ICT-56-2020 "Next Generation Internet of Things"
Grant Agreement number: 957218

IntellioT

Deliverable D3.1

Interoperable, Self-aware & Semi-autonomous Multi-agent Systems

Deliverable release date	03/12/2021
Authors	1. Simon Mayer (HSG) 2. Samuele Burattini (HSG) 3. Danai Vachtsevanou (HSG) 4. Jérémy Lemée (HSG) 5. Andrei Ciortea (HSG) 6. Babis Savvakos (TSI) 7. Konstantinos Fysarakis (SANL)
Editor	Simon Mayer (HSG)
Reviewer	Andreas Zirkler (Siemens), Andreas Ziller (Siemens)
Approved by	PTC Members: (Vivek Kulkarni, Konstantinos Fysarakis, Sumudu Samarakoon, Beatriz Soret, Arne Bröring, Maren Lesche) PCC Members: (Vivek Kulkarni, Jérôme Härri, Beatriz Soret, Mehdi Bennis, Martijn Rooker, Sotiris Ioannidis, Anca Bucur, Georgios Spanoudakis, Simon Mayer, Filippo Leddi, Harshitha Chandregowda, Maren Lesche, Georgios Kochiadakis)
Status of the Document	Final
Version	1.0
Dissemination level	Public

Table of Contents

1 EXECUTIVE SUMMARY

This document describes the design and implementation of open-source software components for Hypermedia Multi-Agent Systems (MAS) within IntellIoT and our first steps towards the integration of low-power devices in this context. This forms the basis on which IoT applications can be created by users within the IntellIoT project – consequently, the architecture and systems discussed in this document are present in all three use cases of IntellIoT, albeit at varying integration levels depending on the requirements of the respective use case. In the project context, these are the first results of our work on Task 3.1 within WP3. We discuss how we propose to enable flexibility, openness, and interoperability within IntellIoT's use cases, and how we propose to create systems that can be (re)configured by Domain Experts[1] rather than requiring programmers for system integration, thereby lowering the integration barrier, and opening up IntellIoT to use cases beyond the considered focuses on Agriculture (UC1), Healthcare (UC2), and Manufacturing (UC3). Towards the Technology Deliverables of the IntellIoT Project, this document reports on the *End-user Programming Approach for Hypermedia Multi-Agent Systems* and the *Open-source Software Components for Hypermedia Multi-Agent Systems*. Regarding the former, we report on our first concept and prototype of a system that permits Domain Experts to configure systems of Agents in the IntellIoT use cases and beyond; the latter reports on the first concepts and prototypes of several infrastructure components (i.e., repositories, discovery components) that support the configuration and running of Hypermedia MAS in the IntellIoT use cases and beyond. Our core contributions as part of T3.1 are several systems that permit:

1. The publishing and discovery of W3C WoT Thing Descriptions (W3C WoT TD)[2]
2. The monitoring of workspaces that contain W3C WoT TD-described devices and services
3. The configuration of Autonomous Agents (in the following referred to as *Agents*) by Domain Experts
4. The deployment of the configured agents and their interaction with devices and services
5. The formulation of goals that deployed systems of agents work towards by End Users[3]
6. The integration of constrained IoT devices

At the time of submission, functional prototypes for each of these five systems are available. In addition, components 1-4 are integrated and together allow the configuration of Agents based on W3C WoT TDs as well as their deployment, in HSG laboratory facilities and with HSG machines, which covers several aspects of the demonstrators of UC1 (agent-operated mobile robots) and UC3 (agent-operated pick-and-place robots). Furthermore, our components are integrated with mocked machines (i.e., emulated laser cutter and milling machine) that are provided by UC3 partners.

[1] In the context of this document, we use the term "Domain Expert" to refer to the individual who configures an IntellIoT application. This person is assumed to have full knowledge of the functionalities that are provided by different devices and services, but is not knowledgeable about how these functionalities, or the device and service APIs that provide them, can be integrated technically. We assume that this person does not have a programming background. In the different use cases, we use the terms "Farming Engineer", "Health System Engineer", and "Manufacturing Engineer", respectively. The definition of a Domain Expert however spans beyond these use cases, e.g. to smart-home enthusiasts, where different scenarios exhibit large differences in the complexity of the available devices and services as well as the desired behavior of the system.

[2] This refers to the World Wide Web Consortium's (W3C) Web of Things (WoT) standardization group that has recently standardized Thing Descriptions (TDs) that describe the metadata and interfaces of Things.

[3] In the context of this document, we use the term "End User" to refer to individuals who interact with deployed IntellIoT applications. In the different use cases, these are a "Farmer", "Doctor", and "Customer", respectively.

2 CREATING INTELLIOT APPLICATIONS

T3.1 focuses on enabling Domain Experts to configure and run IntellIoT applications that integrate the functionality of devices and services across the IntellIoT use cases. In this context, the central decision in the IntellIoT project was not to strive for full automation of this integration, as previous research by us and others has shown that, while possible in sufficiently constrained environments [1], automatic service integration quickly hits scalability boundaries [2]. We rather argue that creativity should be provided by the entity that can most effectively provide it in collaborating groups of humans, automated planners, and machine learning. This is ideally demonstrated in the contributions of WP3 to UC3 of IntellIoT (see Section 4.3 for further details): A Domain Expert configures the overall behavior of the system given available functionalities (T3.1), including the consulting of machine learning systems that inform individual steps of the behavior (see T3.2) and including escalation steps that should be taken to involve further humans as remote experts (see T3.3). Parts of such a configuration may be accomplished by automated planners that the Domain Expert delegates sub-tasks to[4]; the execution of the configured application then follows the specified overall behavior, where the specific, concrete, behavior depends on which devices provide the required functionalities at run time and on the outputs of the entities that are delegated sub-tasks (i.e., the machine learning system and the remote experts, in this case).

Within T3.1 of IntellIoT, we explore this space in-*between* fully automatic service composition and manual service composition. To permit exploring this spectrum, we argue that an abstraction is needed that permits Domain Experts to effectively configure systems of autonomous software programs, that is, of Autonomous Agents (in the following referred to as *Agents*). If designed in a sufficiently open and interoperable way, such an abstraction would then permit us to introduce a separation of concerns between tasks that are efficiently accomplished by Domain Experts and tasks that should be left to the system. The former are typically *top-down creative integration tasks* that require a high-level overview of a system's components and how they should work together – tasks that humans are trained to accomplish when interacting with other (human) agents in their everyday lives, from the planning of a conference and allocation of individual responsibilities to the involved agents, to the planning of a family holiday, and the allocation of tasks to different workers in a manufacturing workshop, where in all these scenarios *balancing* decisions need to be taken (e.g., given individual preferences or idiosyncrasies of the involved agents) that cannot be easily automated. The latter are typically *bottom-up repetitive integration tasks* that require a great deal of precision but are decoupled, that is, they have fewer or no systemic interdependencies – tasks that are easily automatable given access to the right information, from the changing of IP addresses to cope with an upgraded network topology to the formulation of low-level API requests to achieve a provided functionality.

