Escaping the Streetlight Effect: Semantic Hypermedia Search Enhances Autonomous Behavior in the Web of Things

Simon Bienz, Andrei Ciortea, Simon Mayer, Fabien Gandon, Olivier Corby

Escaping the Streetlight Effect: Semantic Hypermedia Search Enhances Autonomous Behavior in the Web of Things

Simon Bienz
Department of Computer Science
ETH Zürich
Zürich, Switzerland
bienzs@ethz.ch

Andrei Ciortea
University of St. Gallen
St. Gallen, Switzerland
Inria, Université Côte
d'Azur, CNRS, I3S
Sophia Antipolis, France
andrei.ciortea@unisg.ch

Simon Mayer
University of St. Gallen
and ETH Zürich
St. Gallen, Switzerland
simon.mayer@unisg.ch

Fabien Gandon
Inria, Université Côte
d'Azur, CNRS, I3S
Sophia Antipolis, France
fabien.gandon@inria.fr

Olivier Corby
Inria, Université Côte
d'Azur, CNRS, I3S
Sophia Antipolis, France
olivier.corby@inria.fr

ABSTRACT

The integration of systems of autonomous agents in Web of Things (WoT) environments is a promising approach to provide and distribute intelligence in world-wide pervasive systems. A central problem then is to enable autonomous agents to discover heterogeneous resources in large-scale, dynamic WoT environments. This is true in particular if an environment relies on open-standards and evolves rapidly requiring agents to adapt their behavior to achieve their goals. To this end, we developed a search engine for the WoT that allows autonomous agents to perform *approximate search queries* in order to find relevant resources in their environment in *(weak) real time*. The search engine crawls dynamic WoT environments to discover and index device metadata described with the W3C WoT Thing Description, and exposes a SPARQL endpoint that agents can use for approximate search. To demonstrate the feasibility of our approach, we implemented a prototype application for the maintenance of industrial robots in world-wide manufacturing systems. The prototype demonstrates that our semantic hypermedia search engine enhances the flexibility and agility of autonomous agents in the WoT.

KEYWORDS

Autonomous Agents, Hypermedia, Search, Semantic Web, Web of Things

ACM Reference Format:
Simon Bienz, Andrei Ciortea, Simon Mayer, Fabien Gandon, and Olivier Corby. 2019. Escaping the Streetlight Effect: Semantic Hypermedia Search Enhances Autonomous Behavior in the Web of Things. In *IoT 2019: International Conference on the Internet of Things, October 22–25, 2019, Bilbao, Spain.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

To counteract the fragmentation of the Internet of Things (IoT), the Web of Things (WoT) enables devices and digital services to interoperate at the application layer. Furthermore, the WoT lowers the entry-barrier for the development of IoT applications: developers can use standard Web technologies to create and execute *mashups of devices and digital services* – a.k.a. *physical mashups* [11]. However, the common practice of manually defining and maintaining physical mashups (e.g., via Node-RED[1] or IFTTT[2]) is not always practical. WoT systems are often required to *evolve rapidly* as the availability of devices – and their services – fluctuates. This is particularly true for constrained devices that are often duty-cycled, and for mobile devices (as well as people) that physically move between spatial domains together with the (localized) services they provide. These inherent dynamics can make a WoT system evolve every few seconds or even faster. In such settings, static physical mashups become impractical: once deployed, they cannot adapt to dynamic environments. The W3C WoT Thing Description (TD) helps mitigate these limitations to great extent through *interaction affordances* and *hypermedia controls* [16]: it allows physical mashups to be defined in terms of abstract interaction patterns rather than specific protocols and device APIs. The resulting physical mashups are then more flexible as they are loosely coupled to the underlying device APIs, but they still have to be defined and maintained manually. Furthermore, manually "wiring" the WoT cannot scale well to large numbers of heterogeneous devices [7, 21]. Ideally, WoT systems would be able to adapt to large and dynamic settings in an *autonomous* manner – with minimal human intervention.

Such autonomous systems have been studied to large extent in the scientific literature on distributed artificial intelligence (AI) and, in particular, *multi-agent systems (MAS)* [33]. In the past, we have

[1] http://nodered.org/
[2] http://www.ifttt.com/

shown that MAS research already provides models, programming paradigms, languages, and tooling that can be used to engineer more adaptive WoT systems (e.g., see [6]). However, autonomous agents operating in such systems need to make decisions based on the systems' current state: they need to find and use available resources in real time and with minimal out-of-band information in order to achieve their goals in an *autonomous* and *flexible* manner. We hypothesize that, similarly to how people need hypermedia search engines to find relevant resources on the Web in order to achieve their *everyday goals* (online shopping, travel planning, etc.), autonomous agents will also require *hypermedia search engines* to help them achieve their goals in the WoT. This analogy is particularly relevant in the context of the W3C WoT[3], which introduces *hypermedia controls* as fundamental building blocks.

Hypermedia search is still insufficiently investigated in the WoT. A common solution for resource discovery in the WoT is the use of directories, such as the *CoRE Resource Directory*[4] or the *Thing Directory*[5]. Autonomous agents could then query directories individually or as a federation. However, existing approaches for federated query processing assume that the complete federation is known beforehand [1] – an assumption that fails in an *open and dynamic* WoT. Autonomous agents would thus be subjected to the "streetlight effect"[6]: they would have to know beforehand where to look for the resources required to achieve their goals. Hypermedia search, on the other hand, enables the *serendipitous* discovery and use of resources on the Web – an important property for sustaining long-lived systems.

