes property affordances (readProperty/writeProperty) and action affordances (invokeAction) that can be be consumed by the agents in the system to access the Things and control it. Each agent can create different `ThingArtifacts` which can also be accessed by others.

The agents use the `DomainArtifact` *KnowledgeGraphCrawler* to navigate the hypermedia environment and update their beliefs about the environment. This artifact discovers the entities in the MAS and the interaction mechanism (e.g., HTTP) exposed by the resources. To enable agents decide autonomously, all the resources (e.g., production line, Things, etc.,) and the relations between them are described semantically in RDF using domain-specific vocabularies. The knowledge graph uniquely identifies every thing instantiated using Internationalized Resource Identifiers (IRIs). Thus, agents can perceive specifications of the real things during run time to create the respective `ThingArtifacts`. To fulfill the purpose of a given use case the agents' desires can be defined either manually by an engineer or produced automatically using an automated AI planner artifact [4].

To achieve the main goal of the use case, e.g. producing filled yogurt cups, the agents should have social ability and interact with one another. In this work, the agents can communicate with each other via messages (cf. marker 3 in Figure 1). Such an interaction always involves two agents exchanging one message with each other. Broader message exchange patterns like interactions with several agents or even broadcasts to all, require the agents to setup messages to all responsible others or use a `DomainArtifact` like a pin board or similar. The messages themselves are use-case specific but can include: (1) message type, (2) sender's identifier, (3) either an identifier on a property to interact with or an action to invoke, and (4) any content or reason part to transfer details about the property or action. Therewith, the agents' interactions with both agents and artifacts can be modeled.

2. Use-case Scenario: Manufacturing Product Line

The application scenario that we use in this challenge is a simulated manufacturing product line in the Industry 4.0 community for filling and packaging yogurt[1]. In this scenario, the factory is composed of different workshops which have different goals: *conveying workshop*, *filling workshop*, *potting workshop* and the *packaging workshop*. Each workshop includes different Things that can be used to sense the current state of the environment using sensors (e.g., presence sensor) and perform actuations on actuators (e.g., robotic arm). The factory also includes external suppliers which are responsible for providing empty yogurt cups, yogurt supplies and package providers for the fulfilment of an order.

In this scenario, the conveying workshop initially, loads the storage rack with yogurt cups, places an order for picking a cup from the shelves, picks up the cups using a Cartesian robot on X/Z axes, places the cups on a conveyor belt and finally the cups move to the head of the conveyor belt. In the filling workshop, when a cup is identified below the filler, subsequently the magnetic valve of the filler opens and the robot starts to follow the moving cup on the X axis. While the cup is moving, yogurt is poured into the cup. When the cup is filled with a certain amount of yogurt, the valve closes and the robot returns to its initial X coordinate while the container keeps moving towards the end of the conveyor. The main responsibility of the potting workshop is to grasp the filled yogurt pots and release them to the next workshop. For this, we use a Bosch APAS robot which moves in six different directions and can control grasping/releasing actuations via the built-in camera. The packaging workshop uses a Cartesian robot on the X/Y axes to pick up the cups form the conveyor belt and place them in an empty package. Once the package is filled with six cups, the Cartesian robot places the package on the next conveyor belt and subsequently fetches another empty package from the package buffer.

3. Implementation and Demonstration

WAT was developed as an extension to the AI for Industry 4.0 summer school[2]. We demonstrate a prototype of WAT within the above-mentioned use-case scenario implemented in Python. The source code is available on GitHub[3]. Also, a video of our simulated demonstrator is accessible on Youtube[4]. The demonstrator consists of six autonomous agents which correspond (1) to the cup provider, (2) dairy product provider, (3) vl10 agent for controlling the conveying workshop, (4) dx10 agent for controlling the filling workshop, (5) apas agent for controlling the robotic arm, and (6) the xy10 agent for controlling the packaging workshop. The factory workspace contains seven artifacts, where six of them are modelled as `ThingArtifact` that the agents can observe and act on, and a `LinkedDataFuSpider` representing the `KnowledgeGraphCrawler` by using Linked Data-Fu [5] to discover all related data in the knowledge graph using HTTP requests, starting from an entry point based on condition-action-rules and inference rules. The interference rules are predefined per thing to obtain all necessary information. Therefore, all the resources in

[1] https://gitlab.emse.fr/ai4industry/hackathon/-/wikis/scenario/
[2] https://ai4industry2021.sciencesconf.org/
[3] https://github.com/ValentinSiegert/WAT
[4] https://youtu.be/czM4L_0AeB4/

our system and their relations have been modelled in RDF using the following domain-specific vocabularies:

- the W3C WoT TD [3] for describing device capabilities and their interaction affordances
- the W3C SSN[5] and SOSA ontology for describing sensor and observation
- the SAREF4SYST[6] ontology for describing the connection between the subsystems in the production line
- product type ontology
- custom ontology for describing scenario specific vocabularies
- Hypermedia Controls Ontology[7] for describing links and forms

Each agent and artifact is represented as a process and agents exchange messages via multi-processing pipes for demonstration purposes. The multi-processing pipes can be exchange with any other FIFO messaging exchange technique. For the demonstrator, the following modules were used from the AI for Industry 4.0 summer school[2] resources:

- use-case scenario applied to WAT
- graphical user interface of the factory for demonstration purposes
- factory resources described in the knowledge graph for reasoning

4. Related Work

JaCaMo [6] integrates three platforms (Jason, CArtAgO and Moise) to enable multi-agent oriented programming with agent, organisation and environment oriented programming. On the other hand, Hypermedia MAS [7] focuses only on the interaction between the main concepts of Jacamo by using Hypermedia as the Engine of Application State (HATEOAS). The main aim is to provide a uniform way to discover agents and artifacts and interact with them. In the semantic web community, Linked Data-Fu [5] is designed based on Abstract State Machines which combines HTTP with RDF for reading and writing linked data at a large scale.