To enable this separation of concerns in complex, large-scale, and open smart environments, we draw upon past research in the Web Architecture, Web of Things, and MAS fields, and have recently proposed Hypermedia Multi-Agent Systems as a "new generation of autonomous systems on the Web" [3,4]. From this perspective, with the IntellIoT project, we strive to alleviate both issues that we highlighted in that article regarding the lack of wide-spread adoption of earlier Web-based MAS: the lack of practical use cases as well as the premature alignment of Agent technologies and the World Wide Web. Observing that Web researchers (including some of the authors) have, over the past decade, built autonomous systems on their own while neglecting relevant research in the MAS domain and that, at the same time, Web-based MAS have been created that are in juxtaposition to the modern Web architecture, we argue that recent developments and initiatives, including the W3C WoT, both motivate and necessitate large-scale and practice-oriented projects that promote that development of dynamic, open, and long-lived systems that are integrated with the modern Web architecture; this is the goal of T3.1 within IntellIoT, where we further argue that our proposed Hypermedia MAS are especially well-suited for systems that exhibit high dynamics with respect to the available

[4] This aspect is not in scope of the IntellIoT project but follows naturally from the architecture that is provided by the project in combination with previous research on automated planning in manufacturing systems.

devices and services (cf. WP4 / T4.1) as well as on the level of their network interfaces (cf. WP4 / T4.2-T4.3) and also apply to constrained IoT devices, if appropriate bridging is available. However, the proposed raising of the abstraction level implies that it is required to provide Domain Experts with an effective way to configure systems of Agents, i.e., no- or low-code approaches to agent-oriented programming.

This deliverable proposes the infrastructural and end-user-facing components that are required to achieve this raising of the abstraction level — that is, to manage, configure, and deploy these Agents, and to enable them to interact with each other and with devices and services that they encounter within individual use cases. Importantly, the infrastructure that we propose is not specific to any of IntellIoT's use cases — on the contrary, our claim is that the features of this higher abstraction level become most relevant when considering modifications to the target scenarios, for instance as part of IntellIoT's Open Calls, and even more so when targeting a future scenario – "Use Case 4".

This is enabled by the decoupling of device and service interfaces and of their provided functionality through W3C Web of Things Thing Descriptions (W3C WoT TD), which hides the technicalities of interacting with the described functionality behind W3C WoT TD-based Interaction Affordances (IAs) and their respective interface bindings. On this basis, our core contribution is a system that permits the publishing and discovery of these IAs (see Section 3.1), their transparent mapping to individual functional blocks that can be composed to IoT applications by Domain Experts (see Section 3.2), and the integration of devices that themselves are unable to expose W3C WoT TD-compatible interfaces (see Section 3.3). These features enable our system to:

- **Replace** functionally equivalent devices at run time without impacting the system, even if the exchange introduces breaking changes to the APIs of the devices or introduces highly constrained devices.
- Deploy configured IntellIoT applications in **new and unknown environments** given that the required functionalities are available, that an entry point to discover their functional descriptions is available, and that the functional descriptions map semantically to the IntellIoT application needs.
- Enable **Domain Experts to extend the functionality** of the system in a given use case, if devices with desired functional profiles become available, by reconfiguring the composition of IAs and redeploying the synthesized agents.
- Enable **Domain Experts** to **configure entirely new IntellIoT applications**, if the required functionality to accomplish the application goal is available.

3 CONFIGURING AND RUNNING HYPERMEDIA MULTI-AGENT SYSTEMS

In this section, we report on the three main components that are developed within the IntellIoT project as part of WP3 / Task 3.1 and that enable the behavior described above. An overview of these components and several other components of IntellIoT is shown in Fig. 1. That figure intentionally simplifies several interactions (e.g., with the Edge Orchestrator) as these are not in focus of this document.

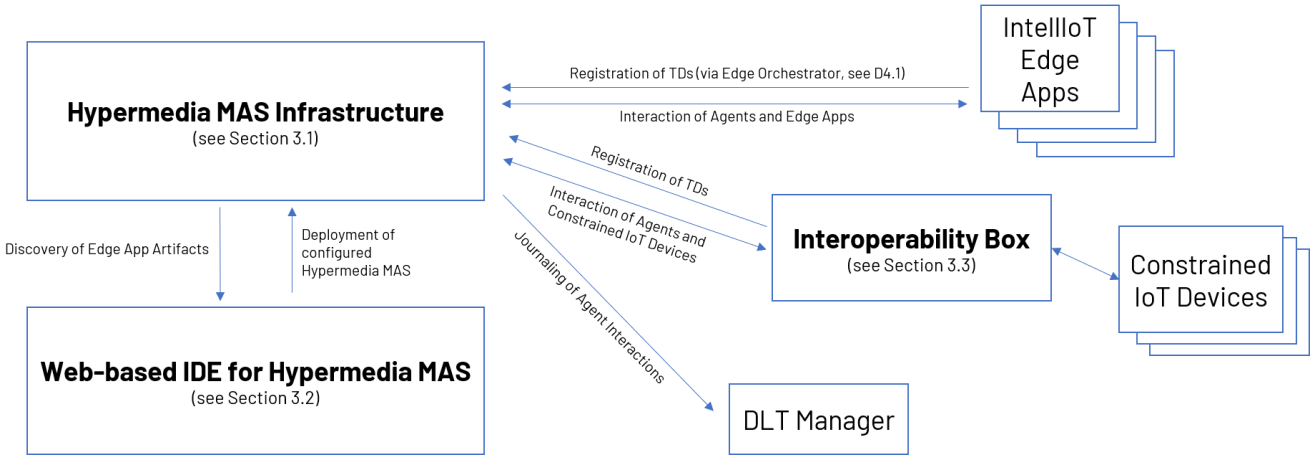

Figure 1: Components of Task 3.1. The Agent Runtime Environment is in the current version still integrated with the Web-based IDE for Hypermedia MAS.

3.1 Hypermedia MAS Infrastructure

The Hypermedia MAS Infrastructure[5] is a platform that is used to create and deploy hypermedia environments based on the Agents & Artifacts meta-model [5]. It also implements a W3C WebSub Hub[6] to permit agents to observe resources in the environment. The main abstractions that are employed by the infrastructure are Environments, Workspaces, and (Hypermedia) Artifacts [5]. All three of these entities are modelled as W3C WoT Things, that is, each is an "abstraction of a physical or a virtual entity whose metadata and interfaces are described by a WoT Thing Description, whereas a virtual entity is the composition of one or more Things."[7]. Thus, the Hypermedia MAS Infrastructure generates and exposes machine-understandable representations for all these entities that are based on the W3C WoT TD Ontology. Thereby, to (machine) clients, the Hypermedia MAS Infrastructure provides the functionality to create, populate, and update environments, workspaces, and artifacts on the platform:

- For (*hypermedia*) *artifacts* (e.g., the IntellIoT UC3 Mock Engraver[8]), the infrastructure exposes the W3C WoT TD-described hypermedia controls to interact with this artifact. In addition, artifact TDs can be updated and clients can subscribe to receive notifications through the infrastructure's W3C WebSub Hub.