1.1 Application Scenario: Maintenance of World-Wide Manufacturing Systems

For illustrative purposes, we introduce a concrete application scenario that we use throughout the rest of this paper: the maintenance of industrial robots in *world-wide* manufacturing systems – in which production sites are distributed across the globe [6]. For our scenario, we consider two types of robots: *manufacturing robots* and *maintenance robots*. *Manufacturing robots* might require maintenance tasks, and they can delegate such tasks either to *maintenance robots* or to *maintenance engineers*. To this end, the manufacturing robots thus have to find in *real time* what *heterogeneous resources* (robots or engineers) are available across production sites, and to decide what maintenance tasks can (and should) be fulfilled by a robot and what tasks would require an engineer – considering also their locality (e.g., robots are immobile, while engineers can travel between production sites).

The above scenario highlights several aspects that are important for searching the WoT. First, unlike the documentary Web, the WoT is populated with *non-textual, non-information resources* that cannot be simply indexed and ranked based on term frequency [20]. Rather it would be more appropriate to index *semantic descriptions* of such resources, which can also capture relevant *contextual information* (e.g., the current location of a maintenance engineer). Second, given the dynamicity of WoT systems, support for *real-time search* is required – a topic that has already been explored in early research on searching the WoT (e.g., [27]). Third, to unlock the full potential of the WoT, it is necessary to discover and query resources at Web scale – as in the *world-wide* manufacturing systems in our scenario.

[3]https://www.w3.org/TR/wot-architecture/
[4]https://tools.ietf.org/html/draft-ietf-core-resource-directory-23
[5]https://github.com/thingweb/thingweb-directory/, accessed: 08.09.2019.
[6]https://en.wikipedia.org/wiki/Streetlight_effect, accessed: 08.09.2019.

1.2 Contribution and Paper Outline

We developed a search engine for the WoT that allows autonomous agents to perform *approximate search queries* in *(weak) real time* in order to find resources in their environment that are required to achieve their goals. The search engine crawls WoT environments and keeps track of their evolution in order to discover device metadata described with the W3C WoT TD. The discovered data is indexed and exposed to clients via a SPARQL endpoint that can process approximate queries. The search engine is based on *Corese* [9], an open-source inference and query engine for Linked Data, and our own implementation of a hypermedia crawler for dynamic WoT environments.

To demonstrate the feasibility of our approach, we implemented a demonstrator based on the maintenance scenario presented in the previous section. The demonstrator shows that our search engine allows agents to cope better with dynamic WoT environments and to pursue their goals in a more flexible and agile manner – therefore *enhancing their autonomous behavior* in WoT environments.

This paper is structured as follows. We discuss foundational technologies of our system from MAS and WoT research as well as relevant related work on searching WoT systems in Section 2. We then give an overview of the design and implementation of our system in Section 3. We describe our demonstrator deployment in detail in Section 4 and provide a discussion of our approach, including its limitations, in Section 5.

2 BACKGROUND AND RELATED WORK

In this section, we first introduce several concepts from MAS research that we use throughout the rest of this paper – with a focus on defining a conceptual bridge between MAS and WoT systems. We then discuss related work on searching the WoT.

2.1 From Multi-Agent Systems to Autonomous WoT Systems

In AI research, an *agent* is commonly defined as an entity "situated in some environment, that is capable of flexible autonomous action in order to meet its design objectives" [15]. *Autonomy* is central to this definition and refers to the agent's ability to operate on its own, without the need of direct intervention from people or other agents. The agent is situated in an *external environment* that it can perceive via sensors and influence via actuators. A distinctive feature of an autonomous agent is its *flexibility* in the pursuit of some design objectives [15]: the agent is *reactive* by responding to changes in the environment in a timely fashion, *proactive* by exhibiting goal-directed behavior and taking the initiative when appropriate, and *social* by interacting with humans or other agents in order to achieve complex tasks that would otherwise overcome its own capabilities. In distributed AI, a *multi-agent system (MAS)* is then a system conceptualized in terms of agents that are situated in a *shared environment* and interact with one another to achieve their design objectives [15, 33].

Agent-oriented programming was first articulated as a paradigm in [28], but its origins can be traced back to the mid 1980's [10]. A well-known meta-model for designing and programming MAS – that we use in our approach – is *Agents & Artifacts (A&A)* [26]. A&A considers the agents' *environment* as a first-class abstraction in the MAS: a component designed and programmed with clear-cut responsibilities, such as mediating interaction among agents or access to the deployment context (devices, digital services, etc.). The *environment* is modeled as a dynamic set of *workspaces*, where a workspace is a dynamic set of *artifacts*. An artifact is a computational object that exposes:

- *observable properties*: state variables that can be perceived by the agent;
- *observable events*: non-persistent, fire-and-forget signals that carry information and can be perceived by the agent;
- *operations*: environment actions provided to the agent; operations can change the values of observable properties or they can trigger events.

The set of all interactions an agent can have with its environment is determined by the artifacts available at run time. Agents use artifacts in pursuit of their goals, and they can create or destroy artifacts at run time.[7]

It is worth to note the similarity between the *artifact* model defined by A&A and the *W3C WoT TD* [16] standardized in the W3C WoT Working Group. Both models define three types of interaction patterns, namely *observable properties*, *observable events*, and *operations* or *actions*, with the W3C WoT TD model being slightly more generic: a TD can expose writable properties, whereas artifact properties are read-only. Applying the WoT TD to decouple *artifacts* from *devices* is thus straightforward and provides a *conceptual bridge* for deploying MAS in WoT environments [6] – and thus to engineer *autonomous WoT systems*.

Recent work further explored this conceptual bridge by applying hypermedia to the engineering of MAS to define *Hypermedia MAS* [4]: socio-technical systems composed of people and autonomous agents situated in a shared hypermedia environment distributed across the open Web. Similar to how people use the Web, agents (both human and non-human) would then navigate the distributed hypermedia environment to discover artifacts and other resources that they could use to achieve their goals. A central problem then is providing agents with the search facility that would allow them to find relevant resources in an efficient manner [5].

2.2 Searching the Web of Things

There is already a considerable body of research on searching the IoT/WoT and several surveys are available, such as [27] and the more recent [35]. The latter, in particular, provides an extensive review of search techniques for the WoT.