One of the applications that integrates the above-mentioned approaches to provide a Linked Data interface for a simulated building scenario is the Building on Linked Data (BOLD)[8]. Similar to our approach, the agents can use hypermedia to browse the environment and discover Things via LDFU utility.

Schraudner et.al. [8], proposed to use the concept of stigmergy for indirect communication between simple-reflex agents. In this work, the agents directly communication with each other by passing messages. Stigmergy can be used to further improve the separation of concerns. In contrast, in this work we also consider integrating WoT devices with MAS and semantic web. As a summary, we provide the following contributions in relation to the literature:

- We present a solution for a manufacturing system that integrates multi-agent systems, semantic web and WoT devices based on Hypermedia MAS [7], which leverages Linked Data-Fu [5].

[5] https://www.w3.org/TR/vocab-ssn/
[6] https://saref.etsi.org/saref4syst/v1.1.2/
[7] https://www.w3.org/2019/wot/hypermedia
[8] https://github.com/bold-benchmark/bold-server

- We developed the prototype of WAT in Python to showcase the feasibility of implementing MAS in combination with WoT, and Semantic Web with one of the most commonly-used programming languages. Additionally, the programming paradigms of JaCaMo [6] is still supported, even though the use case does not highlight organisational oriented programming.
- The proposed modular architecture style in this work, enables web agents (1) to interact with Things in the real world and achieve goals within dynamic environments and (2) to be integrated with microservices within the web. This could be achieved by distributing agents and artifacts on different host with TCP server.

5. Discussion

In the following we discuss the proposed approach in terms of different engineering properties:
Heterogeneity: The proposed system is manually described by a domain engineer according to different vocabularies such as SSN/SOSA, SAREF, etc., for decoupling on the semantic level and based on W3C WoT TD for decoupling from specific device APIs. The production rules and inference rules developed for crawling the knowledge graph are independent of the specific production line and the devices involved. An update on the production line does only require to revise the knowledge graph with the descriptions of the production line and a developer needs to provide the agents plan for reaching a specific goal.
Extensibility: The system is developed on top of WoT, which means that it can be easily extended with any Thing that exposes it's capabilities with the WoT TD.
Modularity: The overall system is developed with different components for the agents and the different types of artifacts. This means that new agents and artifacts can be easily added or replaced without affecting the overall system.
Reusability: By relying on components, they can be reused in different use cases with new goals, as well as with different `ThingArtifacts` or `DomainArtifacts`.
Scalability: The system is scalable, as it can spawn many agents and artifacts using processes. However, we have not done experiments to show the performance. This can be further improved by distributing agents and artifacts to different hosts with an own TCP server and enable the interactions to be transferred via the network. Therefore, the agents can also be integrated with the microservices architecture.
Dynamicity: The system supports dynamic behaviour as it can identify the different production line components, their properties, and the access method dynamically based on linked data. Also, the current state of the machines are always retrieved from sensor data and depending on them the system continually evolves and reacts to changes. For example, depending on the sensed amount of yogurt in the filling machine, new yogurt is ordered from a third party yogurt provider. The current solution's dynamicity is limited in terms of unexpected behaviours such as device failure. In a production system, an agent's goal relies on the Thing's availability, while the WoT devices themselves are dynamic and could join or leave the network at arbitrary times during execution, making their availability unpredictable at runtime.

References

[1] O. Boissier, A. Ciortea, A. Harth, A. Ricci, Autonomous Agents on the Web (Dagstuhl Seminar 21072), Dagstuhl Reports 11 (2021) 24–100. doi:10.4230/DagRep.11.1.24.

[2] A. S. Rao, M. P. Georgeff, et al., Bdi agents: From theory to practice, in: Icmas, volume 95, 1995, pp. 312–319.

[3] S. Kaebisch, T. Kamiya, Web of things (wot) thing description w3c working draft 5 april 2018, W3C Working Draft, W3C (2018).

[4] M. Noura, M. Gaedke, An automated cyclic planning framework based on plan-do-check-act for web of things composition, in: Proceedings of the 10th ACM Conference on Web Science, 2019, pp. 205–214.

[5] T. Käfer, A. Harth, Rule-based programming of user agents for linked data, in: LDOW@ WWW, 2018.

[6] 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.

[7] A. Ciortea, O. Boissier, A. Ricci, Engineering world-wide multi-agent systems with hypermedia, in: International Workshop on Engineering Multi-Agent Systems, Springer, 2018, pp. 285–301.

[8] D. Schraudner, V. Charpenay, An http/rdf-based agent infrastructure for manufacturing using stigmergy, in: European Semantic Web Conference, Springer, 2020, pp. 197–202.