[5] https://github.com/Interactions-HSG/yggdrasil
[6] https://www.w3.org/TR/websub/
[7] https://www.w3.org/TR/wot-architecture/\#dfn-thing
[8] https://yggdrasil.interactions.ics.unisg.ch/environments/intelliot/workspaces/uc-industry/artifacts/engraver

- For *workspaces* (e.g., the IntellIoT UC3 workspace[9]), the infrastructure exposes semantic containment relationships [6] for any artifact that is registered to the workspace along with hypermedia controls to create additional artifacts in that workspace, again described in a machine-readable way using W3C WoT TD.
- For *environments* (e.g., the IntellIoT environment[10]), the infrastructure exposes information about workspaces that are contained in the environment in machine-readable form using the EVE vocabulary (cf. [7]) and exposes W3C WoT TDs that describe the functionality of the environment, that is, that workspaces can be created within the environment, along with the relevant hypermedia controls to enable machines to perform this action.

For interacting with these three types of resources, the Hypermedia MAS Infrastructure provides an HTTP API that exposes Create/Read/Update/Delete (CRUD) operations on the following endpoints:

- Environments:

```
/environments/{environment_id}
```

- Workspaces that are contained in an environment:

```
/environments/{environment_id}/workspaces/{workspace_id}
```

- Artifacts that are contained in a workspace that is in turn contained in an environment:

```
/environments/{environment_id}/workspaces/{workspace_id}/artifacts/{artifact_id}
```

HTTP POST and HTTP PUT requests to these endpoints use Turtle[11] payloads, where our current implementation merely validates the payload syntactically. HTTP POST requests may furthermore use Slug header[12] to hint at a preferred Internationalized Resource Identifier (IRI) for a resource to be created. If the IRI is not already in use, it will be minted to the created resource.

Internally, the Hypermedia MAS Infrastructure implements an event-based, non-blocking architecture based on Eclipse Vert.x[13] and constituted of verticles (i.e. the Vert.x functional units that process events and respond to requests) that take care of the following processes:

- Starting the application and initializing other verticles (Main Verticle).
- Handling of HTTP requests (HTTP Server Verticle).
- Handling of W3C WebSub notifications to subscribers (HTTP Notification Verticle).
- Storing W3C Resource Description Framework (RDF)[14] representations of all entities known to the infrastructure and sending of notifications to subscribed observers when one of these representations is updated (RDF Store Verticle).
- Triggering of operations on artifacts as well as sending of notifications to subscribed observers when an (observable) property of an artifact changes (Cartago Verticle).

[9] https://yggdrasil.interactions.ics.unisg.ch/environments/intelliot/workspaces/uc-industry
[10] https://yggdrasil.interactions.ics.unisg.ch/environments/intelliot
[11] https://www.w3.org/TR/turtle/
[12] https://datatracker.ietf.org/doc/html/rfc5023
[13] https://vertx.io/
[14] https://www.w3.org/TR/rdf11-concepts/

In the Hypermedia MAS Infrastructure, the generation and manipulation of W3C WoT TDs is handled using the WoT-TD-Java Library[15] which provides the functionality to deserialize and serialize W3C WoT TDs in Turtle as well as in JSON-LD 1.1[16] based on the W3C TD Information Model[17]. In this context, the library covers those parts of the W3C TD IM that are relevant in the context of IntelliIoT and its use cases: Thing, Interaction Affordance, Property Affordance, Action Affordance, Data Schema (including Array, Boolean, Number, Integer, Object, String, and Null Schemas), Security Schema (including No-Security, Basic, Digest, and API Key Security Schemas), and Form. WoT-TD-Java furthermore follows the W3C WoT TD Default Value Definitions[18], permits the integration of context knowledge from additional namespaces according to the W3C WoT TD Context Extensions[19], and handles the creation of HTTP and Constrained Application Protocol (CoAP)[20] requests as well as the parsing of responses based on a given TD.

The Hypermedia MAS Infrastructure has been deployed on the premises of HSG and has been providing the described functionality (with incremental extensions of its functional interface) to clients since M6 of IntelliIoT. It thereby provides a platform for (machine) clients to discover artifacts that are registered to workspaces within environments, by following the appropriate links in the EVE vocabulary – this is what our end-user programming approach for Hypermedia MAS relies on, see Section 2.2. The infrastructure is furthermore compatible with our past research on a hypermedia search engine for the Web of Things [8]. That search engine was developed to permit Agents to discover resources in dynamic hypermedia environments at planetary scale, by decoupling the resource discovery concern (through crawling along configurable hypermedia paths) from search requests by Agents while permitting approximate search queries.

3.2 Web-based IDE for Hypermedia MAS

Building on top of the Hypermedia MAS Infrastructure, the second aspect of D3.2 concerns the enabling of Domain Experts to configure Hypermedia MAS. As introduced in Section 2, we argue that a beneficial allocation of integration concerns among humans and systems can be achieved by introducing Agents, and, thus, Agent-oriented Programming, with no- or low-code environments to configure Agents' procedural knowledge as well as their organizational setup. In this document, we report on our first prototype that enables the no-code programming of individual Agents; towards achieving the objectives of IntelliIoT's T3.1, we are currently preparing a user study that should solidify our claims towards the enabling of Domain Experts to program Hypermedia MAS, and will then integrate this system with our past research on the no-code definition of Agent organizations [1] towards enabling the no-code specification of complete (multi-agent) Hypermedia MAS.

In IntelliIoT and based on our past experience with the use of no-code languages to configure Web of Things mashups and specify goals for automated planners in Web of Things contexts (in [9]), we have decided to explore the usage of a blocks-based abstraction to enable Domain Experts to program Hypermedia MAS. Blocks-based programming is today a widely accepted entry point to introductory coding (especially for small children) [10] and is also widely used in other no- or low-code contexts with similar requirements as are present in IntelliIoT, such as the teaching of movement behaviors to collaborative robots – both in a research context [11] as well as in commercial products such as the xArm Studio[21] software by UFACTORY. Blocks-based languages are visual programming languages that permit users to create programs primarily through the manipulation of blocks, where each type of block corresponds to a concept in

[15] https://github.com/Interactions-HSG/wot-td-java
[16] https://www.w3.org/TR/json-ld11/
[17] https://www.w3.org/TR/wot-thing-description/\#dfn-inf-model
[18] https://www.w3.org/TR/wot-thing-description/\#sec-default-values
[19] https://www.w3.org/TR/wot-thing-description/\#sec-context-extensions
[20] https://datatracker.ietf.org/doc/html/rfc7252
[21] https://www.ufactory.cc/pages/xarm

an underlying general-purpose programming language or a domain-specific language. Here, it is common to design blocks in ways so that their (graphically represented) interfaces provide hints to users about which blocks fit together. By assembling them, these blocks allow users to express their ideas in a way that can be parsed to executable code in the underlying language, and that can thus be executed on a compatible target system.

In the context of the IntellIoT project, we thus aspire to (a) provide a mapping of an Agent programming language to a blocks-based no-code language so that the relevant concepts of the Agent programming language can be expressed by Domain Experts through blocks and to (b) demonstrate that Domain Experts are actually able to program MAS in this way more efficiently and more intuitively than they can program functionally similar systems through other abstractions, such as flow-based programming (e.g., in Node-RED), or object-oriented or procedural programming. Here, we could ideally show that it is the *raising of the abstraction level* to Agent-oriented Programming rather than language-specific constructs that lead to the hypothesized increase in intuitiveness and efficiency. This reflects our discussion from Section 2 about the everyday experiences of people "programming" human agents in their environment together with arguments from the scientific literature that posit that the (mentalistic) abstractions in Agent-oriented Programming simplifies the design and programming of artificial agents: Agent-oriented Programming focuses on a societal view of computation [12]. We argue that the mentalistic view of Agents that is adopted in Agent-oriented Programming should enable Domain Experts to more intuitively program agents to deliberate about their mental states (e.g., their beliefs, desires, and intentions), to modify their mental states as needed, and to interact with artifacts in their environment – as this closely matches the way humans reason about other (human) agents and thereby merely generalizes human-oriented abstraction to include artificial Agents, and systems of artificial Agents.