Keyword-based search techniques for the WoT (e.g. [31, 32, 34]) typically target human users – given that choosing meaningful keywords is then of central importance. These techniques are thus less suitable for machines (or autonomous agents). Other search techniques rely on location-based clustering – often in combination with keyword- or tag-based search – and follow the assumption that in the WoT there is a high degree of locality of interactions among human users and devices (e.g. [18, 20]). In another approach, *Dyser* [23] focuses on real-time search given dynamic sensor readings in WoT environments and uses statistical models to predict the state of registered resources: these models induce a ranking on known resources that determines which are contacted first by the engine to find out whether their *actual* current state matches the user query.

More recent approaches include directory-based discovery mechanisms, such as the *CoRE Resource Directory*[8] or the *Thing Directory*[9], which could be queried individually or as a federation. Most interestingly, Thing Directories store device metadata described using the W3C WoT TD [16] and can expose SPARQL endpoints. Nevertheless, existing approaches for federated SPARQL query processing assume that the complete federation is known beforehand [1] – an assumption that fails in an *open and dynamic* WoT. Hypermedia-based discovery via crawling, on the other hand, has proven practical on the open Web.

A crawling-based mechanism for WoT devices was proposed in DiscoWoT [19], which allows WoT devices to be crawled to discover their properties and any exposed interfaces. However, DiscoWoT assumes that an *entry point* for the WoT device to be crawled (e.g., its IRI) is known beforehand. The SPITFIRE architecture [24] suggested crawling the (semantic) WoT periodically to discover devices and other resources, but the crawling process is not discussed in detail – and given the dynamicity of WoT systems, periodical crawling seems impractical.

An approach aiming to crawl the WoT at Web-scale was proposed in WOTS2E [17], which uses metacrawling (i.e., it relies on popular search engines such as Google or Bing) to discover SPARQL endpoints that contain IoT/WoT-related datasets and ontologies. WOTS2E focuses on the global discovery of relevant SPARQL endpoints (rather than individual devices), which is complementary to our proposal (see also Section 5).

To the best of our knowledge, a search engine for dynamic WoT environments that would allow machines to perform approximate queries in real time is not yet available.

3 SYSTEM ARCHITECTURE

We developed a search engine that allows autonomous agents to perform *approximate queries* in *(weak) real time* in order to find resources in their WoT environments that are required to achieve their goals. To achieve this, our approach integrates results from research on the WoT, MAS, and the Semantic Web.

Figure 1 depicts an overview of our system. Following our discussion in Section 2.1, we design and program the system as a *Hypermedia MAS*: people and autonomous agents are situated in a distributed hypermedia environment, and we model devices and any other tools that agents use to achieve their goals as *artifacts* in this environment. All entities in our system (agents, artifacts, workspaces, etc.) are represented as Web resources described in RDF and projected into the distributed hypermedia environment, which enables their *system-wide discovery via crawling*. We present the main components of our system in what follows.

[7] We refer interested readers to [26] for further details on the A&A meta-model.

[8] https://tools.ietf.org/html/draft-ietf-core-resource-directory-23

[9] https://github.com/thingweb/thingweb-directory/, accessed: 08.09.2019.

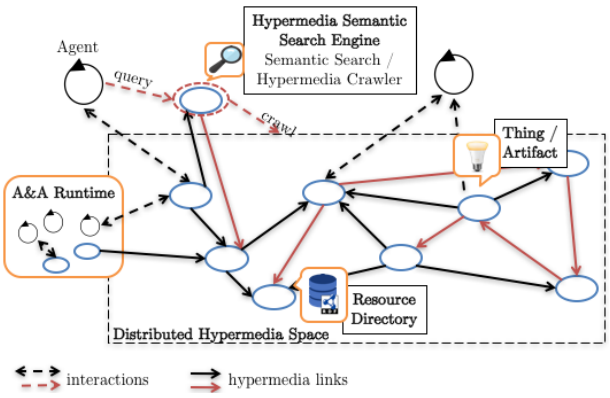

Figure 1: Conceptual overview of our system.

3.1 Agents & Artifacts Container

We use an *Agents & Artifacts (A&A) Container* for programming and running agents and artifacts in W3C WoT environments. The *A&A Container* is developed using JaCaMo [2], a MAS platform that includes the reference implementation for the *A&A meta-model* (see Section 2.1). JaCaMo provides developers with a customizable architecture for cognitive agents, a language for programming cognitive agents, and a Java-based framework for programming artifacts according to *A&A*.

3.1.1 Agent architecture. We model the autonomous agents in our system as *Belief-Desire-Intention (BDI)* agents [3] – a type of cognitive agent designed and programmed in terms of *mental attitudes*: *beliefs* held about the world, *goals* desired to be achieved, and *plans* used to achieve goals. The BDI agent architecture thus provides developers with a formal "human-oriented" level of abstraction that facilitates designing, programming, but also inspecting and debugging autonomous behavior[10] – and therefore facilitates the engineering of *autonomous WoT systems*. Another important feature of the BDI agent architecture that makes it a good fit for our approach is that it can balance goal-directed and reactive behavior: BDI agents commit to goals by executing plans, but they can still react to events and changes in the environment while executing their plans.

The typical *program* of a BDI agent is composed of the agent's initial sets of beliefs, goals, and plans – all of which can evolve at run time. Multiple languages and frameworks are available for programming BDI agents. One of the most prominent agent programming languages is AgentSpeak(L) [25] and its more recent extended version known as Jason [3]. Jason is the language used in the JaCaMo platform on which our *A&A Container* is based. BDI agents in our system are equipped with libraries of *Jason plans*, where a Jason plan has the form:

```
triggering_event : application_context <- plan_body .
```

For illustrative purposes, Listing 1 shows an extract from a Jason program used by a *maintenance agent* in our demonstrator (see Section 4 for details). The `!start` goal on line 9 in Listing 1 is the entry point into our agent program. Depending on the evolution of the system at run time, the `!start` goal may eventually lead to the sub-goal `!notify_engineer(...)` to be created – if the agent decides it is necessary to notify an on-site engineer of a malfunction (cf. scenario in Section 4.2). In our demonstrator, malfunctions can be signaled visually to on-site engineers via light bulbs. The creation of the `!notify_engineer(...)` sub-goal would then trigger the execution of the plan on lines 13-20: the agent turns on a light bulb with a given color code for 2 seconds, and then turns off the light bulb (see also the next section). The initial set of beliefs in our agent program includes, among others, the CIE 1931 XY color codes [30] to be used when notifying on-site engineers (lines 2-3).[11]

Listing 1: Extract from the Jason program of a maintenance agent in our demonstrator. The IoT Schema IRIs in this listing are used only for illustrative purposes.

```
1  /* Initial beliefs */
2  green_color(0.409, 0.518).
3  red_color(0.4, 0.2).
4  // (...)
5
6  /* Initial goals */
7  // Initializes the agent, and may lead to the
8  // "!notify_engineer" sub-goal being created
9  !start.
10
11 /* Plans */
12 // (...)
13 +!notify_engineer(ArtifactName,CIEx,CIEy) : true <-
14   act("http://iotschema.org/SetColour", [
15     ["http://iotschema.org/CIExData", CIEx],
16     ["http://iotschema.org/CIEyData", CIEy]
17   ])[artifact_name(ArtifactName)];
18   .wait(2000);
19   act("http://iotschema.org/TurnOff", [
20   ])[artifact_name(ArtifactName)].
```

3.1.2 Infrastructure artifacts. The *A&A Container* provides agents with several types of infrastructure artifacts that they can use: *browser artifacts*, *crawler artifacts*, and *finder artifacts*.

Browser artifacts serve as facades that allow agents to interact with artifacts discovered at run time in their hypermedia environment as they would interact with any artifact in a local *workspace* (see Section 2.1) on the JaCaMo platform. Browser artifacts are instantiated with IRIs of W3C WoT TDs, and we refer to them as "browser" artifacts because they perform functions similar to Web browsers: they retrieve and parse W3C WoT TDs, expose *interaction affordances* to agents, and translate agents' actions to interactions with the *Web Thing* [16] being described (e.g., via HTTP or CoAP). Unlike regular JaCaMo artifacts, browser artifacts expose metadata (e.g., about the *supported types of actions provided to agents*) via *observable properties*. To perform actions, such as the action of changing the color of a light bulb in Listing 1 (line 14), agents use a generic `act` operation provided by the browser artifact. The `act` operation takes as arguments the IRI of the action type to be executed as well as IRIs of any required parameter types specified in the W3C WoT TD used by the browser artifact. If the W3C WoT TD provides multiple hypermedia controls for the same action type, the first hypermedia control is used.

Agents can use *crawler artifacts* to configure the search engine, for instance by providing seeds to be crawled or by setting the link types to be followed when crawling (see Section 3.3.1), and they

[10] The BDI architecture is the mainstream architecture for cognitive agents in MAS research.

[11] The color code values used in our demonstrator correspond to the Philips Hue API.

Table 1: Prefix bindings.

Prefix	IRI
td	http://www.w3.org/ns/td#
stn	http://w3id.org/stn/core#
eve	http://w3id.org/eve#
ex	http://exammple.org/#

can use *finder artifacts* to perform search queries (see Section 3.3.2). The role of *crawler* and *finder artifacts* is to simplify the agents' logic by encapsulating all logic required to access the HTTP and SPARQL endpoints exposed by our semantic hypermedia search engine.

3.2 Distributed Hypermedia Environment

All resources in our system (agents, workspaces, artifacts, devices, etc.) and relations among them are described in RDF using:[12]

- the W3C WoT TD [16] for describing devices and other artifacts together with the interaction affordances they provide;
- the *STN ontology*[13] for describing networks of agents and artifacts on the Web;
- the *EVE ontology* [4] for describing agent environments on the Web;
- scenario-specific vocabularies defined for the purposes of our demonstrator.

In our system, we map W3C WoT TDs to descriptions of *artifacts* as defined by the A&A meta-model (see discussion in Section 2.1) – i.e., a Web resource can be both a `td:Thing` and an `eve:Artifact`. This mapping allows autonomous agents to use *Web Things* as they would use any other artifacts programmed with JaCaMo (via the *browser artifacts* presented in the previous section).

Both people and autonomous agents can manipulate the hypermedia environment, for instance by adding devices to the system. The hypermedia environment is hosted on *Yggdrasil*, our prototypical platform for programming hypermedia environments for autonomous agents, and can be distributed across multiple nodes. The version of Yggdrasil used in our demonstrator is on GitHub[14] and provides two core functionalities:

- it serves as a repository for semantic descriptions of hypermedia environments; each Yggdrasil node exposes a REST HTTP API for creating, updating, and deleting RDF representations of *environment*, *workspace*, and *artifact* abstractions (cf. A&A meta-model in Section 2.1);
- it acts as a hub that (partially) implements the W3C WebSub recommendation[15]; agents – or any software clients, such as our *A&A Container* – can use this functionality to observe resources in the environment.

Yggdrasil implements an event-driven non-blocking architecture using *Vert.x*[16], a framework that is both powerful enough to support high-throughput Web servers[17] and lightweight enough to perform well on small devices, such as the Raspberry Pi[18].