Against this background, this aspect of T3.1 focuses on the design and development of a platform that Domain Experts could use to configure IntellIoT applications, that is, an Integrated Development Environment (IDE) with appropriate blocks-based abstractions for Domain Experts that are thereby enabled to leverage the Agent paradigm for programming Hypermedia MAS on top of W3C WoT TD-described devices and services. By hiding common obstacles encountered by unexperienced developers (e.g., learning the syntax of a language) through a blocks-based language, we hope that users can focus purely on the definition of the system behavior and the procedural knowledge and organization of Agents. Finally, in the current version of this IDE, configured MAS are executed in a separate runtime environment that is attached to the IDE; however, this component will be decoupled from the IDE and integrated with the Hypermedia MAS Infrastructure in the coming months.

In the following, we describe the current versions of both, the *Web IDE* that is used by Domain Experts to develop MAS and the *Runtime Environment* that runs configured Agents. In the design of the whole platform, we make use of Web technologies to ensure that our software can run on heterogeneous systems, to relieve users from the requirement to install specific software packages, and to enable the platform to be accessible from across different end-user devices. The overall architecture of the current system is depicted below:

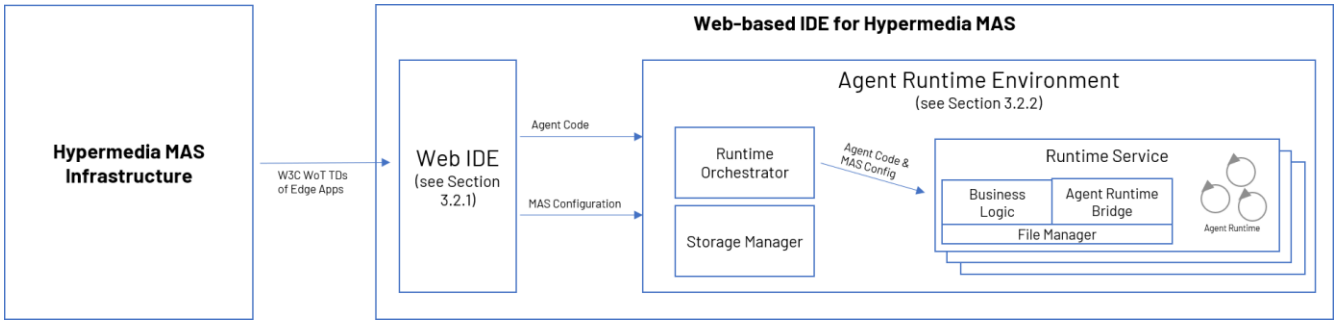

Figure 2: Overview of the components of the Web-based IDE for Hypermedia MAS.

In our platform, the Web IDE takes as input W3C WoT TDs of the devices and services available in an environment that are obtained through the Hypermedia MAS Infrastructure (see Section 3.1). As the Domain Expert interacts with the Web IDE to configure Agent behaviors, synthesized Agent code is persisted in a back end of the platform to allow users

to modify the code they created whenever they need to. After configuring one or several Agents, the Domain Expert is required to define a MAS configuration that determines which of the Agents should run; together with the procedural knowledge (i.e., "Agent Programs") of the individual agents, this is submitted to the Runtime Orchestrator for execution. The Runtime Orchestrator, in turn, creates a new instance of a Runtime Service and starts the execution of the MAS according to the Agent programs. Runtime Services themselves are Web services that provide HTTP APIs to accept the Agent programs and the runtime configuration.

3.2.1 WEB IDE

The Web IDE is at the core of our system. It provides Domain Experts with an interface to design Agents and MAS configurations and submits these to the Runtime Orchestrator. Agents are designed using a blocks-based language that is based on Blockly[22] and maps the most relevant concepts of the Agent programming language Jason (an extension of AgentSpeak) [13] – to our knowledge, this is the first no-code Agent programming language. Our blocks-based language was designed following the principle of modularity: Instead of creating many custom blocks that could generate different statements, the user is given a range of basic conceptual blocks that can then be composed together to write programs as expressive as regular code but simplifying both the syntax and the overall program structure. When creating a new Agent using our system, two initial blocks are created in the Web IDE that encapsulate the initial definition of the Agent's belief state and provide an empty shell to define the Agent plan, respectively. The Domain Expert then programs the Agent plan by adding additional blocks to the system, where we use Blockly's features to constrain which blocks and block types can be assembled (we also use color-coding to suggest fitting blocks). While our current mapping of Jason to this blocks-based language supports all concepts that we believe to be relevant for achieving the project goals of IntellIoT, the language currently does not support Jason Annotations (which are an important feature in the original language). It also does not currently support control statements (conditional clauses, loops etc.): these are extensions that were made by Jason to (pure) AgentSpeak and, while such extensions support the adoption of the language by developers familiar with procedural programming, they are not required — they can instead be substituted by other features of the language that we argue are more intuitive to Domain Experts without a programming background.

Using the Web IDE, Domain Experts are able to select a workspace in which their agents will operate. This workspace is selected from the available workspaces at the Hypermedia MAS Infrastructure and links to the W3C WoT TDs of its contained artifacts. Based on this selection, the programming of an IntellIoT application is then driven by the aforementioned abstractions of the Jason language together with generated blocks that are synthesized from the W3C WoT TDs of Things in the selected workspace – these blocks enable the Agents to make use of interaction affordances that are provided by the devices and services and can be combined with other synthesized blocks and Jason constructs. With these abstractions, Domain Experts are currently able to use our Web IDE to:

- Program one or several agents simultaneously through the blocks-based language and assign names to these Agents to identify classes of Agents. Agents are limited to "living" in one workspace at a time, that is, Agents need to be co-located with the Things they are using. While it is possible for Domain Experts to break through this requirement easily, we decided to impose this constraint to force a conscious design of the environment so that Things are grouped together in workspaces if they are meant to be used together by Agents.
- Persist and load Agent code to execute or modify during a later session. However, the Web IDE does not save Agent programs within a session, and reloading a page causes the programs to be lost.
- Configure Agents to use authentication workflows when using interaction affordances that are provided by devices and services. However, not all authentication workflows are supported by the current version of our

[22] https://developers.google.com/blockly

platform – our current Agents are limited to using key-based authentication methods (e.g., Basic, api-key) where the relevant key is put into the HTTP header or the query string.

- Manipulate the result of interactions with affordances in JSON format[23].
- Define MAS configurations and store them for later reuse.
- Submit a MAS configuration for execution to the Runtime Orchestrator.
- Add and remove Agents to/from a currently running MAS.

At the time of submission, only the HTTP protocol is supported as an interaction protocol between Agents and Things. There is, however, no conceptual problem to extending our platform to any protocol that has a W3C WoT TD binding. Furthermore, our system requires interaction affordances of Things to expose input and output schemas to enable Domain Experts to correctly map relevant (non-semantically-annotated) outputs to inputs. Further technical limitations of the current version of the platform include that write operations on (writable) properties of W3C WoT TDs as well as the handling of events that originate from Thing event affordances are not supported.

In the course of developing the Web IDE, we furthermore developed a Hypermedia MAS Explorer that discovers and parses W3C WoT TDs in the Hypermedia MAS Infrastructure and generates a user interface to discover affordances that are present in a workspace and interact with the respective devices and services. While this was later removed to focus the IDE on the configuration of Agents, we plan to re-integrate it with the IDE since it provides several benefits towards users of the IDE, e.g. discovering and testing of service interfaces.

3.2.2 AGENT RUNTIME ENVIRONMENT

The Runtime Environment is a composition of the Storage Manager, Runtime Orchestrator and the Runtime Service. The Storage Manager is responsible of storing the block program, the generated Agent code, and the defined MAS configurations; based on these building blocks, the Runtime Orchestrator can submit required Agents to be executed within a Runtime Service. The application that runs the Agents is based on JaCaMo REST[24] which provides capabilities to modify MAS at run time, where we modified the endpoint to dynamically add new Agents. We furthermore integrated a special client artifact and a JSON manipulation library to extend the capabilities of Agents within JaCaMo in accordance with the code generated from the blocks in our Web IDE.

At the time of submission, the implementation of the Runtime Orchestrator is limited to a single Runtime Service, which means that at the moment only a single MAS can be executed. While this limitation is not relevant in the context of the IntellIoT project (since, anyway, multiple different runtimes may be executed for the different use cases), it will be removed in future iterations of our platform. Furthermore, reflecting limitations from the Web IDE, our implementation does not support all possible authentication workflows. Also, in the current implementation, the Runtime Environment has no knowledge of the W3C WoT TDs that a currently running program is utilizing, and Agents can at the moment not directly perceive Things they interact with, thereby blocking them from reacting to Thing events. Both these limitations have implications on the project and need to be overcome in future iterations of our platform.

3.3 Interoperability Box: Device Heterogeneity in IntellIoT

Heterogeneous devices and applications with their own, private, semantics are prone to create islands of IoT functionality that constrain the full potential of the NG IoT. IntellIoT will tackle these open issues through compliance with emerging standards (e.g., by the W3C WoT) and a dedicated Interoperability Box (IO Box) that can resolve

[23] https://datatracker.ietf.org/doc/html/rfc7159
[24] https://github.com/jacamo-lang/jacamo-rest

incompatibility constraints. Deployed in the IoT environment, these components enable the mediation between different: communication means (e.g., 5G or Wi-Fi) for technical interoperability, protocols (e.g., MQTT or CoAP) for syntactic interoperability, and vocabularies (e.g., legacy models or W3C Thing Description[25]) for semantic interoperability based on defined semantic frameworks [14].

IntellIoT furthermore promotes secure data sharing while protecting users' privacy (for a detailed description of the pertinent enablers, we defer the reader to deliverable D4.4). Data will only be exchanged with a service in case it is needed, to prevent unnecessary collection or misuse of private data. DLTs and smart contracts will be made accessible by IoT devices under resource constraints to show transparency of performed actions in the three use cases: e.g., a farmer can verify identity and legal possession of a field as part of the access authorization to a tractor operator service, or a mobile robot arm in the manufacturing plant stores transactions with third-party machinery in the DLT for a pay-per-use scheme.

The Hypermedia MAS architecture is based on the notion of artifacts, which are either physical or virtual resources controlled by an agent using a programmatic interface (e.g., HTTP, MQTT, or others) as described in their respective W3C WoT TD. This allows for a high-level definition of an IoT application that can take advantage of and orchestrate a large number of readily available IoT devices to meet a specific goal. IoT devices, though, are very diverse in terms of capabilities and employed communication methods and there is no standardized method that they adhere to to exchange data.

For example, there are simple wired sensors that employ primitive communication mechanisms such as I^2C, digital / analog IO or PWM in order to transfer raw data, while others may use highly complicated and sophisticated wired or wireless networking interfaces (BLE, WiFi, Ethernet etc). On top of these, depending on the devices' computational capabilities, the data they exchange may be organized in different formats (binary data, plaintext, JSON, XML and others) and within these formats a number of alternate semantics for each piece of data may be used. Therefore, it is easily understood that in order to orchestrate all those different devices and protocols under a single IoT application requires a common communication language to allow interoperability. The means to achieve this common understanding is by using a description in the form of W3C WoT TD and realize a mechanism to provide the proper translation between a physical device interface and such a description.

The means to achieve this common understanding and proper communication at the application level is through the abstractions offered by Hypermedia MAS and the related communication mechanisms it uses. Devices that have adequate computational and networking capabilities are able to execute the proper software tasks to comply with those requirements. However, a significant number of IoT devices with limited capabilities may require an intermediate node (assuming the role of a gateway) that will manage the interoperation between the IoT application and them. This gateway will have the necessary physical interfaces to communicate with those limited resource devices and the software stack necessary to bridge these devices with the IoT application. The software stack has two main components. The first one involves the device-specific low-level drivers that represent how the local system views and accesses the device. The second component (called IO Box) provides a translation between this local view and a higher level W3C WoT TD description that the remote IoT application may use to communicate with the device. In this way, devices with protocols that do not yet have a W3C WoT TD binding become available. Once the device description has been advertised to the HyperMAS, the communication API is defined and the IO Box implements the specified HTTP REST API to enable the handling of the device from the IoT application. Figure 3 demonstrates this architecture.

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

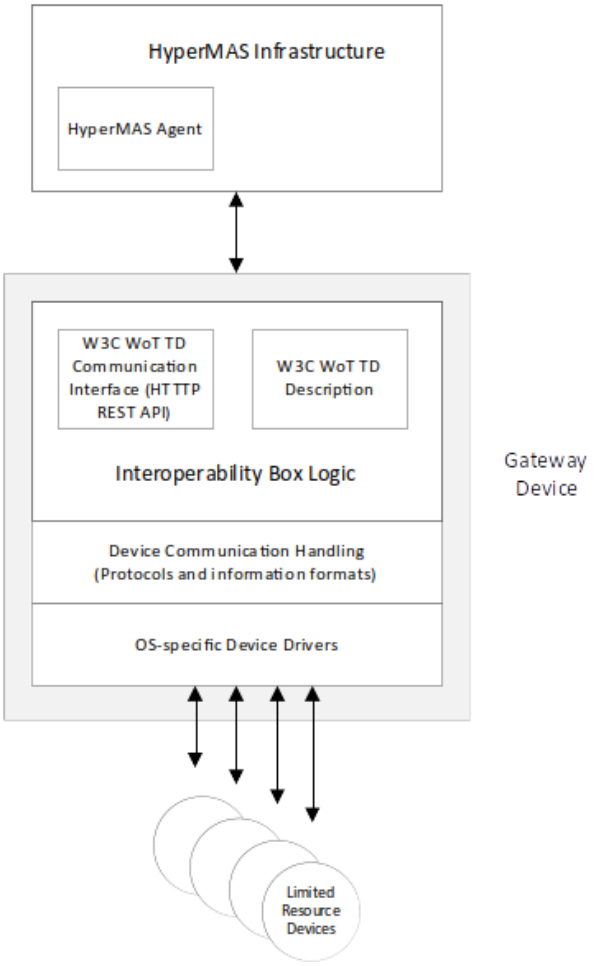

Figure 3. High-level, logical architecture of Interoperability Box

The Interoperability Box as a software component requires a Linux-capable device, with proper interfaces with all devices that have limited resources and need its services in order to communicate with the IoT application. A common example of such a device is the Raspberry Pi, which is the chosen device for demonstration purposes in IntellIoT.