3.3 Semantic Hypermedia Search Engine

Autonomous agents in such distributed hypermedia environments need to be able to conduct searches for a broad range of goals and require structured query and result capabilities to achieve their goals. To this end, we developed a search engine for the WoT that autonomous agents can use to perform *approximate search queries* in *(weak) real time*. The search engine consists of two components: (i) our own implementation of a *hypermedia crawler* for dynamic WoT environments, and (ii) a *semantic query engine* based on *Corese* [9], an open-source inference and query engine for Linked Data.[19] Similar to Yggdrasil, our *semantic hypermedia search engine* implements an event-driven non-blocking architecture using *Vert.x*. The hypermedia crawler and the query engine are loosely coupled (and deployed in separate *Vert.x verticles*), which enhances the evolvability of the system. The semantic hypermedia search engine of our demonstrator is on GitHub.[20]

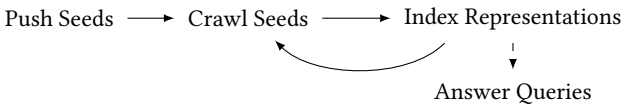

Figure 2: From seeding to query-answering.

Figure 2 depicts an overview of the functioning of our search engine: (i) seed IRIs are pushed to the search engine via crawler artifacts; (ii) the seed IRIs are dereferenced and crawled to discover any available W3C WoT TDs; (iii) the discovered W3C WoT TDs are indexed and (iv) queried using SPARQL. We elaborate on these steps in the following.

3.3.1 Hypermedia crawler. We designed and implemented a crawler that navigates distributed hypermedia environments to discover any resources described with the W3C WoT TD. The seeds for initiating the crawling process can be provided by any entity in the system: humans, autonomous agents, resource directories, etc. We enriched the Yggdrasil platform (see Section 3.2) with the functionality to automatically register seeds with the crawler whenever a component is added to an hypermedia environment it hosts (e.g., the IRI of a newly created *workspace*). This functionality allows the crawler to keep track of the evolution of the distributed hypermedia environment more efficiently as it can rely on Yggdrasil nodes to push notifications whenever parts of the distributed hypermedia environment need to be (re-)crawled.

In addition to seeds, humans and autonomous agents can also configure the crawler with the *link types* to be followed when navigating hypermedia environments (e.g., links among TDs, links

[12] Throughout this paper, we use the prefix bindings in Table 1.
[13] https://w3id.org/stn/
[14] https://github.com/Interactions-HSG/yggdrasil/tree/iot2019/
[15] https://www.w3.org/TR/2018/REC-websub-20180123/
[16] http://www.vertx.io/
[17] According to independent benchmarks for Web frameworks: https://www.techempower.com/benchmarks/, accessed: 08.09.2019.
[18] http://vertx.io/blog/vert-x3-web-easy-as-pi/, accessed: 08.09.2019.
[19] https://project.inria.fr/corese/ and also http://corese.inria.fr/
[20] https://github.com/Interactions-HSG/wot-search/

between workspaces and contained artifacts etc.). This functionality allows the crawler (i) to be customized for hypermedia environments described with various ontologies, and (ii) to navigate large-scale hypermedia environments more efficiently.

The crawler exposes two HTTP endpoints: `/registrations` for pushing seeds for the crawling process (IRIs of artifacts, workspaces, etc.), and `/links` for pushing IRIs denoting link types that should be followed when crawling hypermedia environments. Agents can access these endpoints using the *crawler artifacts* provided by the *A&A Container* (see Section 3.1). After being seeded, the crawler dereferences the registered IRIs to obtain resource representations. From these representations, it extracts links to be followed and continues the crawling recursively in a depth-first manner. We study optimized crawling plans in another work [13]. The crawler stores discovered W3C WoT TDs as RDF data to be indexed and queried.

3.3.2 Semantic query engine. The W3C WoT TDs discovered by our hypermedia crawler are indexed and queried using *Corese*[21] [9]. The query engine exposes a SPARQL endpoint `/search` that autonomous agents can use to search for artifacts needed to achieve their goals. Agents access the SPARQL endpoint using the *finder artifacts* provided by the *A&A Container* (see Section 3.1).

A feature of Corese that is central to our system is the ability to process approximate queries: if there is no exact answer for a query, Corese can approximate the semantics of the query, its structure, or both [8]. To illustrate this feature, let there be an OWL ontology that describes industrial robots in which the `UR5` and `UR10` series of single-armed robots from *Universal Robots* are sibling subclasses of `SingleArmedRobots`, while the `Baxter` series of two-armed robots from *Rethink Robotics* is a subclass of `TwoArmedRobots` (where `SingleArmedRobots` and `TwoArmedRobots` are sibling subclasses of `RobotsWithArms`. If an agent searches for a `UR5` robot and none is available, Corese uses the ontological distance between the classes (as they are defined in the class hierarchy) to approximate a `UR10` robot as being semantically closer to a `UR5` than a `Baxter` robot is. We refer the interested reader to [8] for further details on all the query approximation techniques used by Corese.

Corese provides several other features that could be leveraged for searching the WoT, such as federated queries over heterogenous data sources (see also Section 5).

4 PROTOTYPE DEPLOYMENT

We deployed a demonstrator based on the scenario presented in Section 1. A video of our demonstrator that corresponds with the following description of our deployment is on YouTube,[22] and the source code is on GitHub.[23] In the following, we first present our deployment setup and then discuss the demonstrator scenario.

[21] https://github.com/Wimmics/corese
[22] https://youtu.be/iuTzzMA-7FI
[23] https://github.com/Interactions-HSG/wot-search-manufacturing/

4.1 Deployment Setup

We deployed our system in a laboratory at the University of St. Gallen. We used two devices in our deployment: a *PhantomX AX-12 Reactor Robot Arm*[24] controlled via an HTTP API[25] that can be accessed from the Internet, and a *Philips Hue*[26] light bulb controlled via an HTTP API exposed by a Philips Hue bridge in the local network. We deployed a hypermedia environment distributed across two Yggdrasil nodes running on a MacBook Pro machine in the local network. We deployed the search engine on the same machine together with two *A&A Containers* that host the agents in our demonstrator. Even though in this setup all software components are deployed on the same machine, the components interact with one another via HTTP and thus they could be easily deployed across the Internet.

All HTTP requests used to register resources in the distributed hypermedia environment in our demonstrator are available online as a *Postman* collection.[27]

4.2 Demonstrator Scenario

Each of the two Yggdrasil nodes in our deployment hosts the hypermedia environment of a production site – *Site A* and *Site B*. The Philips Hue light bulb is deployed on Site A and the PhantomX robot is deployed on Site B. Both the light bulb and the robot are modeled as artifacts that agents can observe and use. We deploy two autonomous agents for each production site: on Site A we deploy a *maintenance agent* tasked with monitoring and maintaining industrial robots across all production sites, and on Site B we deploy a *manufacturing agent* tasked with controlling the robot arm during normal operation. Each agent runs in one *A&A Container*.

In the following, we present the evolution of our demonstrator across three phases (cf. demonstrator video):

4.2.1 Phase 1. The maintenance agent on Site A is launched, starts to *observe*[28] the light bulb, and seeds the crawler with the IRI of its local workspace on Site A. The crawler then crawls the workspace by following all `stn:connectedTo`[29] and `eve:contains`[30] links, which lead the crawler to discover a second workspace on Site B that contains the robot artifact.

The maintenance agent on Site A searches for robots to be monitored – across all existing production sites – by querying the search engine for artifacts that are robotic devices. The SPARQL query issued in our demonstrator is shown in Listing 2.

Listing 2: SPARQL query to discover robots.

```
1 PREFIX ex: <http://example.com/> .
2 CONSTRUCT { ?p a ?y . }
3 WHERE {?p a ex:RoboticDevice . }
```

[24] https://www.trossenrobotics.com/p/phantomx-ax-12-reactor-robot-arm.aspx, accessed: 08.09.2019.
[25] https://github.com/Interactions-HSG/leubot
[26] https://developers.meethue.com/develop/get-started-2/, accessed: 08.09.2019.
[27] https://www.getpostman.com/collections/c509a6299e357ff4aff0
[28] The JaCaMo platform allows agents to observe artifacts: changes in an artifact's state, or signals emitted by the artifact are reflected in the observing agent's belief base.
[29] Meaning here: *a workspace is connected to another workspace.*
[30] Meaning here: *an artifact is contained in a workspace.*

The ability of Corese to process approximate queries allows the maintenance agent to query for devices of type `ex:RoboticDevice` and to receive as a result the PhantomX robot on Site B, which is of type `ex:Ax12ReactorArm` – in the vocabulary used for this demonstrator, `ex:Ax12ReactorArm` is a subclass of `ex:RoboticDevice`. Once the robot artifact on Site B is found, the maintenance agent at Site A focuses on it to receive any events it might generate.

The manufacturing agent on Site B is launched at the end of Phase 1 and starts operating the robot.

4.2.2 Phase 2. The robot at Site B malfunctions and issues a maintenance event. This event is pushed from the Yggdrasil node on Site B to the *A&A Container* on Site A, which dispatches the event to the maintenance agent due to the subscription created earlier. Upon receiving the event, the maintenance agent searches for a maintenance supplier that can perform the maintenance task. To this end, it issues the SPARQL query in Listing 3, where `robotURI` is the URI of the robot to be maintained.

Listing 3: SPARQL query for autonomous maintenance.

```
PREFIX ex: <http://example.com/> .
CONSTRUCT { ?p ex:maintains robotURI . }
WHERE {?p ex:maintains robotURI . }
```

Since no maintenance robot is deployed at Site B, the search returns an empty result. The agent then notifies any maintenance engineer that might be available on Site A by switching on the light bulb with a red color (cf. Listing 1). A maintenance engineer travels to Site B, repairs the manufacturing robot, and the robot resumes its tasks.

4.2.3 Phase 3. A maintenance robot is deployed on Site B and registered with the local Yggdrasil node, which automatically updates the workspace of Site B. Yggdrasil pushes the IRI of the newly added robot to the crawler, which discovers the robot.

Similar to Phase 2, the manufacturing robot malfunctions and issues a maintenance event, which is dispatched to the maintenance agent on Site A. Upon receiving the event, the maintenance agent issues again the SPARQL query in Listing 3 and receives as a result the newly installed maintenance robot on Site B. The maintenance agent delegates the task to the maintenance robot and informs manufacturing engineers on Site A by switching on a green light (cf. Listing 1).

5 DISCUSSION AND LIMITATIONS

The deployed demonstrator proves the two key elements of our approach. First, the maintenance agent is able to use *approximate search queries* to find industrial robots in a hypermedia environment distributed across two production sites. In our setup, the Yggdrasil nodes for the two production sites run on the same machine, but they could be easily distributed across the Web – as they are in the Yggdrasil demonstrator presented in [4]. Second, our prototypical search engine has the ability to keep track of the evolution of the hypermedia environments it indexes, which allows the maintenance agent to perform searches in *(weak) real time*: when the maintenance robot is added to the environment, Yggdrasil notifies the search engine, which crawls and indexes the robot. In the future, we intend to implement a similar mechanism for tracking components that are removed from the environment. We say searches are performed in *weak real time* because the notification-based mechanism used in our prototype would be insufficient for keeping track of large-scale, rapidly evolving environments. In the future, complementary mechanisms could be added to improve real-time search. For instance, predictive crawling (e.g., see [12]) could be used to crawl and index fast-changing areas in the environment more frequently, or to determine which parts of an environment should be prioritized during crawling.

Our search engine crawls WoT environments to discover and index W3C WoT TDs. In most cases, the TDs would describe devices, but they could also describe resource directories, such as the *Thing Directory*.[31] Our current implementation would treat a discovered Thing Directory as any regular Thing in the environment – and thus leaves it to agents to use the Thing Directory if they are able to do so. In the future, we intend to extend our search engine with the ability to automatically query SPARQL endpoints discovered in the environment at run time. Corese already supports federated SPARQL queries, and we study the automatic discovery and querying of SPARQL endpoints in another work [22]. Our current implementation also does not check the correctness of discovered TDs beyond RDF syntax – for instance, to check if a given TD is usable and corresponds to the Thing being described, or if a described device is operational. We leave it as future work to investigate such mechanisms.