The first version of the IO Box is implemented with a number of mocked devices for testing purposes. These are virtual devices that emulate actual physical character devices (examples of such devices are all devices that are accessed through serial interfaces, like I2C). The mocked devices that have been implemented emulate a temperature sensor and a switch. For each device, the IO Box defines a user-level driver that abstracts the device details. The user-level driver operations are mapped to the defined HTTP API to complete the communication path. A demo instance is up and running on our premises and it is reachable through the public internet. An example GET request on `http://<IOBoxGateway_address>/temperature` from an external entity will trigger a response from the virtual temperature sensor similar to this:

```
{
  "degrees":-7.808579856712106,
  "unit":"celsius"
}
```

Through this simple example, it can be seen that an external entity (an IoT application) that does not have any specific knowledge of the device that provides temperature sensing, is able to communicate with this device and retrieve the information that it requires. The switch mock device, unlike the temperature sensor, is not only read, but can accept and respond to commands. Therefore, a PUT request can be used to alter the switch state. Supposing that the switch is in a disabled state, then a GET request (`http://<IOBoxGateway_address>/switch`) from an external entity will trigger the following response:

```
{
  "enable":false
}
```

Sending the following PUT request:

```
curl -X PUT -d '{"enable": true}' http://<IOBoxGateway_address>/switch
```

will prompt

```
{
  "enable":true
}
```

and a subsequent GET request will verify that the switch is indeed in enabled state by returning the same response:

```
{
  "enable":true
}
```

In future work, support for real-world devices is going to be added. For example, support for MiroCards [15] has been planned, which will be employed in the manufacturing use case (see Section 4.3, below). The MiroCard is a batteryless device and as such it satisfies the description of a constrained device.

4 HYPERMEDIA MULTI-AGENT SYSTEMS IN THE INTELLIOT USE CASES

This section explains the role of the components of Task 3.1 across the use cases of the IntellIoT project. Since, as described in Section 2, our Hypermedia MAS is particularly useful in scenarios that involve complex integrations of heterogeneous devices and services that may change their syntactic and semantic interfaces at run time, the most relevant target use case for the discussed components is UC3, Manufacturing, and we therefore emphasize UC3 in the context of this document as well. However, as one claim of our systems is that they are broadly applicable and support the open integration of heterogeneous components, the two main components of Hypermedia MAS as well as the IO Box are utilized across all three use cases of IntellIoT.

4.1 Use Case 1: Agriculture

In IntellIoT's Agriculture use case, we explore increased autonomy of farming vehicles. Equipped with local sensing and decision-making components, these vehicles should be enabled to roam a field in a pre-defined pattern and carry out their agricultural functions while being able to avoid obstacles locally through an on-board machine-learning system that keeps training at run time and having the possibility to escalate to (remote) human drivers who take control of the vehicle in situations that this local "artificial intelligence" cannot decide. Towards these goals, in UC1 we deploy a prototype of a self-driving farming vehicle that is integrated with IntellIoT's collaborative IoT, human-in-the-loop, and trustworthiness technologies.

In the context of T3.1 and UC1, we emphasize the flexible configuration of a farming system by a Domain Expert, where equipment (i.e., the concrete vehicle to be used) is selected at run time given the requirements of a concrete goal – e.g., fertilization of a field requires a vehicle with different implements (e.g., a spreader) than ploughing of a field does. Furthermore, similar to UC3 (where this idea is explored in more depth), UC1 demonstrates the capability of a Hypermedia MAS to coordinate different types of intelligences – the local machine-learning system on the vehicle (to avoid obstacles locally) and the remote human expert that can take control of a vehicle and interact with it through a virtual reality (VR) system if the local system fails, where the Agents in the Hypermedia MAS serve to decouple vehicle, local AI, and remote driver.

At the time of submission of this deliverable and in the context of UC1, the Hypermedia MAS Infrastructure and the Web-based IDE for Hypermedia MAS are being tested through integration with two prototypical mobile robots in the HSG laboratory that represent the autonomous tractors of this use case. These robots also expose TD-described HTTP interfaces[26] that has been created to mock functions of the autonomous tractor, and we are currently upgrading them to use a global spatial reference (to mock GPS locations). The two mobile robots, along with other devices in the laboratory (see Section 4.3, below) expose W3C WoT TDs, are registered to our infrastructure, and can consequently be configured using the no-code environment. We have already successfully configured and executed Agents that control these robots. As a next step towards achieving UC1, we will finalize – together with the use case partners – the interfaces and interface descriptions for the autonomous tractor and our next goal is that an Agent that has been configured using our Web-based IDE actually controls a real autonomous tractor.

4.2 Use Case 2: Healthcare

[26] See https://raw.githubusercontent.com/Interactions-HSG/example-tds/main/tds/spock-tractorbot.ttl for an example.

Within IntellIoT's Healthcare Use Case, we investigate collaborative semi-autonomous systems with humans in the loop that leverage the IntellIoT technologies to provide more accurate and timely information about the health status of patients to clinicians. Focusing on heart failure patients, this UC targets at enabling patients to take a central role in their own recovery process while being monitored by clinicians. Timely and reliable notification of clinicians is required in two cases (see D2.1):

1. When the local AI detects a potential health emergency, predicts a deterioration that requires the intervention of the clinician, receives a patient request that involves reaching out to a clinical expert, or when the defined workflow defines the involvement of the clinician.
2. When the system encounters an exception that it does not know how to address, or in case of technology failure.

Within IntellIoT and in the context of Task 3.1, Agents in Hypermedia MAS will be configured using our system to take over the task of notifying the clinician in either of these cases, and that it is thus this Agent that is configured with the decision logic for escalating such alarms. This setup allows to decouple the alarm-originating system (i.e., the local machine learning system or the patient) from the alarm-receiving systems (i.e., the notification devices, e.g., smartphones, of clinicians who can be reached via different channels, emergency centers, etc.) and will allow simple extension of the system with further escalation modes at run time.