As a direction for future research, we intend to investigate the ranking of resources based on agents' goals and current context. For instance, if an agent has the goal to increase the light level in a room, it could achieve this goal either using light bulbs or window blinds, but the relevance of these resources is contextual: opening the window blinds during night-time would have little impact on the room's light level. Going further, in our current approach agents rely on libraries of plans programmed by developers in order to "bridge" their goals to relevant resources. This knowledge currently has to be hard-coded into the agents (cf. Listing 1), it can be obtained at run time from other agents (if available), or could potentially be inferred at the expense of added complexity (e.g., via automated planning). Providing agents with a context-aware search engine that can process goal-oriented queries (rather than resource-oriented queries) would further enhance the agents' flexibility in achieving their goals.

6 CONCLUSIONS

We hypothesize that similar to how hypermedia search enhances people's ability to achieve their everyday goals through the Web (for online shopping, travel planning, etc.), *semantic hypermedia search* can enhance the autonomous behavior of software agents in WoT environments. To this end, we designed and implemented a prototypical search engine that allows autonomous agents to use *approximate search queries* for finding relevant resources in their WoT environment in *(weak) real time*. We demonstrate these features in a maintenance scenario for a prototypical agent-based manufacturing system deployed in one of our laboratories at the University of St. Gallen. Our demonstrator shows that – through

[31] https://github.com/thingweb/thingweb-directory/, accessed: 08.09.2019.

these features – the search facility enhances the agents' flexibility and agility in achieving their goals.

Our prototypical search engine is crawling and indexing only W3C WoT TDs, but the same search facility could serve a broader range of purposes. For instance, autonomous agents could use the search engine to discover how to interact with one another based on declarative specifications of agent interaction protocols (e.g., in a formal language such as BSPL [29]) or of multi-agent organizations (e.g., in a formal language such as MOISE OML [14]). Such resources could be designed into the hypermedia environment to further enhance autonomous behavior in open and long-lived WoT systems.

REFERENCES

[1] Maribel Acosta, Olaf Hartig, and Juan Sequeda. 2018. *Federated RDF Query Processing*. Springer International Publishing, Cham, 1–8. https://doi.org/10.1007/978-3-319-63962-8_228-1

[2] Olivier Boissier, Rafael H. Bordini, Jomi F. Hübner, Alessandro Ricci, and Andrea Santi. 2013. Multi-agent oriented programming with JaCaMo. *Science of Computer Programming* 78, 6 (2013), 747 – 761. https://doi.org/10.1016/j.scico.2011.10.004

[3] Rafael H Bordini, Jomi Fred Hübner, and Michael Wooldridge. 2007. *Programming multi-agent systems in AgentSpeak using Jason*. Vol. 8. John Wiley & Sons.

[4] Andrei Ciortea, Olivier Boissier, and Alessandro Ricci. 2018. Engineering World-Wide Multi-Agent Systems with Hypermedia. In *Proceedings of the 6th International Workshop on Engineering Multi-Agent Systems*. http://emas2018.dibris.unige.it/images/papers/EMAS18-17.pdf

[5] Andrei Ciortea, Simon Mayer, Fabien Gandon, Olivier Boissier, Alessandro Ricci, and Antoine Zimmermann. 2019. A Decade in Hindsight: The Missing Bridge Between Multi-Agent Systems and the World Wide Web. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2019, Montreal, Canada, May 13-17, 2019*. International Foundation for Autonomous Agents and Multiagent Systems.

[6] Andrei Ciortea, Simon Mayer, and Florian Michahelles. 2018. Repurposing Manufacturing Lines on the Fly with Multi-agent Systems for the Web of Things. *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)* (2018), 813–822. https://www.alexandria.unisg.ch/255802/

[7] Andrei Ciortea, Antoine Zimmermann, Olivier Boissier, and Adina M. Florea. 2015. Towards a Social and Ubiquitous Web: A Model for Socio-Technical Networks. In *2015 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, Vol. 1. 461–468. https://doi.org/10.1109/WI-IAT.2015.205

[8] Oivier Corby, Rose Dieng-Kuntz, Fabien Gandon, and Catherine Faron-Zucker. 2006. Searching the semantic Web: approximate query processing based on ontologies. *IEEE Intelligent Systems* 21, 1 (Jan 2006), 20–27. https://doi.org/10.1109/MIS.2006.16

[9] Olivier Corby, Alban Gaignard, Catherine Faron-Zucker, and Johan Montagnat. 2012. KGRAM Versatile Inference and Query Engine for the Web of Linked Data. In *IEEE/WIC/ACM International Conference on Web Intelligence*. Macao, China, 1–8. https://hal.archives-ouvertes.fr/hal-00746772

[10] Michael P Georgeff and Amy L Lansky. 1987. Reactive reasoning and planning.. In *AAAI*, Vol. 87. 677–682.

[11] Dominique Guinard, Vlad Trifa, Thomas Pham, and Olivier Liechti. 2009. Towards physical mashups in the web of things. In *Networked Sensing Systems (INSS), 2009 Sixth International Conference on*. IEEE, 1–4.

[12] Shuguang Han, Bernhard Brodowsky, Przemek Gajda, Sergey Novikov, Mike Bendersky, Marc Najork, Robin Dua, and Alexandrin Popescul. 2019. Predictive Crawling for Commercial Web Content. In *Proceedings of the 2019 World Wide Web Conference*. 627–637.

[13] Hai Huang and Fabien Gandon. 2019. Learning URI Selection Criteria to Improve the Crawling of Linked Open Data. In *ESWC2019 - The 16th Extended Semantic Web Conference*. Portoroz, Slovenia. https://hal.inria.fr/hal-02073854

[14] Jomi F. Hübner, Jaime S. Sichman, and Olivier Boissier. 2007. Developing Organised Multiagent Systems Using the MOISE+ Model: Programming Issues at the System and Agent Levels. *Int. J. Agent-Oriented Softw. Eng.* 1, 3/4 (Dec. 2007), 370–395. https://doi.org/10.1504/IJAOSE.2007.016266

[15] Nicholas R. Jennings and Michael Wooldridge. 1998. *Applications of Intelligent Agents*. Springer Berlin Heidelberg, Berlin, Heidelberg, 3–28. https://doi.org/10.1007/978-3-662-03678-5_1

[16] Sebastian Käbisch and Takuki Kamiya. 2018. *Web of Things (WoT) Thing Description*. W3C Working Draft. W3C. https://www.w3.org/TR/2018/WD-wot-thing-description-20181021/.

[17] Andreas Kamilaris, Semih Yumusak, and Muhammad I. Ali. 2016. WOTS2E: A search engine for a Semantic Web of Things. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*. 436–441. https://doi.org/10.1109/WF-IoT.2016.7845448

[18] Aman Kansal, Suman Nath, Jie Liu, and Feng Zhao. 2007. SenseWeb: An Infrastructure for Shared Sensing. *IEEE MultiMedia* 14, 4 (Oct 2007), 8–13. https://doi.org/10.1109/MMUL.2007.82

[19] Simon Mayer and Dominique Guinard. 2011. An Extensible Discovery Service for Smart Things. In *Proceedings of the Second International Workshop on Web of Things (WoT '11)*. ACM, New York, NY, USA, Article 7, 6 pages. https://doi.org/10.1145/1993966.1993976

[20] Simon Mayer, Dominique Guinard, and Vlad Trifa. 2012. Searching in a Web-based Infrastructure for Smart Things. In *2012 3rd IEEE International Conference on the Internet of Things*. 119–126. https://doi.org/10.1109/IOT.2012.6402313

[21] Simon Mayer, Ruben Verborgh, Matthias Kovatsch, and Friedemann Mattern. 2016. Smart Configuration of Smart Environments. *IEEE Transactions on Automation Science and Engineering* 13, 3 (July 2016), 1247–1255. https://www.alexandria.unisg.ch/255762/

[22] Franck Michel, Catherine Faron-Zucker, Olivier Corby, and Fabien Gandon. 2019. Enabling Automatic Discovery and Querying of Web APIs at Web Scale using Linked Data Standards. In *WWW 2019 - LDOW/LDDL Workshop of the World Wide Web Conference*. San Francisco, United States. https://doi.org/10.1145/3308560.3317073

[23] Benedikt Ostermaier, Kay Römer, Friedemann Mattern, Michael Fahrmair, and Wolfgang Kellerer. 2010. A Real-Time Search Engine for the Web of Things. In *Proceedings of Internet of Things 2010 International Conference (IoT 2010)*. Tokyo, Japan.

[24] Dennis Pfisterer, Kay Romer, Daniel Bimschas, Oliver Kleine, Richard Mietz, Cuong Truong, Henning Hasemann, Alexander Kröller, Max Pagel, Manfred Hauswirth, Marcel Karnstedt, Myriam Leggieri, Alexandre Passant, and Ray Richardson. 2011. SPITFIRE: toward a semantic web of things. *IEEE Communications Magazine* 49, 11 (November 2011), 40–48. https://doi.org/10.1109/MCOM.2011.6069708

[25] Anand S Rao. 1996. AgentSpeak (L): BDI agents speak out in a logical computable language. In *European Workshop on Modelling Autonomous Agents in a Multi-Agent World*. Springer, 42–55.

[26] Alessandro Ricci, Michele Piunti, and Mirko Viroli. 2011. Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems* 23, 2 (2011), 158–192.

[27] K. Römer, B. Ostermaier, F. Mattern, M. Fahrmair, and W. Kellerer. 2010. Real-Time Search for Real-World Entities: A Survey. *Proc. IEEE* 98, 11 (Nov 2010), 1887–1902.

[28] Yoav Shoham. 1993. Agent-oriented programming. *Artificial intelligence* 60, 1 (1993), 51–92.

[29] Munindar P. Singh. 2011. Information-driven Interaction-oriented Programming: BSPL, the Blindingly Simple Protocol Language. In *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2 (AAMAS '11)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 491–498. http://dl.acm.org/citation.cfm?id=2031678.2031687

[30] T Smith and J Guild. 1931. The C.I.E. colorimetric standards and their use. *Transactions of the Optical Society* 33, 3 (jan 1931), 73–134. https://doi.org/10.1088/1475-4878/33/3/301

[31] Chiu C. Tan, Bo Sheng, Haodong Wang, and Qun Li. 2010. Microsearch: A Search Engine for Embedded Devices Used in Pervasive Computing. *ACM Trans. Embed. Comput. Syst.* 9, 4, Article 43 (April 2010), 29 pages. https://doi.org/10.1145/1721695.1721709

[32] Haodong Wang, Chiu C. Tan, and Qun Li. 2010. Snoogle: A Search Engine for Pervasive Environments. *IEEE Transactions on Parallel and Distributed Systems* 21, 8 (Aug 2010), 1188–1202. https://doi.org/10.1109/TPDS.2009.145

[33] Gerhard Weiss. 2000. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press.

[34] Kok-Kiong Yap, Vikram Srinivasan, and Mehul Motani. 2005. MAX: Human-centric Search of the Physical World. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys '05)*. ACM, New York, NY, USA, 166–179. https://doi.org/10.1145/1098918.1098937

[35] Yuchao Zhou, Suparna De, Wei Wang, and Klaus Moessner. 2016. Search techniques for the web of things: A taxonomy and survey. *Sensors* 16, 5 (2016), 600.