To permit this setup, the system will have access to several notification systems whose interfaces are described using W3C WoT TDs and that permit clients to stay updated about the delivery status of a notification (ideally also providing reading confirmations to clients). The IntellIoT project will implement several notification services that can be configured by Domain Experts to fit their application requirements.[27] Given these available notification interfaces, the Domain Expert then configures one or multiple Agents using our provided no-code environment (see Section 3.2), where the high-level creative input concerns the mapping of events to alarm requirements, the concrete ordering of service usage or the decision to have Agents use all service in parallel, and the sensitivity to time-outs; configured Agents run in the Hypermedia MAS Infrastructure (see Section 3.1) and interact with the W3C WoT TD-described services to notify clinicians. Together with the DLT journaling systems from T3.4, this system also creates transparency about the interactions between Agents and clinicians. Towards implementing this scenario over the coming months, we still need to agree on the level of detail that the Agent (and, hence, the Domain Expert) may access about the content of notifications. This forms a trade-off between the relevant information for Agents to take informed decisions (e.g., about which doctor to escalate to, and the priority of the escalation) and the amount of personal data that is exposed beyond the local machine learning system.

4.3 Use Case 3: Industrial Manufacturing

To recapitulate from D2.1, one main aim of UC3 is to enable flexible and individualized (up to lot-size one) production, where we consider a shared manufacturing-as-a-service scenario. Individual machines and services are provided by different machine vendors. From the perspective of T3.1, UC3 has the highest potential to showcase the potentials of

[27] After consultation with the end users in UC2, we have agreed to provide such interfaces to an instant messaging service (e.g., the *Viber* messaging service which is popular among doctors in Greece) and to at least one service that enables Text Messaging in our system (for broad applicability, e.g., through *Twilio*). All these interfaces will be W3C WoT TD-described. Because of the Hypermedia MAS-induced flexibility and that interface metadata, however, the system remains extensible with any notification service.

all components in T3.1: The Hypermedia MAS Infrastructure, the no-code programming interface for Hypermedia MAS, and the IO Box.

The main feature from T3.1 that we emphasize within UC3 is the usage of Hypermedia MAS to decouple the individual devices in the UC, thereby increasing the flexibility (i.e., individual functional components can get replaced and upgraded) together with the usage of our Web IDE to enable Domain Experts to focus on tasks that require creativity – in this case, the integration of individual functional components in a complex environment – while not being required to engage in the low-level engineering of the system (e.g., the configuration of network addresses or setting-up of component internals). Like UC1, UC3 is furthermore ideally suited to demonstrate how the high-level coordination capability of Agents that are deployed within our Hypermedia MAS (and that are programmed by Domain Experts) integrates with low-level local machine learning (to compute appropriate grab and engrave spots on workpieces) and how they can, at run time, involve human agents to solve any issues that the Agents and local machine learning fail to overcome. Within UC3, we can furthermore ideally demonstrate the integration of resource-constrained devices via the IO Box that we will use for the flexible integration of low-power (even batteryless) equipment to monitor the individual components of the UC.

Within UC3, the following devices and services are relevant in the context of the interoperable Hypermedia MAS and therefore are required to expose W3C WoT TD-described interfaces – we briefly review the role of these entities in the context of UC3:

- **Robot Controller**: The robot device in UC3 is coupled with a video camera that exposes a video feed. Towards agents, the device is hidden behind a "Robot Controller" service that also is aware of the camera and exposes information to allow interfacing with the camera alongside the TD-described API to the robot; it furthermore exposes a direct handle to control the robot without going through the Robot Controller (this is required for the HIL scenario in UC3).
- **Milling Machine Service**: UC3's milling machine along with its coupled clamping mechanism is likewise abstracted behind an API that exposes a TD.
- **Laser Engraver Service**: In the same way as for the Milling Machine Service, the Laser Engraver, its lifting device, and control abstractions are abstracted behind a TD-described API.
- **Grab Spot AI** and **Area Detection AI (AIs)**: This service consumes the camera stream and is responsible for computing grabspots from this stream (when picking up workpieces) and for fine-tuning the engraving location (after positioning the workpiece). The TD-described API to the AIs allows agents to pass a handle to the camera stream and to receive grab spots and engrave spots from it.
- **Human-in-the-Loop Service (HIL Service):** Very similar to the AIs, the HIL Service encapsulates human (rather than artificial intelligence) support. It exposes a TD-described API that allows passing handles to the camera stream and the robot device controller to the HIL Service and receives back acknowledgements when the human has been (un)successful solving the problem.
- **Edge Orchestrator** (to be integrated in Cycle 2): The Edge Orchestrator will create and manage instances of Edge Applications only at run time while registering the functions of these applications using W3C WoT TDs that do not contain protocol bindings. This will permit Domain Experts to program Agents against abstract functional interfaces of Things that are only at run time deployed and bound to concrete APIs.

From this list, the functional coupling of the devices becomes visible; we followed the rule that (physical and virtual) entities that are expected to always change together remain coupled and abstracted behind a single API. In this way, for instance, the robot remains decoupled from the AIs as well as from the HIL Service since, after moving the camera via the robot, it is *the Agent* who triggers the AI to compute the grab spot or the engrave spot, and it is likewise *the Agent* who triggers the HIL to get human help. This permits the AI and HIL to work with multiple robots and ideally demonstrates IntellIoT's decentralization and flexibility idea. As part of UC3, a Domain Expert will use our no-code Hypermedia MAS programming environment to discover the interfaces of these services and mash them up within one or multiple Agent programs that are then deployed for execution to the Hypermedia MAS Infrastructure.

Most interestingly, towards the individual scenarios of UC3, the Hypermedia MAS is configured by the Domain Expert to coordinate the following tasks:

1. Coordination between AIs and Robot Controller

To find an appropriate grabspot, the Domain Expert will configure Agents to first move the robot to an appropriate viewpoint for a workpiece, then trigger the AIs to compute a grabspot, and then use this grabspot to move the workpiece to the engraver or milling machine.

2. Coordination between AIs and Engraver

To find the correct engravespot on a workpiece that has been inserted into the engraver, the Domain Expert will configure Agents to first move the robot to an appropriate viewpoint for the engraver, then trigger the AIs to compute a corrected engravespot, and then instruct the engraver to engrave at this spot.

3. Coordination of the Escalation to the Human Operator

It might occur that the AIs are unable to compute an appropriate grabspot. To handle this case, the Domain Expert will configure an Agent to handle such (negative) responses from the AIs and then to use the HIL Service to establish a direct control feedback between a human operator (wearing a VR headset), the robot, and the camera. This involves not only triggering the HIL Service, but also providing the HIL Service with the required information to establish this feedback link, i.e., a handle to the low-level robot controller and a handle to the camera stream – both are obtained from the Robot Controller's API. After the (manual) moving of the robot to the appropriate location is successful, the Agent will furthermore be configured to provide the AIs with the required training data to learn this specific situation.

In all of these scenarios, we have taken care that the individual devices remain decoupled and are only integrated by the Domain Expert – that is, all integration is possible in the provided no-code environment and using the individual services' W3C WoT TDs. Within these scenarios, this is ideally suited towards demonstrating IntellIoT's Collaborative IoT and Human-in-the-Loop pillars and, together with the integration of W3C WoT TDs with T3.4, also demonstrates the project's Trustworthiness pillar.

At the time of submission of this deliverable and in the context of UC3, the Hypermedia MAS Infrastructure and the Web-based IDE for Hypermedia MAS are deployed at the laboratory facilities of HSG. They have access to several devices (2 stationary industry-grade robots, a simple pick-and-place robot, and several sensors) in the laboratory that are described using W3C WoT TDs and consequently the Web IDE can be used to configure IntellIoT applications in this context. We have already successfully configured and executed Agents that use the provided device functions. Furthermore, these systems are integrated with a mock API of the Engraver that is supplied by our use case partners. Our next step is to integrate this setup with the actual UC3 devices that are deployed at the use case partners. In Cycle 2 of the IntellIoT project, T3.1 within UC3 will focus on the integration of the Edge Orchestrator (see D4.1) for run-time management of the concrete APIs and on the integration of the IO Box in this UC. Towards the Edge Orchestrator, we are currently planning the concrete integration points and information flows – this will ideally demonstrate the usage of abstract W3C WoT TDs (i.e., W3C WoT TD Templates together with input and output schema information) as standardized coordination points for the orchestration of resources on the edge.

Towards the integration of resource-constrained devices through the IO Box, we will in Cycle 2 enable and demonstrate the run-time extension of our system with a batteryless temperature sensor to monitor the Laser Engraver Service and detect overheating. Concretely we will integrate a MiroCard [16], which is ideally suited to demonstrate the capabilities of the IO Box, even going beyond low-power devices and into batteryless, energy-harvesting devices. The MiroCard brings an accelerometer as well as temperature, humidity, and ambient light sensors and communicates via BLE. The IO Box would then take the role of a bridge that exposes the MiroCard's

sensors as W3C WoT TD-described services, so that Domain Experts can use our no-code environment to integrate MiroCard sensors at run time. We propose to create a simple use case where an Agent refrains from using the Laser Engraver Service if the detected temperature crosses a (Domain-Expert-configured) threshold. Due to the decoupling of the individual services that we achieve through our Hypermedia MAS, no other component of UC3 needs to be aware of the MiroCard or of its integration, thereby ensuring (and demonstrating) the openness and extensibility as well as the flexibility of IntellIoT. We can furthermore extend this scenario to include detection of motor activity (through the MiroCard's low-frequency accelerometer).

5 CONCLUSIONS AND FUTURE WORK

The individual components from Task 3.1 in the IntellIoT project have been designed and implemented according to the work plan of the project, and laboratory demonstrators of the components are available. Several of these components have been integrated in our laboratory environment and we are today able to mock two of the three use cases of the IntellIoT project (while not foreseeing hurdles to mock the Healthcare use case as well). The positioning of the components with respect to their integration in the individual use cases in WP2 (see Section 5) is finalized and we are looking forward to the integration with other components across the project that will be driven in WP5.

These integrations will be performed next, and we plan to have no-code-configured Hypermedia MAS interact with the UC1 tractor, the UC2 clinician interfaces, and the UC3 industrial machinery in two months after the date of submission of this deliverable, where constrained devices that are connected through the IO Box are present in at least UC3. As part of Cycle 2 of IntellIoT, we plan to evaluate the performance of our no-code environment, further extend the Hypermedia MAS infrastructure with discovery and search components, and integrate our components with the Edge Orchestrator as well as the distributed-ledger-based journaling system of IntellIoT.

REFERENCES

[1] Ciortea, A., Mayer, S., Michahelles, F.: Repurposing Manufacturing Lines on the Fly with Multi-agent Systems for the Web of Things. In Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, pp. 813–822, 2018. https://dl.acm.org/doi/10.5555/3237383.3237504

[2] Kovatsch, M., Hassan, Y. N., Mayer, S.: Practical semantics for the Internet of Things: Physical states, device mashups, and open questions. In Proceedings of the 5th International Conference on the Internet of Things, pp. 54-61, 2015. https://doi.org/10.1109/IOT.2015.7356548

[3] Ciortea, A., Boissier, O., Ricci, A. (2019) Engineering World-Wide Multi-Agent Systems with Hypermedia. In: Weyns D., Mascardi V., Ricci A. (eds) Engineering Multi-Agent Systems. EMAS 2018. Lecture Notes in Computer Science, vol 11375. Springer. https://doi.org/10.1007/978-3-030-25693-7_15

[4] Ciortea, A., Mayer, S., Gandon, F., Boissier, O., Ricci, A., Zimmermann, A.: 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, pp. 1659–1663, 2019. https://dl.acm.org/doi/10.5555/3306127.3331893

[5] Ricci, A., Piunti, M. & Viroli, M. Environment programming in multi-agent systems: An artifact-based perspective. Autonomous Agents and Multi-Agent Systems 23, 158–192 (2011). https://doi.org/10.1007/s10458-010-9140-7

[6] Ciortea, A., Boissier, O., Ricci, A. (2019) Engineering World-Wide Multi-Agent Systems with Hypermedia. In: Weyns D., Mascardi V., Ricci A. (eds) Engineering Multi-Agent Systems. EMAS 2018. Lecture Notes in Computer Science, vol. 11375. Springer. https://doi.org/10.1007/978-3-030-25693-7_15

[7] Ciortea, A., Mayer, S., Bienz, S., Gandon, F., Corby, O. : Autonomous Search in a Social and Ubiquitous Web. Personal and Ubiquitous Computing, 2020, https://doi.org/10.1007/s00779-020-01415-1

[8] Bienz, S., Ciortea, A., Mayer, S., Gandon, F., Corby, O.: Escaping the Streetlight Effect: Semantic Hypermedia Search Enhances Autonomous Behavior in the Web of Things. In Proceedings of the 9th International Conference on the Internet of Things, Article 28, 2019. https://doi.org/10.1145/3365871.3365901

[9] Mayer, S., Inhelder, N., Verborgh, R., Van de Walle, R., Mattern, F.: Configuration of smart environments made simple: Combining visual modeling with semantic metadata and reasoning. In Proceedings of the International Conference on the Internet of Things, pp. 61-66, 2014. https://doi.org/10.1109/IOT.2014.7030116

[10] Bau, D., Gray, J., Kelleher, C., Sheldon, J., Turbak, F.: Learnable programming: blocks and beyond. Commun. ACM 60(6), 72–80, 2017. https://doi.org/10.1145/3015455

[11] Yadagiri, R. G., Krishnamoorthy, S. P., & Kapila, V.: A blocks-based visual environment to teach robot-programming to K-12 students. Computers in Education Journal, 8(2), 24-32, 2017. https://nyuscholars.nyu.edu/en/publications/a-blocks-based-visual-environment-to-teach-robot-programming-to-k-2

[12] Shoham, Y. Agent-oriented programming, Artificial Intelligence, 60(1), 51-92, 1993. https://doi.org/10.1016/0004-3702(93)90034-9

[13] Rafael H. Bordini, Jomi F. Hübner, and Michael Wooldridge: Programming Multi-agent systems in AgentSpeak using Jason. John Wiley & Sons, 2007. https://onlinelibrary.wiley.com/doi/book/10.1002/9780470061848

[14] Hatzivasilis, G., Soultatos, O., Anicic, D., Broring, A., Fysarakis, K., Spanoudakis, G., ... & Ciechomski, L. (2019, December). Secure Semantic Interoperability for IoT Applications with Linked Data. In 2019 IEEE Global Communications Conference (GLOBECOM) (pp. 1-6). IEEE.

[15] Gomez, A.: On-demand communication with the batteryless MiroCard: demo abstract. In Proceedings of the 18th Conference on Embedded Networked Sensor Systems. ACM, pp. 629–630, 2020. https://doi.org/10.1145/3384419.3430440